A Script-driven Simulation Tool as an Improvement to the Education in Web-based Virtual Learning Environments

Diploma Thesis

Torsten Klie Christoph Schalong

TU Braunschweig
Institute of Economics
Department of Business Administration, Business Computer Science, and Information Management
Prof. Dr. Stefan Voß

Advisor: Dipl. Wirtsch.-Inform. Torsten Reiners

December 16, 2002

Abstract

Simulation is an important technology with has a high practical relevance. If used properly simulation will help to avoid errors in planning processes. Undetected errors in planning processes can lead to very high costs. The demands on the users of this technology are high. Therefore, teaching in the subject Simulation is a very important issue.

Actual courses in Simulation are quite theoretical and lack practical examples. This work will describe the applying of learning methods, for example tutorials, on the subject Simulation for enhancing teaching in the sense of illustrating the drily subject. Therefore, a simulation tool that has with a self-explanatory GUI is created which will use interactive components to communicate with the learners. The tool is based on a component oriented discrete event simulation engine which allows the learner to build simulation models by using provided simulation components. The teacher can create demonstrations of simulation models and interactive tutorials. To accomplish this, an XML based script language is developed which can be used to define demonstration steps and exceptions.

Due to the nature of the developed tool as a prototype, major aims are extensibility and modularity. Simulation components can be plugged in the system at runtime, for example.

Preface

After first contacts with a professional simulation tool during a course in the subject Simulation, we developed an interest in creating a simulation tool that is easier and more appropriate to use teaching. Sixth months later, we got the opportunity to implement such a simulation tool in cooperation with other students as part of a course in information management. With our tutor Torsten Reiners, we developed lots of ideas which could not all be implemented in only one semester. We are glad that Torsten Reiners offered us to write our diploma thesis about the development of new features (learning supportive methods) for the simulation tool, because we could continue the development of "our" tool.

We would like to thank everybody who helped and supported or work: Prof. Dr. Stefan Voß for giving us the opportunity to accomplish this work at his department, Torsten Reiners for being a faithful source of new ideas, the former team which the development started with: Oleg Dück, Marco Hellmann, Jörg Hevers, Florian Maul, Nils Oppermann and Björn Rösemeier; Sandra Bangewitz and Manuela Bernhard for developing a simulation component (an assembly station) and showing that our concept of modularity is working, Jörg Hevers for providing a very stable CVS server and our parents for supporting us financially. Thanks to Steve Kent, Aarti Raßmann and Torsten Reiners for proof reading our work and helping us with some problems with the English language. Last but not least, we would like to thank our girlfriends Maria-Alba Bosch i Borràs and Silke Langner for giving us moral support.

Declaration of Independence

Herewith I declare on oath that the presented diploma thesis is done independently and unassisted using only the referred literature.[1]

Torsten Klie

Herewith I declare on oath that the presented diploma thesis is done independently and unassisted using only the referred literature.[1]

Christoph Schalong

This diploma thesis has been written as a cooperative work. Torsten Klie was responsible for the chapters focused on simulation (Chapter 2.3) and on script instructions (Chapters 3.4, 4.4, 4.6 and 5.2.3). Christoph Schalong was responsible for the chapters focused on e-learning (Chapter 2.1) and on the exception concept (Chapters 3.5, 3.6, 4.5, 5.2.5 and 5.2.6). The other chapters were developed in cooperation.

[1] Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe nur unter Verwendung der angefürten Literatur angefertigt habe.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simulation is one of the most important technologies that is used in the planning of production systems, in enterprise logistics and in production planning.[1] The demands on the users of this technology are high. Simulation works on the GIGO[2] principle.[3] Thus, if the users of simulation do not know how to create, use and interpret simulation models properly the results of simulation studies will be false. Undetected errors in the planning process can cause huge economic damages. If a factory has been built and the company realises that the desired output cannot be achieved with the installed machine layout, immense costs will arise. If it is used properly, simulation will help to avoid errors during the planning process.

A fundamental problem of simulation is the lack of simulation experts who fulfil the demands described above.[4] With regard, teaching in Simulation as a subject is a very important issue. Unfortunately, courses in simulation are quite theoretical and lack practical examples. Existing simulation tools (for example eM-Plant[5]) are too complicated for using in courses, because the learners would have to spend too much time in learning how to use these applications rather than learning how simulation works.

The aim of this work is to develop a simulation tool that can be integrated

[1] see [KG95] p. 1
[2] Garbage In – Garbage Out
[3] see [VG01] p. 180
[4] see [KG95] loc. cit.
[5] see [Tec02]

into an existing virtual learning environment. The tool must have an easy to use, self-explanatory GUI[6]. It should help visualising simulation principles which are usually hidden by simulation tools. Learners will have the possibility to create simple simulation models easily. Furthermore, a demonstration facility is desired. Therefore, a script language is needed which the teachers can create presentations for demonstration purposes with. XML[7] will be used as the file format for script files. The simulation tool will also interact with the learners. Thus, the system should be able to react to the learner's actions, that means, learners shall be informed when they commit errors. The script language will be extended so that it will be possible for teachers to write tutorials that include tasks that should be solved by the learners. The system shall be able to check whether the learners have solved the task. The simulation will give useful feedback. Therefore, an exception handling will be implemented.

This work does not start the development of a new programme from scratch. It is based on the Java application SimTool[8] being implemented in the winter semester 2001/2002 as part of a course in information management which we participated in.

1.1 Structure of this diploma thesis

- *Chapter 2*: Basics
  This chapter will give a short overview about learning theories in general and e-learning in particular. A selection of existing virtual learning environments will be presented (2.1). Furthermore, the basics of simulation and probability distributions(2.3) will be described.

- *Chapter 3*: Concepts
  This chapter will present the concepts of SimTool, especially the concepts of the script language (3.4) and the exception handling (3.5). Furthermore, it will describe the configuration options (3.2) and the possibilities of using the modular design of SimTool (3.3).

[6]Graphical user interface
[7]Extensible Markup Language
[8]see [DHH+02]

- *Chapter 4*: Aspects of Implementation
  In this chapter, special aspects of the implementation are discussed. After describing the initialisation (4.2), the invocation of script commands (4.4) and the raising of exceptions (4.5) in detail, it will describe how script commands can be added (4.6). An example is used for the explanation.

- *Chapter 5*: A Sample lesson
  This chapter will describe the implementation of a sample lesson that consists of a tutorial.

- *Chapter 6*: Conclusion
  This chapter will summarise this diploma thesis (6.1). Furthermore, possible extensions will be presented (6.2) as well as ideas for lessons (6.3).

1.2 Naming conventions

The following formatting conventions are used in this document:

- `Typewriter`: used to mark names of Java objects, classes and packages[9] This style is also used for XML code and Java code.

- `<Typewriter>`: refers to XML tags

- *Italic*: used to emphasise important text phrases and in enumerations and lists

- **Bold**: used for headings and in tables

- Small capitals: used for names of people

[9]Note that in case of a Java class or Java object the full qualified name with additional package information will be presented as a footnote at the first occurrence of the class or object in a section.

Chapter 2

Basics

2.1 E-learning

2.1.1 Introduction

E-learning can be defined as learning that is supported electronically by the use of new media like the Internet, multimedia computers, videos, DVDs, etc..[1] The focus of e-learning is the learning process that occurs in virtual learning environments. It is possible to classify e-learning systems according to three dimensions:

1. *Temporary dimension*: synchronous/asynchronous
   Are teachers and learners communicating at the same time?

2. *Locational dimension*: online/offline
   Is a network connection necessary?

3. *Controlling dimension*: teacher/student
   Who controls the system? Can the student influence the order of the lessons?

[1] see [Kre02] p.25

Computer-based Training (CBT) and Web-based Training (WBT) are two important labels for learning programmes. CBT systems are flexible in temporary and locational dimensions. The student does not have to communicate directly with the teacher, because the learning content is stored on a storage media. The difference between CBT and WBT systems is the storage of the files. In WBT systems, files are distributed via Internet (or Intranet respectively).

Learning programmes can be classified in the following way:

- *Practice and drill programmes*:
  Small and simple tasks are repeated often.
- *Tutorials*:
  Learners are instructed step by step.
- *Intelligent tutorials*:
  Decisions of a teacher are imitated.
- *Learning games*:
  Learning is achieved by playing.
- *Simulation programmes*:
  Real situations are simulated.
- *Hyper-media programmes*:
  Free navigation in the programme is possible.

2.1.2 Learning theories

In the following, three perspectives on learning will be discussed. They will try to explain why and how humans learn and will be presented in the order of their historical appearance.

Behaviourism

Learning is a quite permanent change in a person's behaviour.[2] This change needs to be the consequence of experience. Changes in a person's behaviour

[2] see [AAS+93] p. 253

resulting from maturation are excluded from this definition. The changes are not completely permanent because human beings forget what they have learned with the pass of time. Also, the facts which will be learned later might influence the already learned facts.[3]

The term *behaviourism* was formed by WATSON. A behaviourist works with the measurable aspects of the behaviour. A person's behaviour is assumed to consist of reactions. It was the aim of the behaviourists to deduct general principles which explain the relations between stimuli and reactions.[4]

Fundamental work was done by PAWLOW in 1904 who discovered in experiments with dogs that certain stimuli cause certain reactions after a while. In his experiment, he investigated the salivation flow of dogs. For stimulating the salivation, he put food directly into the mouths of the dogs. The resulting salivation is an unconditioned response because no learning is involved. Later, after the process had been repeated multiple times, he noticed that the salivation of the dogs was stimulated by viewing the food or the person that gives them food. This salivation is a conditioned response, the view of the food or the person is the conditioned stimulus. These experiments were the basis of the classical conditioning.[5]

THORNDIKE's work is based on reinforcement. The influences of external factors were described in his law of exercise which says that relationship between stimuli and reactions are reinforced if they are exercised often within a short period of time.[6]

SKINNER introduced the concept of operant conditioning which consists of the following aspects:[7]

- *Contingency of behaviour*:
  Contigencies of behaviour express relations like "if a then b".

[3] see [Zim95] p.264
[4] see [Lef94] p. 17
[5] see [AAS+93] p. 254
[6] see [Lef94] p. 26
[7] see [Zim95] p. 278

- *Reinforcer*:
  Reinforcers are events that may support reactions if they occur in a contingent relationship.
  - *Positive reinforcer*:
    Positive reinforcers are stimuli that increase the probability of the occurrence of a reaction.
  - *Negative reinforcer*:
    Negative reinforcers are stimuli that increase the probability of the occurrence of a reaction if removed from a situation (for example, electric shocks, extreme heat, etc.).
  - *Removal*:
    Removals mean that the initial contingency is no longer valid.
  - *Punishment*:
    Punishments are the use of aversive stimuli to lower the probability of reactions.
- *Discriminative stimuli*:
  Discriminative stimuli indicate the presence or absence of reinforcers.

The need for immediate feedback can be deducted from the behaviouristic learning theories when applying them to the design of e-learning systems.[8] Good answers of the students shall be rewarded. For generating this feedback, clearly defined learning targets are necessary. E-learning systems that follow these guidelines cannot be used to teach complex issues, they are only useful for well structured subjects.

Cognitivism

Cognitivism became popular in the early 1960s. It describes that organisms gain knowledge of their environment via cognitions.[9] Cognitive processes can be distinguished from emotional and motivational processes. In the human context, cognitive processes are:

- Perception

[8] see [Kre02] p.48
[9] see [Ede00] p. 114

- Imagination
- Thinking
- Judging
- Speaking

The realisation of relationships leads to an organisation and structuring of experience. The success of the learning process depends on the contents of these structures which are the consequence of former learning processes. The learner must participate actively in the process in order to create the structures.[10]

In the sense of e-learning, these cognitive processes must be supported by interaction between the user and the e-learning system. Tutorials and simulations fit these demands. They are more flexible than behaviouristic practice and drill programmes because the computer acts as a tutor who helps the learner to choose the right method to solve a given problem. Furthermore, these systems allow the specification of the learner's knowledge level, for example beginner or advanced user.

Constructivism

Constructivistic concepts which appeared in the early 1990s focus on the learner who participates actively in the learning process which he controls independently. Learning in this context means the construction of a subjective reality based on the experience of the student.[11] The knowledge of a teacher cannot be used by his students directly by transmitting the data (or information, respectively) which the knowledge is based on.[12] The students have to understand these data and information in order to learn successfully.

The learning situation must be authentic.[13] Therefore, tasks in an e-learning system must be designed problem oriented and according to the later application of the gained knowledge. Furthermore, cooperation and team work should be trained.

[10] see [Kre02] p. 49
[11] see [Ede00] p.287
[12] see [VG01] p. 14
[13] see [Ede00] loc.cit.

A learner should notice that there is more than one possible solution.[14] This can be ensured, for example, with simulations where the problem to solve is not given. Then it is the learner's task to develop proper strategies for unknown problems. The order of the lessons in the programme depends on the student. The computer serves as a cooperative and supporting coach.

2.1.3 Virtual learning environments

Virtual learning environments are a quite popular research field these days. Therefore, a lot of universities work on an e-learning system. In Germany, the BMBF[15] supports most of these projects financially. This section will present a selection of different systems. The following three approaches can be distinguished:

1. Multidisciplinary systems
2. Discipline specific systems
3. Combined systems

Multidisciplinary systems

Virtual learning environments are focused on the presentation of learning content. Multidisciplinary environments are designed to present learning content related to more than one subject. These systems provide simply to use modules for authoring learning lessons. They are designed to work trans-sectoral. Examples of such trans-sectoral learning systems are virtual universities which aim to substitute the normal learning as currently provided by the universities. The systems normally provide a GUI which can be accessed via the Internet using a standard web browser.

Learning supportive methods are provided and used to make the learning content more navigable (hyper text, tree structure, learning plans, etc.). The motivation of the learner is one of the major focus points. Therefore, these systems provide easy to use GUIs which use animations and multimedia to

[14] see [Kre02] p.51
[15] see [Bun02]

present learning content. However, it is much harder to be motivated to a virtual learning on a virtual campus where learners actually sit alone at home and do not have any personal contact with other learners. In contrast, in classical universities learners benefit from group dynamic processes. Video conferences, virtual meeting rooms, white boards, etc. are provided by the systems of this category to compensate the missing communication among learners and between learners and teachers.

The focus on the presentation and the providing of communication possibilities reflect main goals that apply for concepts of learning environments: They should be created such that the engagement with the content provided by the system generates motivation. Systems should offer the possibility to assist the learner by the creation of his own learning plan. This goal is important for the systems described here, because learners and tutors do not have or have only rare personal contact. KERRES refers to a motivating system as a system that is able to abstract design patterns and perspectives of the learning content and present them in a way to the learner that allows the learner to work with them. In this context, a motivating system can be seen as a system that allows the user to gain knowledge by "playing" with it ("learning by doing").

Most of the systems in this category do not provide sufficient support to reach this goal. For example, achievement tests at the end of a lesson are only supported in a rudimentary way. This results out of the fact, that the platform that is used to create the system influences the didactic methods that will be provided by the system.[16] Systems that are used in this context are normally based on HTML[17]/Javascript platforms which are not very flexible in the sense of modularity and provided programming facilities. Other systems use more powerful platforms but the actual content of the e-learning system is restricted as it can only be created by tools provided by these platforms. The lesson authors can only create contents as well as the authoring tool will allow it.

[16] see [Ker99]
[17] Hypertext Markup Language

In case of an HTML based platform, external tools (Hot Potatoes[18], CBIquick[19]) are used to generate tests such as the following:

- Interactive multiple-choice
- Short-answers
- Jumbled-sentences
- Crosswords
- Matching/ordering
- Gap-filling exercises

However, these tools cannot be used to generate complex subject related test, which are too specific to be reflected by the system or generated automatically by generic tools.

The test (interaction) methods are intended to check the knowledge gained after a lesson. They are only usable to define the state of the user's knowledge in a static way. They can not be used during the actual learning process to support the user and to improve knowledge. This is especially the case because a test at the end of a lesson is only limited useful to rate the user according to his knowledge level or amount of learned content. A dynamic classification of learners according to their knowledge level is difficult.

An example of a virtual learning environment that fits in this category is described below.

VGU — Virtual Global University This virtual university is an example of a commercial system which offers certified MBI[20] courses. The university describes themselves as:

> The Virtual Global University is based on knowledge, experience and people from "real" universities in Europe. VGU's School

[18] see [Hal02]
[19] see [AMT02]
[20] International Master of Business Informatics

> of Business Informatics (SBI) has brought together top experts in the field. The school's faculty is a network of experienced professors from reputable universities in Central Europe. SBI's unique feature is this concentration of high-level knowledge and expertise in one institution.

[21] As already mentioned above, the focus of the system is on the presentation of learning content (video based courses, powerpoint slides, audio, etc.). The communication between students and teacher or among students is provided through the implementation of virtual meeting rooms. Test and interactive elements can be created by using external tools (see above). The system is based on the commercial e-learning platform WebCT[22].

Discipline specific systems

Systems of this category are intended for the use in a discipline specific environment. They try to deal with the main criticism of the multidisciplinary systems as systems that are mostly concentrated on the visualisation of existing content.[23] In contrast, the creation and the design of an e-learning project should be centred around a "didactic problem".[24] Such a problem has to be defined issues related for a single subject. The understanding of simulations is an example of such a problem.

Today, psychologic research and educational research ask for more open systems where the user can "play" with the system (leads to greater acceptance) and the system is able to measure the learning progress via the monitoring of this "playing" and the analysis of tests and tasks solved by the at the end of a lesson. To fulfil these demands the systems have to be built on other platforms than the multidisciplinary systems. Especially an HTML/JavaScript based platform does not offer enough possibilities to create modules that provide the desired behaviour. As a consequence, a lot of these systems are not web-based because another technology than HTML is chosen. Communication between learners and teachers or among learners is often not reflected by the systems. These systems are mostly designed as supporting tools which

[21] see [Vir02]
[22] see [Web01]
[23] see [RWT02]
[24] see [Ker99]

can be used in addition to the normal courses at the university where communication will happen.

To summarise it, one can say that most of these discipline specific systems are only usable in a supportive way to explain special aspects of a subject in addition to the classic way of learning. They seem to be more appreciated in the sense of providing learning supportive methods. An example will be shown below.

In^{2}Math This system can be used by students of Mathematics and Informatics to update their knowledge gained in undergraduated courses about mathematical coherences.[25] The system uses external tools like provers (pl2tme[26] or protein[27]) to verify solutions of tests or systems used to display mathematical content (MuPad[28]). Especially, the provers or automatic deduction test systems allow tests focused on mathematical subjects that go to a larger extent than the simple solving of multiple choice tests. The system is accessible via the web. However, there is a lack of communication possibilities like they are provided by the multidisciplinary systems. Especially the communication between learners and teachers or among learners, implemented as a chat or forum, for example, is not presented. But it has to be considered that this system is still under development.

Combined systems

As described above, a major drawback of the reviewed multidisciplinary systems is the missing programming functionality to provide subject related modules which support the user during the learning process. On the other hand, discipline specific systems provide such specific modules but lack communication mechanisms. A system that combines both approaches should consist of a powerful platform offering methods used for communication via the Internet and for structuring learning content. It should also provide programming interfaces to create specific subject related modules. A module which allows the user to create and experience simulation is an example of such a model. Today, platforms providing programming facilities are ei-

[25] see [Uni02d]
[26] see [Uni02c]
[27] see [AI 01]
[28] see [Sci02]

ther environments developed at universities (mostly Java based) or commercial content management systems. Hyperwave[29] is an example of a content management system which provides an e-learning environment. The actual e-learning environment is named Gentle-WBT[30]. Examples of systems developed at universities are shown in the following.

VAWI – Virtuelle Aus- und Weiterbildung Wirtschaftsinformatik (Virtual further education in Business Informatics) The system is intended for postgraduated studies. The learning environment is based on the "Clear Campus" system[31] created by the University of Bamberg.[32] In difference to other systems, it allows the creation of complex subjects related modules via the usage of the programming language Java as the platform. Different modules can be downloaded from the Clear Campus web side and plugged in the provided authoring tool. The modules created with the authoring tool can be embedded, in addition to the normal way of embedding in HTML (like in most of the other systems), in a Java applet or a Java application, which allows the development of complex Java modules which can be provide more feedback based on the actual knowledge level of the user when reacting to the learners actions. However, the objects provided by the system cover only the normal test and interaction methods that are also used in virtual universities.

Monist Simulation as an abstraction and modelling medium is used to present learning content.[33] This concept represents a constructivistic learning approach (see Chapter 2.1.2) that claims simulation to be the best method for (e-)learning. The system is especially useful in informatics and psychological subjects where content can easily be presented by using simulation. The system is planned to be web-based and shall also contain an authoring tool. It is still at an early stage.

[29] see [Hyp02]
[30] see [Aus02]
[31] see [Uni02a]
[32] see [Uni02b]
[33] see [mon02]

2.2 The programming language Java

Java was introduced by the software company *Sun* in November 1995. It was designed as a platform independent object oriented language. Thus, Java programmes will run with very small restrictions on every technological platform that is able to run the Java Virtual Machine. The implementation of applets and applications are two ways to create Java programmes. Applets are programmes that are accessible via the Internet (or Intranets, respectively) and displayed in a web browser.

Java provides a very powerful API[34] that enables the programmer to create GUIs: Swing API. Some important classes which will be referred to later are described below.

- `JFrame`: a window in the system
  Windows are used to display information. Examples of the use of a `JFrame` are the list with the simulation steps (see Chapter 3.1.1), pop-up information and error messages. Furthermore, the class is used to reflect the main window if the system runs as an application.

- `JPanel`: an object storing other components
  It works as a container storing information about the layout in addition to the GUI components. The panel itself is an object that needs to reside in another object. The frame object usually holds one or more panels.

Another major feature of the programming language Java is the possibility to look up and change attributes and methods of the different objects used in the programme at runtime. The reflection mechanism provided by the API of Java (`java.lang.reflect`) can be seen as a possibility to gain meta data about the running programme.

The reflection mechanism and the object orientation of Java make this programming language very useful for creating simulations. Two roles are involved in the development process: the system developer and the simulation analyst. The system developer creates a simulation engine and provides a simulation library . The components found in that library can be used by

[34] Application Programming Interface

the simulation analyst to create simulation models. Java's object oriented design allows the analyst to combine existing components and to derive new specific components which fit his demands. These new components can be included in the library and reused by other simulation analysts. The reflection mechanism simplifies runtime binding, because it allows the invocation of unknown methods on unknown objects.[35]

2.3 Simulation

Simulation is the imitation of a dynamic process in a model which is used to gain firm insights that can be transferred to the real world. Its use is necessary, if:[36]

- An analytical, mathematical optimisation model cannot be defined for complexity reasons.
- Analytical methods require oversimplified presumptions.
- The analysis of the real system is too complex, too expensive, too dangerous or not possible at all.

Three basic types of simulation can be distinguished:[37]

1. *Monte Carlo simulation*:
   Monte Carlo simulation is used to analyse static problems with known probabilities.
2. *Discrete event simulation*:
   Discrete event simulation is based on state variables that change only at discrete points of time (see Chapter 2.3.1).

[35] see [PC00]
[36] see [VG01] p. 176
[37] see [DD95] p. 207

3. *Continuous simulation*:
   Continuous simulation is used to model dynamic systems in which the state variables change continuously with respect to time. A typical example of a continuous model is a biological model: the competition between two populations.[38]

2.3.1 Discrete event simulation

In discrete event simulation models, the state variables are time dependent and change due to the occurrence of events at discrete points of time. Their behaviour is controlled by a next event technique. Thus, the time increment forwarding the simulation clock is variable. The model is examined and updated only when it is known that a change of the state may occur.[39] Therefore, an event list (or event calendar) is used which represents the core of a discrete event simulator. The event list is a queue sorted by the time of occurrence of the events. Each event is associated with the point of time when it will occur. The topmost event of the event queue is the next event that will be raised.

At the start of a simulation run, the event list is filled with start events. Then the topmost event is processed. Event processing may generate new events. If the processed event represents the entering of a part in a machine, for example, it will be calculated when the part is supposed to leave the component and the determined value will be added as a new event to the event list. Processed events are removed from the list. If an event cannot be processed (due to congestion in the model, for example), it will be removed from the list and added with a new time value. If the event is supposed to be processed at time t, for example, it will be scheduled for processing at $t + 1$. After that, the simulation clock is forwarded to the time of the next event in the list and the next event will be raised. If the event list is empty, the simulation run will stop.

[38] see [LK91] p. 109
[39] see [Pid92] p. 17

2.3.2 Stochastical values in simulation models

Statistical know-how is needed for input data collection, input data preparation, interpretation of simulation results and for the proper planning of simulation experiments.[40] The assessment of input data is very important, because with incorrect input data, simulation will lead to wrong results.[41]

In real systems, it is not always possible to predict the values of all variables. Typical examples of variables usually based on random numbers are interarrival times and process times. In these cases, the probability distribution is known or can be determined, which the values of the variables are based on.

Two results can be achieved with the use of probability distributions. First, it is possible to ensure that the values of the variables are chosen randomly. Second, the produced numbers are in the correct proportion.[42] For example, if a normal distribution is chosen, values in the middle of the range will be the most likely ones to occur.

Continuous distributions (like normal distribution, uniform distribution, etc.) can be parameterised in several ways. Therefore, location parameters, scale parameters and shape parameters are used.The position on the x-axis can be specified with a location parameter. It is usually the lower endpoint or the midpoint (for example the mean in case of the normal distribution). Scale parameters specify the scale of measurement of the values within the range. The third type, shape parameters, determine the basic form of the distribution.[43]

Some important probability distributions will be discussed in the following.

[40] see [KG95] p. 24
[41] see [VG01] p. 180
[42] see [Pid92] p. 193
[43] see [LK91] p. 329

Uniform distribution

Usually, the uniform distribution is used in cases with limited information.[44] It is only known that the values vary between two borders of a range (a and b). All values in the range of the uniform distribution occur with the same probability. Random number generators usually generate uniform distributed values between 0 and 1. These values can be used in transformation processes to generate random values from other distributions.[45]

The graph of the density function of the uniform distribution (see Equation 2.1) will be shown in Figure 2.1. The distribution function will be presented in Equation 2.2. The parameters of the uniform distribution are the lower bound a and the upper bound b. a is a location parameter and $b - a$ is a scale parameter. The range of the distribution is $[a, b]$, its mean $\frac{a+b}{2}$ and its variance $\frac{(b-a)^2}{12}$. The maximum likelihood estimator for parameter a is $\hat{a} = \min_{1 \leq i \leq n} X_i$, for b it is $\hat{b} = \max_{1 \leq i \leq n} X_i$

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$F(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ 1 & \text{if } b < x \end{cases} \tag{2.2}$$

[44] see [LK91] p. 330

[45] for further information see [LK91] p. 462

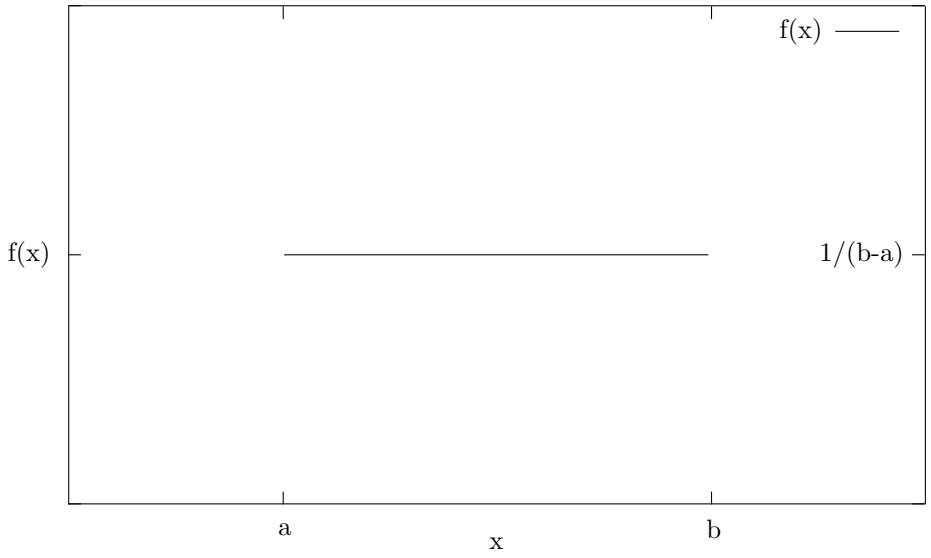

Figure 2.1: The density function of the uniform distribution

Exponential distribution

The exponential distribution can be used to model interarrival times of customers that occur at a constant rate. It has only one parameter: the scale parameter $\beta > 0$. The density function of the distribution will be shown in Equation 2.3, its distribution function in Equation 2.4. Its range is $[0, \infty)$, its mean β and its variance β^2. The maximum likelihood estimator for β is $\hat{\beta} = \bar{X}(n)$. In Figure 2.2, the graphs of the density functions of the exponential distribution for $\beta = 1$, $\beta = 2$ and $\beta = \frac{1}{2}$ will be shown.

$$f(x) = \begin{cases} \frac{1}{\beta} e^{-\frac{x}{\beta}} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$F(x) = \begin{cases} 1 - e^{-\frac{x}{\beta}} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

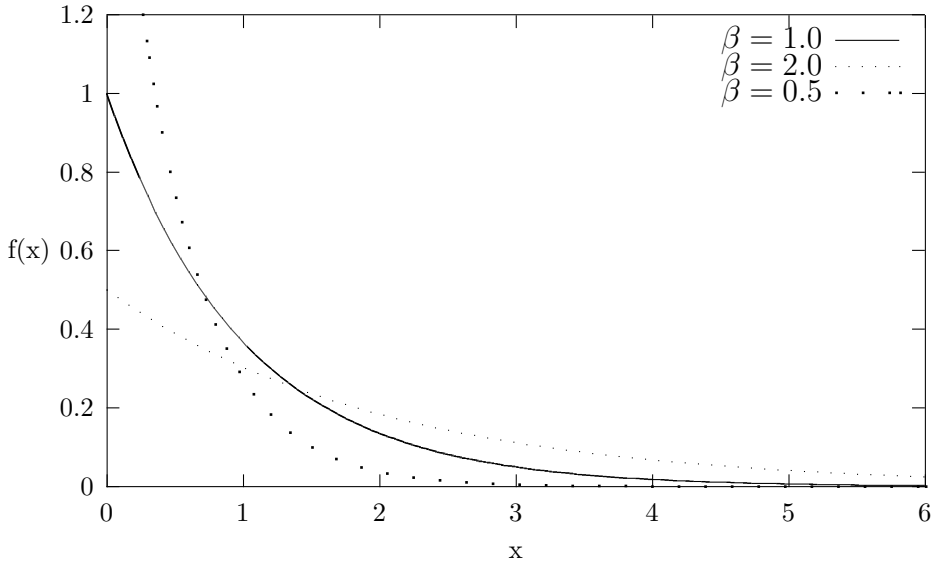

Figure 2.2: Density functions of exponential distributions with different scale parameters β

Normal distribution

The normal distribution can be used to model various types of errors. Due to the virtue of central limit theorems, it is possible to use normal distributed values for quantities that are the sum of other quantities. Its location can be specified by the parameter $\mu \in (-\infty, \infty)$. It represents the mean of the distribution. The variance is σ^2 which is the square of the scale parameter $\sigma > 0$. The parameters can be estimated using the maximum likelihood estimators $\hat{\mu} = \bar{X}(n)$ and $\hat{\sigma} = \sqrt{\frac{n-1}{n} S^2(n)}$. The range is not bounded $(-\infty, \infty)$. The distribution function does not have a closed form, the density function will be shown in Equation 2.5.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.5}$$

The normal distribution with $\mu = 0$ and $\sigma = 1$ is also known as the standard normal distribution.

The graph of its density function will be shown in Figure 2.3.

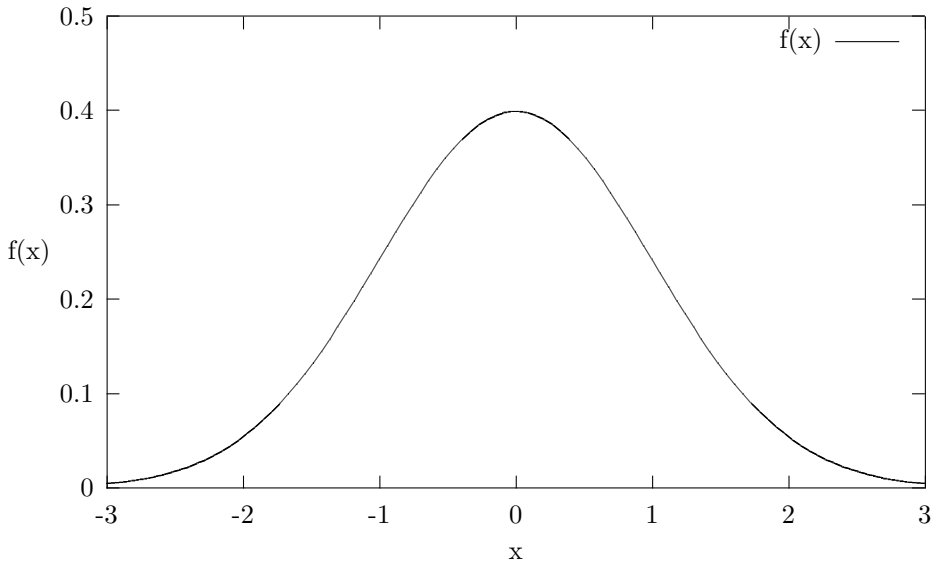

Figure 2.3: The density function of the standard normal distribution

Triangular distribution

If no data is present, a triangular distribution can be used. It is parameterised with a lower bound a (location parameter), an upper bound b ($b - a$ is a scale parameter) and an expected value c (shape parameter). Its range is $[a, b]$, its mean $\frac{a+b+c}{3}$ and its variance $\frac{a^2+b^2+c^2-ab-ac-bc}{18}$. Maximum likelihood estimators for the parameters do not make sense, because the triangular distribution will be used, if no data is present. The distribution function will be shown in Equation 2.6, the density function in Equation 2.7.The graph of the density function will be shown in Figure 2.4.

$$F(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & \text{if } a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & \text{if } c < x \leq b \\ 1 & \text{if } b < x \end{cases} \tag{2.6}$$

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & \text{if } a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{if } c < x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

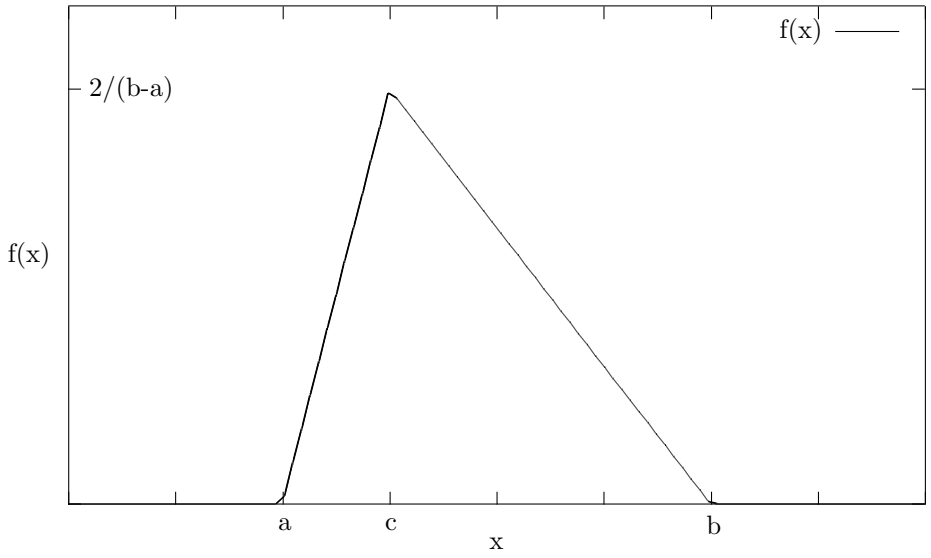

Figure 2.4: The density function of the triangular distribution

Poisson distribution

In contrast to the other explained distributions which are continuous distributions, the Poisson distribution is a discrete distribution. It is used to describe the number of items in a batch of random size or the demanded number of items from an inventory. Another application are numbers of events

which occur at a constant rate in an interval of time. Its parameter $\lambda > 0$ is equal to the mean and the variance. Its maximum likelihood estimator is $\hat{\lambda} = \bar{X}(n)$. The range of the distribution is $\{0, 1, \dots\}$. Its distribution function will be presented in Equation 2.8.

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ e^{-\lambda} \sum_{i=0}^{\lfloor x \rfloor} \frac{\lambda^i}{i!} & \text{if } 0 \leq x \end{cases} \tag{2.8}$$

Discrete probability distributions do not have density functions, they have mass functions instead. The mass function of the Poisson distribution will be shown in Equation 2.9, the graph of that function (with $\lambda = 2$) will be displayed in Figure 2.5.

$$p(x) = \begin{cases} \frac{e^{-\lambda}\lambda^x}{x!} & \text{if } x \in \{0, 1, \dots\} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

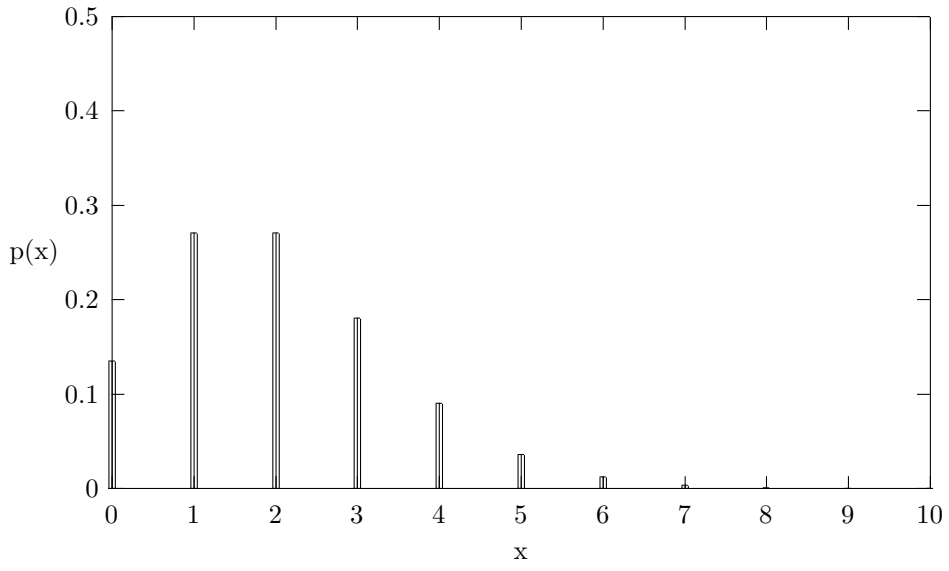

Figure 2.5: The mass function of the Poisson distribution with $\lambda = 2$

2.3.3 Simulation components

In component oriented simulation tools, the user can create models by choosing simulation components from a library. These components are not standardised and may vary from vendor to vendor. In the following, some simulation components will be presented. Some of them are implemented in SimTool (see Chapter 3.1).

The following model elements can be distinguished:[46]

1. *Dynamic elements*
   - *Products*:
     Products are created and/or processed.
   - *Workers*:
     Workers process or accompany objects.
   - *Movable transport objects*:
     Movable transport objects move objects.
2. *Stationary elements*
   - *Fixed transport objects*:
     Fixed transport objects move objects between stationary elements.
   - *Stations*:
     Stations process objects.
   - *Warehouses*:
     Warehouses store objects.
3. *Interface elements*
   - *Source*:
     The source allows objects to enter the system.
   - *Destination*:
     The destination removes objects from the system.

In this work, the term "simulation component" will refer to stationary elements and interface element. The term "movable element" will be used to refer to dynamic elements.

[46] see [KG95] p. 72

2.3.4 Routing in simulation networks

Every component (except the destination component) tries to forward a processed movable element to its successor component. If there is more than one successor, the question will arise how the component chooses the actual successor component from the set of possible successors. In the following, several strategies will be discussed. Of course, there are a lot of other algorithms to determine a successor.

Best alternative This algorithm will regard the number of movable elements on the links to the successor components as well as the state of the successor components. The alternative that is supposed to have the shortest delay will be chosen.

Round robin The successor components are iterated step by step. If the last time component i is chosen, the algorithm will choose $i+1$. If $i+1$ is larger than the number of components, the first component will be chosen. There exist two different versions of this algorithm. The strict round robin follows the above scheme exactly. If the chosen component is blocked, the movable element will not be forwarded. The other version of the round robin routing algorithm tries the next component $(i+2)$ upto a component that is able to process the movable element will have been found.

Weighted routing This algorithm allows to specify weights for each component. For example, if there are two possible successors and the weights $(2, 1)$ are specified, the ratio of the total number of movable elements routed to the successor components will be $2:1$.

Fixed order This algorithm can be used to define a priority hierarchy of the successor components. For example, it is possible to specify a hierarchy for three components: 2, 1, 3. The component will try to forward all movable elements to the successor 2 until the link to this successor blocks. Then all movable elements will be forwarded to successor 1 until the link to successor 2 is free again or the link to successor 1 is also blocked. If the links to successor 1 and 2 are blocked successor 3 will be used.

2.3.5 Web-based simulation environments in Java

The creation and use of web-based simulations are often discussed at the moment. Java as a programming language for the web arises because it offers an object-oriented, modular way to create programmes for the web. With these properties, it fits especially for the creation of simulation engines. However, not only the engines are created with Java. Easy to use GUIs which enable the user to create simulation modules without programming are also available. This chapter describes two of the approaches to establish simulation in the Internet.

JSIM

JSim[47] is a Java-based simulation environment. It can be used via the Internet. The aim of the system is to make: "simulation models as well as simulation results to be as readily distributable and composable as today's Web documents.". The authors also mention that because of the "larger scope of Web-based simulation, greater demands are placed on simulation environments.".

The system uses the possibilities of the web in different ways. The authors of this system have especially researched how simulation can benefit from the web. One approach leaded to the possibility to "assemble models into a model federation". Thus, modules (parts of the simulation) can reside on different web spaces and will be dynamically linked in case of the execution of the simulation model. Modules are built by using the Java Beans technology which is especially useful to send and retrieve event based information over the Internet: "Using component based software, in this case Java Beans, one model is able to inject entities into another model."[48] The use of this technology allows it to create the modules in a way that they can be plugged in during runtime without recompiling. The system is able to run event-scheduled or process-interactive models.

The system is internally divided into three different layers. The bottom layer enables the core simulation engine. The middle layer holds the simulation modules (components) and the third layer contains the simulation

[47] [Mil01]
[48] [JAMX00]

environment which allows the user to create models in a "click and point" technique. The execution and creation of simulation models is supported by agents which provide help, especially while building the model. The user can create simulation models by clicking on the particular control button that provides the module (component) that has to be added to the simulation area. The created model can be enhanced by Java code.

A picture of the simulation environment can be found at Figure 2.6.

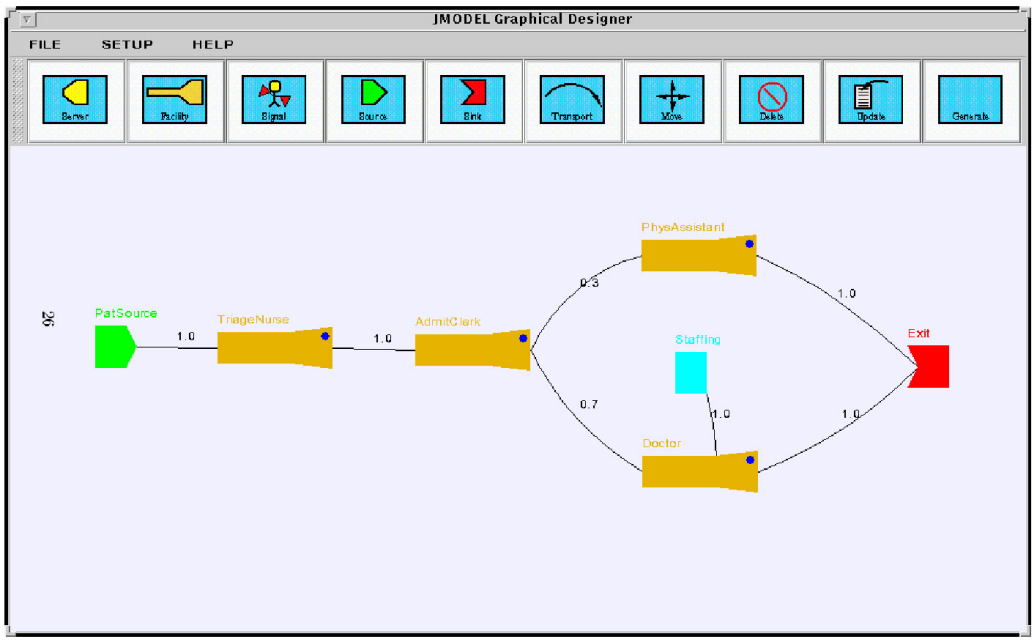

Figure 2.6: The JSim simulation environment

JSim can also be used to enhance the learning of simulation: "Besides JSIM's use as a research test-bed for web-based simulation, it can also be used to teach simulation (it has been used in the CS 421/621 simulation course at the University of Georgia)."[49] However, the system itself does not provide enough learning support to the user. Especially, interactive components are missing and a classification of the users is not possible.

[49] see [JAMX00]

simjava

This package allows the creation of process based discrete event simulation.[50] The understanding of the simulation will be supported by animations. The system is thread based. Thus, each entity (component) of the simulation runs in its own thread and communicates to the others via events. These events will be sent and received by special interfaces of the entities called ports. The system allows statistical evaluations by recording the values of the simulation during runtime. Recent approaches try to create packages that allow the simulation to be presented in 3d by using VRML[51]. Figure 2.7 will show an example of a simulation model created with the simjava system.

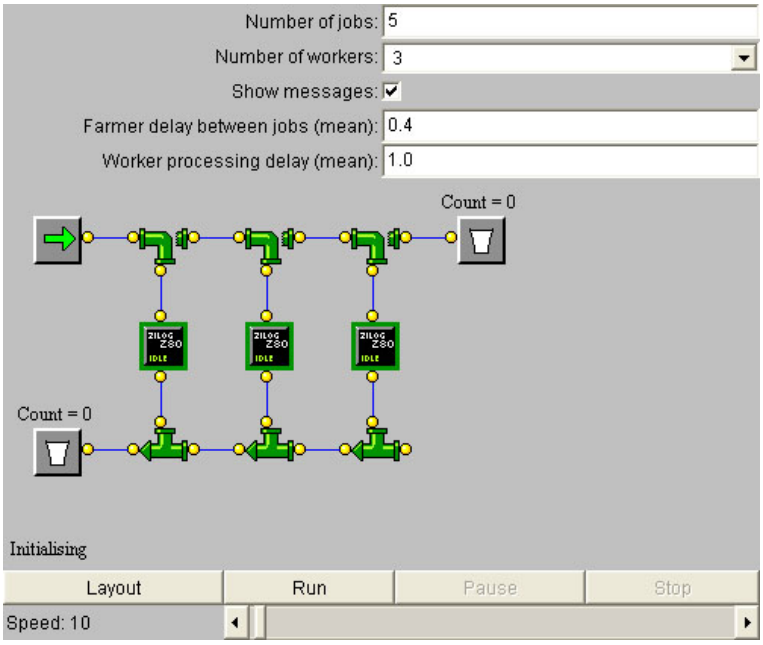

Figure 2.7: A sample simulation model created with simjava

[50] [Ins02]
[51] Virtual Reality Modelling Language

Chapter 3

Concepts

3.1 SimTool

This work is based on a programme called "SimTool 1.0"[1] being implemented in the winter semester 2001/2002 as part of a course in information management in which we participated. The aim of this project was to design and implement a prototype application which could be embedded into an interactive virtual learning environment. SimTool is a component oriented simulation tool, based on a discrete event oriented simulation engine (see chapter 2.3 for additional information). The main focus during the development of the prototype was set on modularity, extensibility and reusability. Furthermore, basic pedagogical support such as a good visualisation of internal processes and interactive components was designed. The user should be introduced to the subject simulation step by step. The content is represented in a manner that is easy to understand. Furthermore, the learner should be motivated by the system using interactive components.

Commercial simulation tools like eM-Plant[2] offer a huge variety of functions. Most of them are intended for professional users. Beginners suffer from a proliferation of menus and buttons. Therefore, the approach of SimTool is to reduce the number of accessible functions for beginners. Teachers can add functions by explicitly marking them as available in lessons for advanced

[1] see [DHH+02]
[2] see [Tec02]

users in configuration files; see Chapter 3.2 for a detailed description of the file format.

For SimTool 1.0 was designed as a prototype, another major goal was extensibility. Java developers should be able to create new simulation component classes or new implementations of mathematical probability distributions which can be loaded in SimTool at runtime without having to recompile the whole system. This possibility has been demonstrated by implementing an assembly station[3] and adding two new kinds of movable elements, pallet elements[4] and assembly elements[5].[6]

Other important design goals were the visualisation of internal processes of simulation models and their relation with mathematical issues and the possibility for the users to perform statistical evaluations. A learner should not only see movable elements jumping from simulation component to simulation component. Statistical information should also be made available. For example, the actual state of the components should be indicated by the use of different colours (blocked = red, processing an element = green, preparing = yellow and idle = blue). The time that a component spent in the different states during a simulation run should be visualised in a diagram not only after a simulation run has been performed but also during a simulation run. Figure 3.1 will show a screen-shot of the statistics window of SimTool.

During this work, SimTool has been extended. The result of this work, the new version, is called SimTool 2.0. See Appendix B for a comparison of the features of both versions.

3.1.1 SimTool architecture

SimTool 1.0 is implemented as a Java application to make it platform independent. Figure 3.2 will show an overview about the packages and classes of SimTool 1.0 in comparison to SimTool 2.0 which will be explained in this section. The darkened classes are implemented in SimTool 1.0. However, most of these classes are enhanced to meet the requirements of this work.

[3] `simulation.ComAssemblyStation`
[4] `simulation.MovablePalletElement`
[5] `simulation.MovableAssemblyElement`
[6] see [BB02]

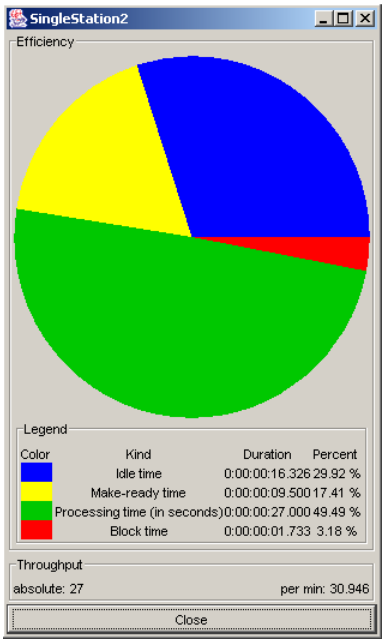

Figure 3.1: Statistic window for a component

Package `gui`

`Simtool` is the main class of the system. In SimTool 1.0, it was not part of any package, in version 2.0 it belongs to package `gui`. This change in the package structure was necessary to solve problems which occurred with some Java compilers. Jikes[7], for example, is not able to compile classes that do not belong to any package.

The package `gui` contains the GUI of SimTool. The main application window is implemented in class `Simulationswin` in version 1.0, which is renamed to `SimToolWin` in 2.0. The visible parts of the main window are the tool tree, the work sheet (`MainPanel`) and the help panel (`HtmlPanel`). The screenshot in Figure 3.3 shows the main window of the application after startup.

[7] see [IBM02]

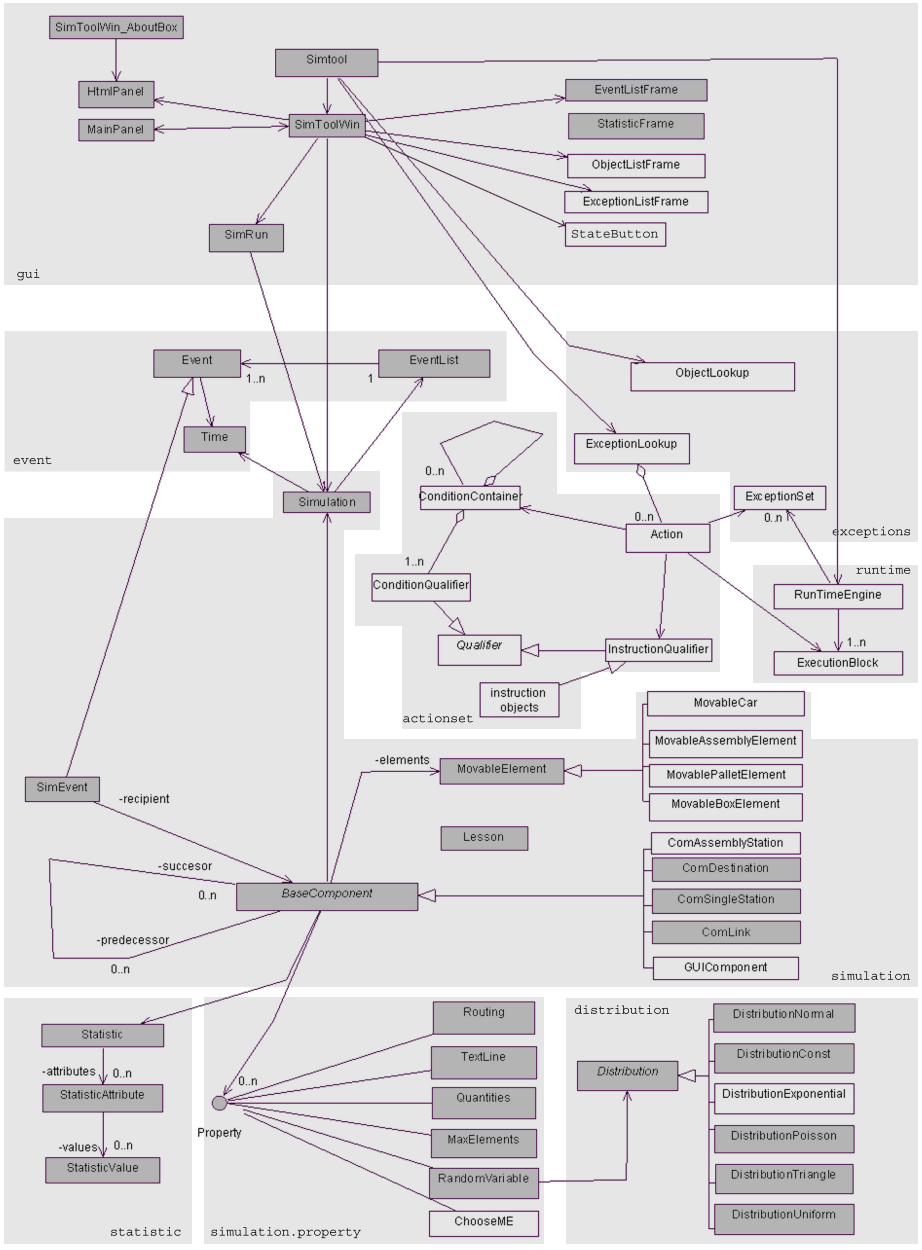

Figure 3.2: Selected classes and packages of SimTool

The class `EventListFrame` provides a `JFrame`[8] which is used to display the event list (see Figure 3.4). The `JFrame` which is provided by the class `StatisticFrame` is used to display statistical information about simulation components. A screen-shot of a window with statistical information can be found in Figure 3.1 on page 32. The two other frames (`ObjectListFrame` and `ExceptionListFrame`) display debug information to support lesson authors. The `ObjectListFrame` displays the objects that are registered on the system with all its script interface methods (script interface methods are described in Chapter 3.4). The `ExceptionListFrame` displays information about the current loaded exception sets (exception sets are described in Chapter 3.5).

The link between this package and the simulation core package (package `simulation`) is the class `SimRun`. This class is a thread class[9] which is used to run the simulation independently of the user interface, because the execution of the simulation must be delayed in order to be able to visualise the run. If no delay were set, the learner would not see neither movable elements shifting from component to component nor state information about the components. The delay between the execution of two simulation steps is based on the actual settings of the speed slider in the main window. Note that if the speed slider is set to a too high value strange effects such as elements moving reversely might occur.

Package `event`

This package provides classes for event handling. Events can happen during simulation runs and are represented by the class `Event`. At the moment, the only type of event derived from this base class is a simulation event implemented in the class `SimEvent`[10] (see UML class diagram in Figure 3.5). Simulation events are generated by components (see description of package `simulation`). The following types of simulation events exist:

- *Leave-event*:
  A movable element tries to leave a component.
- *Make-ready-event*:
  A component has finished its preparing process.

[8] `javax.swing.JFrame`
[9] derived from `java.lang.Thread`
[10] `simulation.SimEvent`

- *Create-event*:
  A source generates a movable element.

- *Destroy-event*:
  A destination removes a movable element from the system.

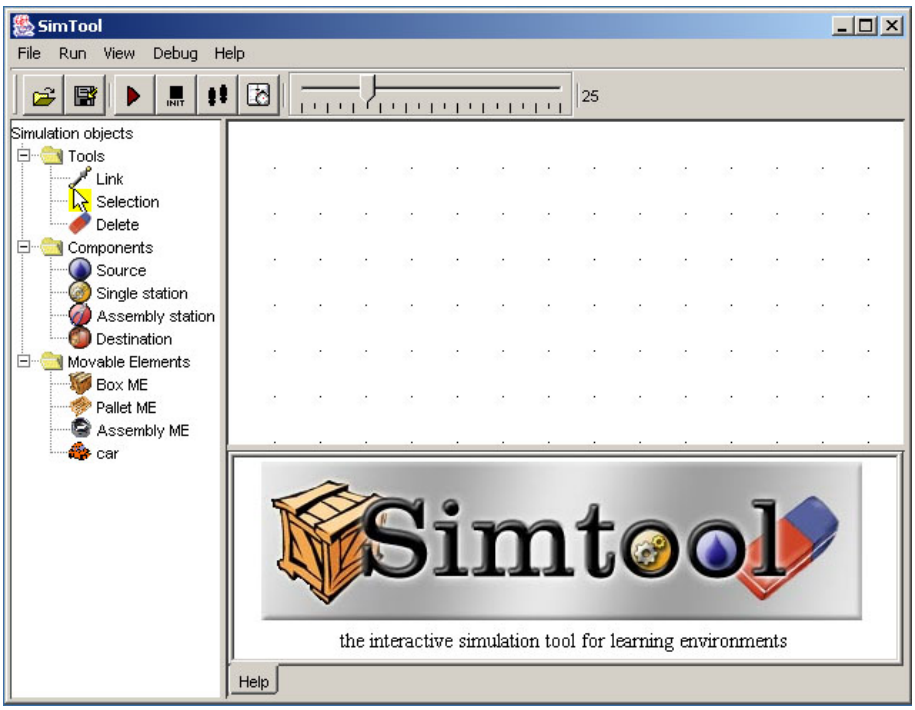

Figure 3.3: The main window of the application after startup

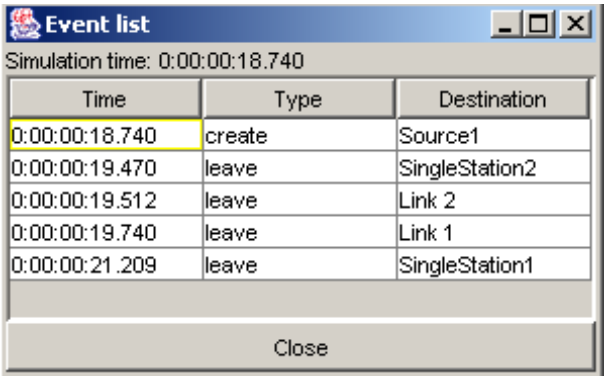

Figure 3.4: The event list

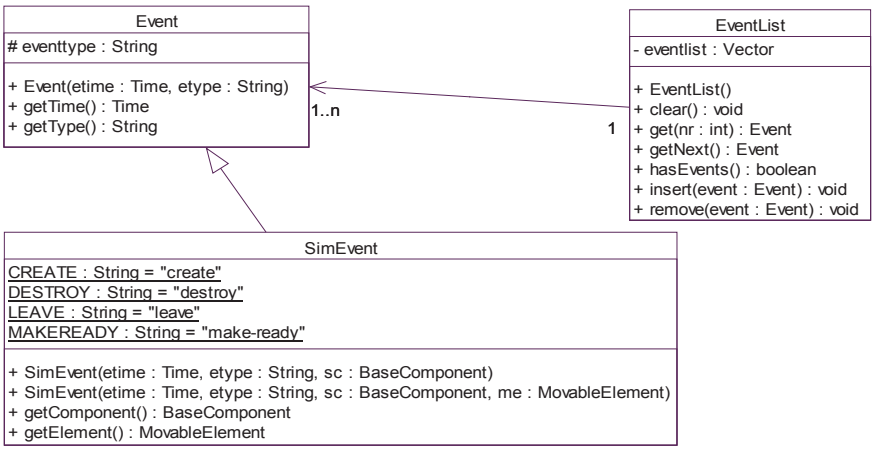

Figure 3.5: Selected methods and attributes of Event classes

Events are managed by the event list (class `EventList`),which stores the events, the point in time when they will occur and the involved simulation component (for example, a source component for a create-event). Figure 3.4 shows the contents of an event list displayed in an `EventListFrame`. The list is always sorted by the target simulation time of the events. The topmost event is the next event that must be raised. In a discrete event simulation, the simulation clock is always forwarded to the time when the next event has to be raised. Then, the involved component must handle the event. This can lead to new entries in the event list. If a leave event occurs, for example, the system will try to move the movable element from the component where it actually resides to a successor of the component. That successor component will calculate the point in time when it will have finished processing the movable element and enter a leave-event in the event list.

Package `simulation`

The package is the core simulation package and contains all classes that belong to the simulation engine. The class `Simulation` is the central object of this package, because it holds the event list (class `EventList` in package `event`) and a list of all simulation components which exist in the simulation model. A simulation component is represented by the abstract class `BaseComponent` (see Figure 3.6). Movable elements that are processed by simulation components are stored in a vector. Whether the movable ele-

ments may have different types depends on the component, as well as the maximum number of movable elements which can be in a component at one time. The following component classes are derived from the base class:

- `ComSource`: a source component
  Creates movable elements.
- `ComDestination`: a destination component
  Movable elements exit the system through this component.
- `ComSingleStation`: a single station
  Station (a machine, for example), which can process only one movable element at a certain time.
- `ComLink`: a link between two other components
  Links a predecessor component with its successor.
- `ComAssemblyStation`[11]: an assembly station
  Assembles different movable elements to one assembly element.
- `GUIComponent`[12]: graphical component
  Used to display components in the tree.

For more information about simulation components see Chapter 2.3.3.

In general, components can have any number of predecessor and successor components. However, some components are different. The link, for example, must have exactly one predecessor and one successor. The source component does not accept predecessors, the destination component cannot have successors.

As described above, components handle events. At the moment, the only type of events that exists is a simulation event which is represented by the class `SimEvent` that is derived from `Event` in package `event`. Other possible events would be events of the GUI or user defined events. Simulation events are events that are generated by simulation components.

[11] not available in SimTool 1.0
[12] not available in SimTool 1.0

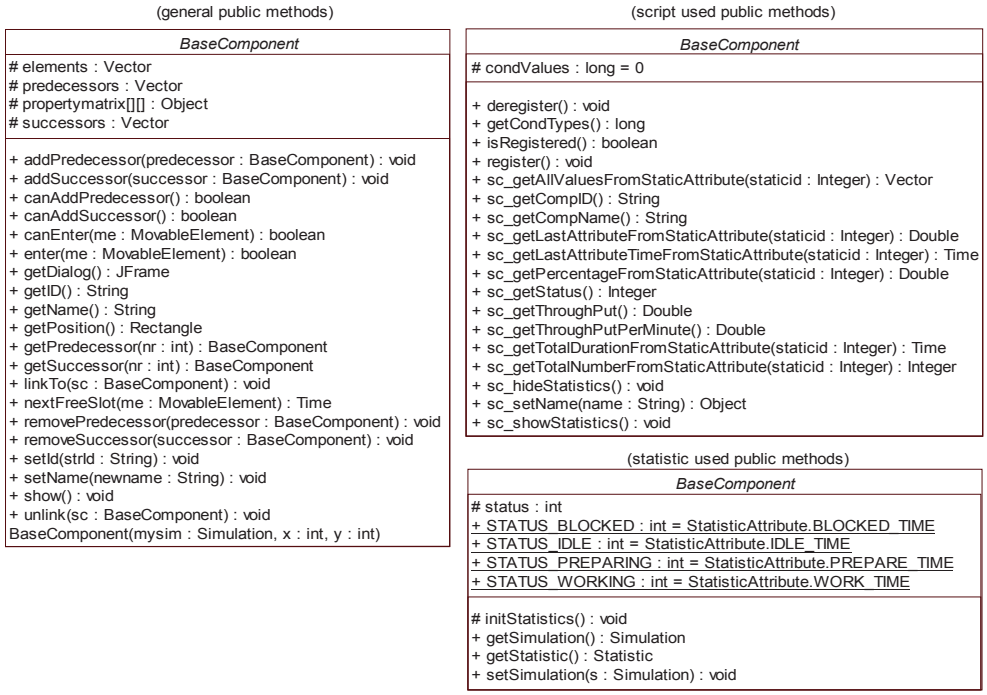

Figure 3.6: Selected methods and attributes of the base class of all simulation components

Each component records statistical information about itself during a simulation run. Statistical data are managed by an object of the type `Statistic` in package `statistic`.

Components usually have properties. The name is an example of a property that all components have. Other examples are the process time of a single station or the maximum number of movable elements that can be on a link at one time. The values of these properties can be changed by the user in the property dialogue of the component (see Figure 3.7). The interface `Property` in package `simulation.property` represents component properties.

The class `Lesson` is the interface to the configuration file which will be called "lesson file". This file stores information about the available components, activated GUI elements, etc. (see Chapter 3.2). Whenever a class needs information from the lesson file, it should use one of the static methods defined in this class (see below). In the case of a change in the format of this configuration file, it will not be necessary to edit all classes that access

the lesson file. Only `Lesson` will have to be modified. The following static methods exist in class `Lesson`:

1. `boolean isEnabled(...)`
   This method is used to check whether a specific feature is activated in the lesson file. The keyword of the feature is the first parameter, the name of the feature is the second parameter. Keywords and names of available features are explained in detail in Chapter 3.2.1. The following example shows an excerpt of a lesson file.

   ```
   # activated components
   components = Source,SingleStation,Destination
   ```

   The source component, as well as the single station and the destination are activated, such that a call to `isEnabled("components","Source")` would return `true`. Here, the assembly station is not activated, so a call to `isEnabled("components","AssemblyStation")` would return `false`.

2. `String[] getFeatures(...)`
   This method can be used to read all activated features. The result is an array of strings containing the names of the activated features. The parameter of the method is the keyword indicating which features shall be read. In the case of the example mentioned above, a call to `getFeatures("components")` would return the array {`"Source","SingleStation","Destination"`}.

3. `String getValue(...)`
   This method is intended for keywords which only accept one parameter ("language" for example).

Package `statistic`

Every component that records statistical information needs to have an object of the type `Statistic` which manages the recording and display of statistical data. For example, the `Statistic` object can show a window with all statistical information of a component. A screen-shot of a statistic window is presented in Figure 3.1 on page 32. Statistical information are recorded and represented by instances of the class `StatisticAttribute`. For example, the

different states of a component exist as statistical attributes in the base class of all components (`simulation.BaseComponent`).

- `STATUS_WORKING`:
  The component is processing a movable element.
- `STATUS_IDLE`:
  The component is idle.
- `STATUS_PREPARING`:
  The component is preparing itself to be able to process movable elements. This might be necessary between the processing of two movable elements.
- `STATUS_BLOCKED`:
  The component is blocked. The maximum number of movable elements in the component has been reached and the component is not able to forward a movable element to a successor component. Blocked components usually are displayed in red.

A statistical attribute stores a list with the recorded data which can be either a time-value-pair represented by class `StatisticValue` or an aggregated value, for example a sum. Sums are used in the statistical attributes for the states of the component where the total amount of time that the component has spent at this state is stored. Every state has its own statistical attribute (`stat_work`, for example). Whenever a movable element has been accessed, the process time of that element is added to the sum stored in `stat_work`. Besides the statistical attributes for the states, the throughput and the throughput of a component per minute can be obtained.

Package `simulation.property`

Component properties are represented by the interface `Property` (see UML diagram in Figure 3.8). A `Property` object does not only store the values of the component properties, it also provides a `JPanel`[13] which offers a user interface to change the value. In case of the name property, this user interface consists of a label ("Name:") and a text field which allows to enter a single line of text (class `TextLine`). Figure 3.7 will show a screen-shot of a property dialogue of a single station. The following properties are available:

- `ChooseME`[14]: a property to choose a class of movable elements
  This property is used by the source component to specify the kind of movable elements it shall create.
- `MaxElements`: a property to specify an integer value
  This property is used to specify the maximum number of movable elements which can be on a link at the same time.
- `Quantities`[15]: a property to specify an amount of movable elements
  The property is used in the assembly station to specify what parts are needed to assemble an element.
- `RandomVariable`: a property to choose a distribution
  This property is used to specify the process time or the make-ready time of a single station. Figure 3.7 shows a tab with two `RandomVariable` properties. For the process time, a uniform distribution between 0.5 and 3.0 is chosen. The property of the make-ready time is set to a constant value of 0.5.
- `Routing`: a property to specify the routing algorithm
  This property is used used to determine the successor component.
- `TextLine`: a property to specify a line of text
  This property is used in all components to specify the name.

[13] `javax.swing.JPanel`
[14] not available in SimTool 1.0
[15] not available in SimTool 1.0

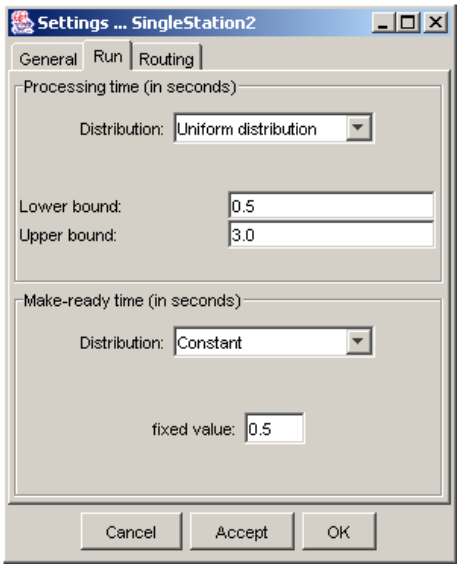

Figure 3.7: A property dialogue of a single station

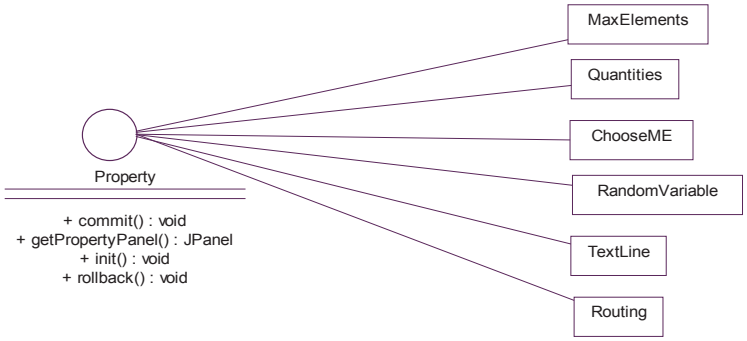

Figure 3.8: The property package

Package `distribution`

This package contains implementations of various mathematical probability distributions (see Chapter 2.3.2 for more information on mathematical distributions). All implementations of distributions are derived from the abstract base class `Distribution` (see UML diagram in Figure 3.9).

The following distributions exist in this package:

- `DistributionNormal`: a normal distribution
- `DistributionUniform`[16]: a uniform distribution
- `DistributionTriangle`: a triangular distribution
- `DistributionPoisson`: a Poisson distribution
- `DistributionConst`: a constant value
- `DistributionExponential`[17]: an exponential distribution

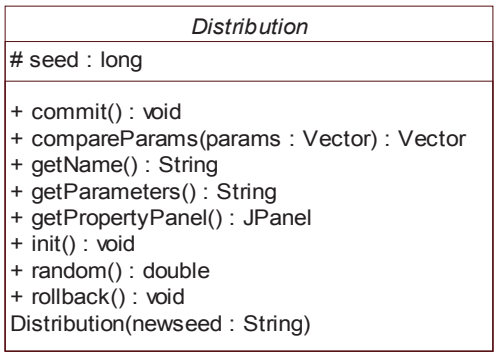

Figure 3.9: Selected methods and attributes of the base class of all distributions

package `runtime`

In SimTool 2.0, the package `runtime` has been added. It contains a class that reflects a runtime engine for scripts (`RunTimeEngine`) and parts of scripts (`ExecutionBlock`). The concepts of scripts and the runtime engine will be explained in Chapter 3.4 in detail.

[16] in SimTool 1.0 named `DistributionEqual`
[17] not available in SimTool 1.0

package `actionset`

Package `actionset` which has been added in SimTool 2.0 provides classes that represent instructions (class `InstructionQualifier` and subclasses) which can be performed by scripts. Furthermore, this package contains classes that are necessary for the exception handling (`ConditionQualifier`, `ConditionContainer` and `Action`). For a detailed description of the concepts of scripts and the exception handling see Chapters 3.4 and 3.5.

package `exceptions`

In SimTool 2.0, the package `exceptions` has been added. It contains a class that reflects a set of exceptions (class `ExceptionSet`). Furthermore, it contains two look-up objects (`ObjectLookup` and `ExceptionLookup`) and several helper methods. The concepts of the exception handling will be described in Chapter 3.5.

3.1.2 Evaluation of SimTool 1.0

The aim of the previous work was the creation of a simulation tool which accompany lectures. This tool should animate students to "play" with simulation models by presenting the material in a vivid way.[18] This goal was partly achieved, especially, due to the GUI which is informative and easy to use. However, there are certain drawbacks.

Application vs. Applet

The fact that SimTool 1.0 is a Java application reduces the number of use cases, because a standalone Java application cannot be embedded into a web-based virtual learning environment. For using SimTool in such an environment, it should be an applet so that it can be run inside a web browser as well as loaded from a http-server via a URL.

[18] see [DHH+02] p. 3

Simulation components

The number of existing components is quite limited. In SimTool 1.0, there are only four different components:

1. Source
2. Single station
3. Link
4. Destination

The user needs more components for being able to create more realistic models. For example, a parallel station would be helpful to simplify models. At the moment, they have to be modelled using two (or more) single stations. Another example is an assembly station that is able to combine different movable elements.[19] A counterpart to the assembly station, a disassembly station where some movable elements are disassembled into its components would be useful as well.

The conceptual model of SimTool makes it possible to develop new components without having to alter the existing source code. The same applies to distributions as well and, with some restrictions, to the routing algorithms. The routing algorithms are all implemented in the same class.[20] Each routing algorithm should be encapsulated in its own class so it would be possible to add new routing algorithms at runtime, without having to recompile the programme. To create new components, the developer has to follow specific guidelines (see Chapter 3.3.1).

[19] implemented in SimTool 2.0

[20] `simulation.property.Routing`

Demonstration mode

If a teacher wants to present a model to students, the teacher has to build and save the model. This model can be opened with SimTool but explanatory elements are missing. The teacher has to explain the core concepts either orally or, in a virtual learning environment, by presenting texts. The teacher does not have other possibilities to emphasise the ideas which underly the model. A demonstration mode would be helpful in which the students can see how a model is constructed.

In addition to the demonstration mode, it would be helpful to be able to record the presentation that a teacher performs in class. The learners could watch the recorded presentation as often as desired, at home.

Interaction

Learners usually create simulation models according to a given task. If they think they have solved the task, they show their solutions to their teachers or they submit them via e-mail. The teacher has to examine each model by opening the property dialogues of all components for comparing the values with the values in the model solution. This task is quite time consuming. In an interactive learning environment, model validation should be performed automatically by the system.

Knowledge levels and database

In SimTool 1.0, all users are treated the same by the system (if they use the same lesson file). This could be changed if the system knew who was using it. Therefore, a database connection would be appropriate. In addition to user profiles, information about the student's learning progress can be stored in that database. In combination with model analysis it would be possible to mark the created simulation models and store the results in the database. An evaluation mechanism could generate a ranking, motivating the students. Additionally, the experience level of a learner can be measured and stored in the database, so that the system can adapt the user interface according to the learner's knowledge level.

Configuration and modularity

The lesson file concept allows the teacher to disable distributions, components and routing algorithms. Furthermore, the language and the country can be specified to use a localised version of SimTool. To custom-tailor SimTool for a specific lesson, it would be better to have more possibilities. Both, more restrictions and a method to set reasonable default values are needed in order to enable teachers to custom-tailor SimTool to better fit their ideas of lessons. In lessons intended for beginners, for example, it is better to hide a lot of features in the property dialogues of components.

3.1.3 Converting the application into an applet

SimTool shall be used in a web-based interactive virtual learning environment. Here, we assume a given virtual learning environment with the properties of running in a web browser, having a mechanism to classify learners in respect of their knowledge level and the possibility to integrate the simulation tool into the existing learning material.

For being able to run SimTool in a web browser, it is necessary to convert the Java application into a Java applet. For security reasons, applets have certain restrictions depending on the web browser.[21] Sun recommends that the following operations should not be permitted:

- Reading and writing in the local file system
- Opening network connections to remote hosts (except the host from which the code has been loaded)
- Accessing system properties
- Waiting for connections on privileged ports (ports < 1024)
- Accepting connections from privileged ports
- Using multicast-sockets
- Leaving the Java interpreter (`System.exit()` or `Runtime.exit()`)

[21] see [Fla98] p. 148

- Creating new processes using `Runtime.exec()`
- Dynamic loading of native libraries
- Initiating print jobs
- Accessing the clipboard
- Accessing the event-queue of the system
- Creating threads outside the thread group of the applet
- Using reflection methods to retrieve non-public elements of a class
- Manipulating security identities or security properties

Especially, due to the first three items, some changes to the original concept of SimTool have to be made. They are described in this section.

Accessing files

When converting a Java application to an applet, problems occur if the application uses files. As described above, applets normally are not permitted to operate on the local file system. SimTool 1.0 uses files at the following spots:

1. Stored models
2. Lessons
3. Documents in the help panel

The third topic is not a problem, as long as the help documents to display are part of the jar file where SimTool resides in. Help documents describing the features of the system and the simulation components should be located in the following path:

```
help/[lang]/[subject]/[name of the specific topic]
```

The system provides localisation support. The different localised versions of the help documents have to be stored in different language directories. For a single station in English, for example, the relative path would be the following:

```
help/en/component/simulation.ComSingleStation
```

Help files for a movable box element in a German environment must be stored under the following path:

```
help/de/me/simulation.MovableBoxElement
```

If other help topics are added, they should be put into the `help` directory following the above scheme. The HTML file describing the components has to be named "index.html". An alternative way, if the bundling of the help files is not desired, will be the access via URLs.

In the other two cases, the easiest way to access files is following the same method described above. A subdirectory of SimTool could be created to store the files. However, more flexibility is needed. As a consequence, all files used by SimTool besides bundled help files are referred to by URLs. However, in some cases, permissions to access these files must be given to the Java Virtual Machine. Figure 3.10 will show which files can be accessed by the applet. The security manager will determine if permissions are required. They are usually required for files located on the local file system and on remote servers (other than the server hosting the applet). The necessary permissions are described in the next section in detail.

The new concepts that have been added to SimTool, scripts and exception sets (see Chapters 3.4 and 3.5), use files which are referred to by URLs. Thus the correct permissions will have to be granted to access these files if they are hosted by a different server than the applet.

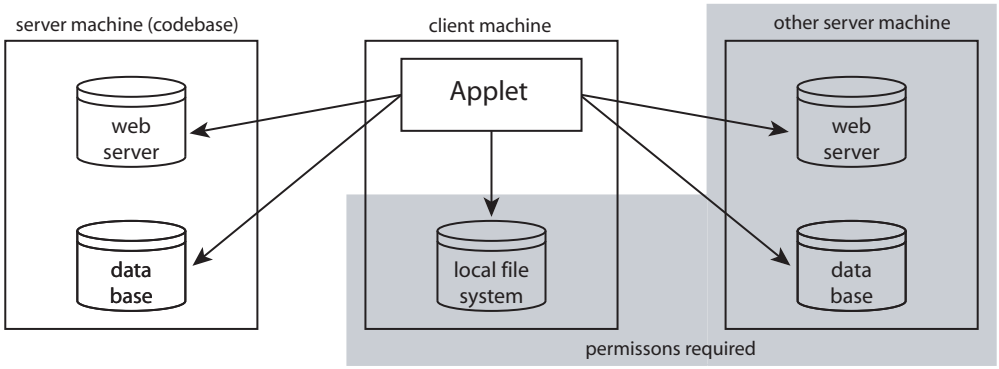

Figure 3.10: Accessing resources from applets

Required permissions

The Java2 platform security model allows fine-grained access control.[22] Access permissions are based on the codebase of an applet or an application. A group of permissions for a codebase is called *policy*. These policies are specified in policy files which exist in two versions:[23]

1. *The system wide policy file*:
   - *java.home*[24]`/lib/security/java.policy` (on Unix)
   - *java.home*\`lib\security\java.policy` (on Windows)
2. *The user policy file*:
   - *user.home*[25] `/.java.policy` (on Unix)
   - *user.home*\`.java.policy` (on Windows)

These files can be altered in a text editor or with the programme called "policytool" which is part of Sun's JRE and JDK. A sample file for a specific installation scenario is shown in Appendix A.2.

[22] see [Gon02]
[23] see [Sun02]
[24] The system property that is used by Java to refer to the installation directory of the Java runtime environment.
[25] The system property that is used by Java to refer to the user's home directory.

SimTool is a localised programme, thus it is possible to change the language of the user interface. To be able to use this feature, it must be allowed to change the language settings of the Java Virtual Machine. To grant the permission, the policy file must contain the following entry for the codebase of the applet:

```
permission java.util.PropertyPermission "user.language",
  "write";
```

If files in the local file system are accessed, permissions must be granted. Lesson files, scripts and exception sets are examples of files used by SimTool. To load the file "c:\simtool\lessons\lesson1" from the local file system (on Windows), the following entry for the codebase of the applet must be specified in the policy file:

```
permission java.io.FilePermission
  "c:\\simtool\\lessons\\lesson1" , "read";
```

If the menu point "file/save" is enabled, various permissions must be granted. Read and write access to the user's home directory and all files must be given. The file choose dialogue needs to synchronise its contents with the file system. Therefore, it is necessary to be able to modify threads. Additionally, it must be possible to change the current directory. The following entries must exist for the actual codebase (for a user with a home directory named "/home/simulation"):

```
permission java.io.FilePermission "/home/simulation/-",
  "read,write";
permission java.lang.RuntimePermission "modifyThread";
permission java.util.PropertyPermission "user.dir","write";
```

Web files which reside on the same server as the applet can be accessed without special permissions.[26] If the files are located at a different machine, network permissions must be granted.

[26]Other JVMs than Sun's may deal with that in a different way

For example, if they reside on the computer named "myserver.mydomain.net" with the ip-address 192.168.0.6 (not hosting the applet), the following entry in the policy file is required:

```
permission java.net.SocketPermission "myserver.mydomain.net",
  "resolve";
permission java.net.SocketPermission "192.168.0.6",
  "resolve,connect"
```

File format changes

The load/save mechanism has changed completely. In SimTool 1.0, the standard Java object serialisation is used.[27] This mechanism allows a quick serialisation but it produces compressed binary output which cannot be read by human beings.[28] Furthermore, there exist incompatibilities between certain JDK releases.

Using XML to store the simulation models is a better way since it is human readable and, with JAXP[29], there exists a very flexible API to process XML documents.[30] Since JDK 1.4, `JAXP` is a part of the standard JDK. More information about XML can be found in the Section "XML as a format for a script language" on page 72. Models are stored in the format that applies for runtime scripts (see Chapter 3.4). These models can be loaded when SimTool is started by specifying their URL as the applet parameter `Script`. The applet parameters are described in the next section.

XML is used in all files except the lesson files. The current lesson file concept uses Java property files (see Chapter 3.2). These files provide all needed features and are human readable. Therefore, it is not necessary to transition the lesson file format to XML.

[27] Object serialisation is a mechanism provided in Sun's JDK which can be used to write Java objects to files.

[28] see [Fla98] p. 182

[29] Java API for XML processing

[30] see [MDB01] or [Arm02]

Applet parameters and application parameters

SimTool 2.0 can be executed as a stand-alone application. In this mode, two command-line parameters are accepted. The first (`-l`) is the lesson file specified as an URL. If it is not specified, the file named "defaultLesson" (in the main package) will be used. The second parameter (`-s`) is the URL of a script which shall be executed when opening the SimTool. The default is not to open a script and to start with an empty simulation workspace (no simulation model and no exceptions loaded).

In contrast, the applet accepts more parameters. There are four boolean parameters used to specify the desired GUI elements. If the lesson allows it, it is possible to change the visibility of these four GUI elements at runtime via the menu bar. With the other parameters, the simulation environment is configured by loading a lesson file, executing a script and loading one or more exception sets. All filenames must be proper URLs. An URL consists of a protocol specification, for example, "http" or "file", followed by a colon and two slashes. If the http-protocol is used, the next part is the name of the host. The last part is the full path of the file, with slashes as separator characters between directories. Examples of valid URLs are:

- "http://www.simtool.der/scripts/script2.xml",
- "file:///S:/scripts/script4.xml" (on Windows)
- "file:///home/simulation/scripts/script6.xml" (on Unix)

A parameter can be omitted if its default setting is desired. The default values of the parameters are shown in Table 3.1. The following excerpt of a HTML file shows how applet parameters are specified. The complete HTML file can be found in Appendix A.3.

```
<applet codebase = "."
        code     = "gui/Simtool.class"
        name     = "SimToolWeb"
        width    = "800"
        height   = "600"
        hspace   = "0"
        vspace   = "0"
```

```
        align    = "top">

  <param name="Menu" value="true">
  <param name="HelpWindow" value="true">
  <param name="TreeArea" value="true">
  <param name="Lesson"
    value="http://www.simtool.de/simtool/defaultLesson">
  <param name="Script"
    value="http://www.simtool.de/simtool/run1.xml">
  <param name="Exceptions"
    value="http://www.simtool.de/simtool/basicSet1.xml;
     http://www.simtool.de/simtool/basicSet2.xml">
  <p>Sorry, but your browser does not support Java Applets</p>
</applet>
```

name	description	default value
`Menu`	Enable the menu bar	"true"
`Toolbar`	Enable the tool bar	"true"
`TreeArea`	Enable the tool tree	"true"
`HelpWindow`	Enable the help panel used for explanations of the tools	"false"
`Lesson`	Load the specified lesson file	"defaultLesson"
`Script`	Run the specified script	none
`Exceptions`	Load the specified exception sets	none

Table 3.1: The applet parameters and their default values

3.2 Configuration

3.2.1 Basic syntax of lesson files

The syntax of the lesson file is kept quite simple.[31] It is limited to keywords which any number of values can be assigned to. The values must be separated by commas, additional blanks are not allowed. Lines starting with the character '`#`' are considered to be comments and ignored. In SimTool version 1.0, the following configuration options are available in the lesson file:

[31] see [DHH+02] p. 26

- `distributions`: activated distribution functions
- `routing`: enabled routing algorithms
- `components`: activated simulation components
- `language`: use this language for the GUI
- `country`: localise according to the specified country

All these options are global options, thus features enabled (for example, distributions) can be used by all simulation components.

Global options

Distributions Names of the distributions that shall be enabled in the lesson can be assigned to the keyword `distributions`. The implementations of the distributions must reside in the package `distribution` and their classes must have the prefix (`Distribution`). The exponential distribution is implemented in the class `DistributionExponential`[32], for example. For more information about distributions see Chapter 2.3.2. In the original version, the names of the distributions were partly in German, partly in English. All names have been changed to English. The following distributions are available:

- `Normal`: normal distribution
- `Uniform`: uniform distribution
- `Triangle`: triangular distribution
- `Poisson`: Poisson distribution
- `Const`: constant value
- `Exponential`: exponential distribution

```
# globally available distributions
# (enable uniform, exponential and constant value distribution)
distributions = Uniform,Exponential,Const
```

[32] `distribution.DistributionExponential`

Simulation components Simulation components which can be chosen by the learner are specified using the keyword `components`. A similar naming rule as for distributions applies: the class names of the implementations must start with the prefix `Com` and must reside in the package `simulation`. See Chapter 2.3.3 for more information about simulation components. The following options exist:

- `Source`: source component
- `SingleStation`: single station
- `Destination`: destination component
- `AssemblyStation`: assembly station[33]

```
# globally available components
# (source, single station and destination)
components = Source,SingleStation,Destination
```

Routing algorithms Using the keyword `routing` it is possible to specify the routing algorithms that can be used to determine the successor component which a movable element is forwarded to. Routing algorithms are all implemented in the class `simulation.property.Routing`. The different routing algorithms are described in Chapter 2.3.4. At the moment, the following algorithms are implemented:

- `Fixed`: fixed order
- `Best`: best choice
- `RoundRobin`: round robin
- `WeightedRouting`: weighted choice

```
# globally enabled routing algorithms
# (best choice and round robin)
routing = Best,RoundRobin
```

[33]contributed by Sandra Bangewitz and Manuela Bernhard

Country and Language If the language for the labels and dialogues of SimTool is specified, the keywords `language` and `country` must be used. As opposed to the other settings, only one value can be assigned to these keywords. If no language-country-combination is specified, the default settings of the Java Virtual Machine will be used. If a non supported language is set, English will be used. At the moment, the following combinations are possible:

1. `language = de`
   `country = DE`
2. `language = en`
   `country = US`
3. `language = en`
   `country = GB`

See Appendix A.1 for a sample lesson file.

3.2.2 Extended configuration

Global options

Movable elements To specify the movable elements which can be used with SimTool, the keyword `movables` is used. Classes implementing movable elements must reside in the package `simulation` and their names must start with the prefix `Movable`.

The following movable elements are available at the moment:[34]:

- `BoxElement`: a movable element with a box icon
- `PalletElement`: a movable element with a pallet icon
- `AssemblyElement`: a movable element with nut icon
- `Car`: a movable element with a car icon

```
# globally enabled movable elements
# (box element only)
movables = BoxElement
```

Tools in the tool tree The keyword `tools` can be used to specify the tools which will appear in the tool tree of the application (or applet respectively). If the tool tree is visible[35], the following tools are available:

- `Select`: the selection tool
- `Link`: the link tool which is used to link to components
- `Erase`: the rubber which can be used to delete components

```
# tools in the tool tree
# (selection tool and eraser)
tools = Select,Erase
```

Menus and menu items A lot a functions of SimTool are accessible via the menu bar. It is often desired to restrict the number of available functions.[36]

[34]The `BoxElement` and `PalletElement` were contributed by Sandra Bangewitz and Manuela Bernhard

[35]The tool tree may be invisible if the applet parameter `TreeArea` was set to "false" (see Chapter 3.1.3) or if the menu item "show tool tree" has been disabled.

[36]If the applet parameter `Menu` is set to "false" (see Chapter 3.1.3), the settings are ignored and the menu bar is hidden completely.

Therefore, available menus and menu items can be specified with the following keywords:

- `menuFile`: file menu
  - `New`: menu item "new (empty) model"
  - `Save`: save model
  - `SaveAs`: save model using a different filename
  - `Exit`[37]: exit application

```
# menu items in the run menu
# (enable save and save as)
menuFile = Save,SaveAs
```

- `menuRun`: run menu
  - `Start`[38]: start the simulation
  - `Init`: initialise the simulation
  - `Step`: perform a step in the paused simulation

```
# menu items in the run menu
# (start, init and step)
menuRun = Start,Init,Step
```

- `menuView`: view menu
  - `Comment`: show/hide help panel
  - `Tree`: show/hide tool tree
  - `Toolbar`: show/hide tool bar
  - `Zel`: show/hide event list
  - `Grid`: show/hide grid
  - `GridOptions`: open grid options dialogue

[37]This menu item is disabled, if SimTool runs as an applet. Otherwise it would be possible to close the web browser via a menu item in the applet.

[38]This menu item becomes "pause simulation" once the simulation has been started.

```
# menu items in the view menu
# (show/hide event list, show/hide grid, grid options)
menuView = Zel,Grid,GridOptions
```

- `menuDebug`: debug menu
  - `EH`: show exception list
  - `ObjectList`: show object list

```
# debug menu
# (exception list and object list)
menuDebug = EH,ObjectList
```

- `menuHelp`: help menu
  - `About`: open about dialogue

```
# help menu
# (about)
helpMenu = About
```

Buttons in the tool bar If the tool bar is visible, the buttons that shall be visible in the toolbar can be specified with the keyword `buttons` to which the following items can be assigned:

- `Save`: save current model
- `Start`[39]: start simulation
- `Init`: initialise simulation
- `Step`[40]: perform a step in the simulation model
- `Simspeed`: slider which is used to set the speed
- `Zel`: show event list

[39]This button becomes the "pause" button if the simulation is running.
[40]This function works only if the simulation has been paused.

```
# buttons in the tool bar
# (enable save, start and slider)
buttons = Save,Start,Simspeed
```

Usually, it is a good idea to synchronise the availability of menu items with that one of buttons in the tool bar. Normally, the author's lesson should disable whole function packages and not single buttons or menu items.

Component specific options

Global options which apply to components (`distributions` and `routing`) can be overridden for components. Options specific to a component class start with the name of the component class (without package information and without the prefix `Com`). In the single station, for example, there are two properties setting distributions: process time and make ready time. The available distributions for the process time (make ready time) can be assigned to the keyword `SingleStation.proctime.distributions` (`SingleStation.mkreadytime.distributions`, respectively).

```
# set the available distributions
# for the process time property
SingleStation.proctime.distributions = Exponential

# set the available distributions
# for the make ready time property
SingleStation.mkreadytimes.distributions = Exponential
```

In the same way routing algorithms can be specified for component classes. A whole sample lesson file can be found in Appendix A.1.

```
# set the available routing algorithms
# for the single station
SingleStation.routing = RoundRobin
```

Pop-up menus The available menu items for the pop-up menus of the components can be specified with the *ComponentName*`.popup` keyword. Table 3.2 will show all available pop-up menu items for the components. The following example shows a pop-up menu entry of a single station.

```
# popup menu for single station
SingleStation.popup = Statistic,Delete,Move,Link,Properties
```

	Source	SingleStation	AssemblyStation	Link	Destination
Statistic	*	*	*	*	*
Delete	*	*	*	*	*
Move	*	*	*		*
Link	*	*	*		*
Properties	*	*	*	*	*

Table 3.2: Available pop-up menu items

Properties The property dialogues of the components contain one or more tabs on which the properties are displayed. The available properties depend on the component (see Table 3.3). It is only possible to enable properties which are available for that component. In general, there exist the following options:

- *Component*[41]`.props.general`: general properties
  - `name`: name of the component
  - `id`: ID of the component
  - `createelements`: type of movable elements that shall be created
  - `maxelements`: maximum number of movable elements that shall be created
  - `elementcount`: number (and type) of movable elements needed to assemble

```
# features in the general property tab
# (enable name and IDproperties)
SingleStation.props.general = name,id
```

[41]Class name of the component without package information and without the prefix `Com`

- *Component*`.props.run`: run properties
  - `mkreadytimes`: make ready time
  - `createInt`: interval between the creation of two movable elements
  - `proctime`: process time
  - `trtime`: transport time
  - `maxelements`: maximum number of movable elements which can be in the component at the same time
  - `destdel`: destroy time

```
# features in the run property tab
# (enable make ready time and process time properties)
SingleStation.props.run = mkreadytimes,prctime
```

- *Component*`.props.routing`: routing properties
  - `routing`: routing settings

```
# features in the routing property tab
# (enable the routing property)
SingleStation.props.routing = routing
```

	Source	SingleStation	AssemblyStation	Link	Destination
genaral tab					
name	*	*	*	*	*
id	*	*	*	*	*
createelements	*				
maxelements	*				
elementcount			*		
run tab					
mkreadytimes	*	*	*		*
createInt	*				
proctime	*	*			
trtime				*	
maxelemnts				*	
destdel					*
routing tab					
routing	*	*	*		

Table 3.3: Available properties

In case that the user starts SimTool without specifying a lesson file the default lesson file, provided by the system, will be used. This file is presented in Appendix G.1. Default values of the different properties of the components are listed in Appendix G.2

3.3 Modularity

As described above, mathematical probability distributions, movable elements and simulation components are implemented in a modular way. It is possible to add new distributions, new movable elements and new components without having to recompile the system. This ensures easy extensibility, in particular it is possible to spread the task of implementing new parts among various developers. Merging the results is very easy because the developers use well defined interfaces which act as guidelines for their work.

3.3.1 Guidelines for adding movable elements, distributions and simulation components

Adding movable elements

New movable elements can be added by deriving a new class from the base class `MovableElement`[42] (see Figure 3.11). The name of the new movable element class must start with the prefix `Movable`. Additionally, movable element classes must reside in the package `simulation`. The implemented movable elements only differ in their graphical representation. However, they will be treated differently by the assembly station. The visibility flag, the label of the movable element and the image icon shall be set by the constructor. The following example will show the constructor of a box element (`MovableBoxElement`):

```
public MovableBoxElement() {
    visible = true;
    label = "box";
```

[42] `simulation.MovableElement`

```
    setImage("images/BoxElement.gif");
}
```

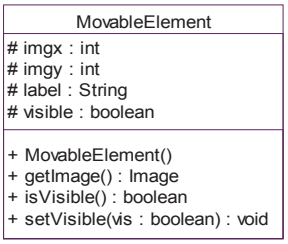

Figure 3.11: Selected methods and attributes of the base class of all movable elements

Adding distributions

To add a new mathematical probability distribution, it is necessary to create a new class which is derived from the abstract class `Distribution`[43] (see UML class diagram in Figure 3.9 on page 43). The name of the new class which must reside in the package `distribution`, must start with the prefix `Distribution`. Some abstract methods of the abstract superclass `Distribution`, described below, must be implemented by a new distribution class.

1. *Constructors*:
   (a) *Constructor without parameters*:
   This constructor has to set some reasonable default values.
   (b) *Constructor with a new seed value*:
   This constructor has to set reasonable default values and a new random seed value (`long`).
   (c) *Constructor with a string representing the distribution*:
   This constructor has to decode all parameters of the distribution from the string. This is necessary to enable the use of distribution objects in scripts (see Chapter 3.4.5).

[43] `distribution.Distribution`

2. `commit()`:
   Store the values that the user has entered in the property dialogue of the component.

3. `compareParams(...)`:
   This method will receive a vector with parameters which it will compare against the parameters of the distribution. The result is a `boolean` value.

4. `equals(...)`:
   Overrides `equals()` of `Object`[44]. The parameter of the method is an object. If this object is a distribution with the same parameters as the distribution on which the method is called, the objects will be considered equal.

5. `getName()`:
   This method returns the name of the distribution.

6. `getParameters()`:
   This method returns a string containing all needed information for a reconstruction of the distribution via the constructor with a string parameter.

7. `getPropertyPanel()`:
   A `JPanel`footnote`javax.swing.JPanel` which can be displayed in the property dialogue of a simulation component must be returned by this method.

8. `random()`:
   The core of the distribution is this method. It delivers the next random number according to this distribution. The returned number must be a `double` value.

9. `rollback()`:
   As opposed to `commit()`, this method undoes the changes the user has made.

To be able to use the new distribution, it must be enabled in a lesson file, so it can be found and loaded by the system. Therefore, the name of the distribution without package information and without the mandatory prefix `Distribution` must be assigned to the keyword `distributions` in the lesson file (see Chapter 3.2.1).

[44] `java.lang.Object`

Adding simulation components

Simulation components are derived from `BaseComponent`[45], the abstract superclass of components (see UML diagram in Figure 3.6 on page 38). This includes the links which are components as well, although they are different in appearance and handling. A new component must implement the following abstract methods:

1. `getDefaultNamePrefix()`:
   This method must return the default prefix of the name and the ID of a new instance of the component. This prefix is used when a user adds a new instance of this component class to the main panel. The ID and the name of the new instance will be compound of this prefix and the number of existing instances of this class.

2. `writeObject(...)`:
   This method generates the script commands that are necessary to recreate the component via a script. The XML document is the first parameter, the second one is the block element in which the generated script commands shall reside. Usually, the commands generated by this method will be a `createComponent` instructions and some calls to set methods. See Chapter 3.4 for more information on scripts.

3. `writeRouting(...)`:
   Like `writeObject`, this method is used to write script commands to an XML file to generate a script which can reconstruct the component. The routing information can only be set if all successor components in the simulation model are created. Therefore, this method is called after all `createComponent` instructions have been written to the script. That way it can be assured that all needed components exist when setting the routing properties.

In order to provide functionality, more methods should be implemented according to the desired use case of the component. Especially properties for the settings of the component must be added. The method `getDialog` of the superclass does not need to be overridden.[46] Every component has a matrix of its properties (`Property` objects) `simulation.property.Property`.

[45] `simulation.BaseComponent`
[46] see [DHH+02] p. 22

The property object does not only store the value of the property, it offers a `JPanel` with appropriate tools to change the property value. In case of the name property, this is a single text field with a label, for example. Each row of the matrix represents the tab in the property dialogue with the name specified in the first column. The properties that the components uses are stored in other columns. For more information about properties refer to Chapter 3.1.1.

Additionally, the event handling of the component should be implemented. At the moment, all components can handle the events they generate. But it is possible that a new component must deal with foreign events. The "normal" event handling behaviour is described in the following. A component receives a notification that a movable element has entered (method `enter(MovableElement me)`. That method immediately calculates the finishing time of the movable element and writes this point in time into the event list. When that point in time is reached, a new message will be sent to the component (`handleEvent`, a `leave` event in this case). The component will try to forward the movable element to its successor which is chosen by the selected routing algorithm (`canEnter(me)`, `enter(me)`). The following methods are used to shift movable elements from component to component:

1. `handleEvent(...)`:
   This method is called to start the forwarding process. It will use the actual routing algorithm to determine the successor component and invoke `canEnter` and `enter` on that component.

2. `canEnter(...)`:
   As with `enter`, this non abstract method should be overridden to be able to process movable elements in the component. It must return whether a movable element (given as a parameter) can enter the component or not. The default implementation in the superclass `BaseComponent` always returns `false`.

3. `enter(...)`:
   Although this is not an abstract method, it should be overridden in the new component, because the version implemented in `BaseComponent` always returns `false`. Otherwise, it would not be possible to process movable elements with the new component. This method should only be called after a successful call to `canEnter`.After a movable element has entered, it must be processed by the component.

3.4 Scripts and instructions

3.4.1 Demonstration mode

If a teacher wants to show a simulation model to students, he has three possibilities with SimTool 1.0: create a new model from scratch, load the complete model or load an incomplete model and finish it in front of the students. Loading a complete model does not show the students how to build models. The other ways can be quite time consuming tasks, depending on the size of the model. A demonstration mode for SimTool would be a useful feature.

In a web-based learning environment, the needs for a demonstration mode are even more obvious, because there is no teacher available who could build the models step by step. To use SimTool in such an environment, a mechanism to imitate the teacher's actions is necessary. The imitation of at least the following actions (instructions) is needed:

1. Add a new simulation component to the simulation model
2. Set the properties of a component
3. Move a component
4. Link two components
5. Remove components and links
6. Show the event list
7. Show the statistic window of a component
8. Start/stop/pause the simulation or perform a simulation step
9. Set the execution speed of the simulation run

In addition, it would be helpful to have commands to perform the following tasks:

1. Show message dialogues
2. Show help texts

3.4.2 Instruction hierarchy

This section describes the concept of instruction objects which are implemented for imitating a teacher's action by the execution of methods (for example script interface methods which can be called from a script) that perform the action on the involved component. Another way of imitating the teacher's action is the creation of specific instruction objects which will be shown in Figure 3.12. The base class of all instructions is the class `InstructionQualifier`[47]. It is derived from the abstract super class `Qualifier`[48] which represents a qualifier in a script. A qualifier in this context is an object that contains all necessary information to execute a script interface method. The most important method of `InstructionQualifier` is `perform()` which performs the instruction. The following example will show a qualifier in the script that refers to the script interface method `sc_setName`.

```
<instruction srcId="MySource" function="setName"
             classType="instance">
  <param name="name" type="stdType">
    <value>Entrance</value>
    <type>java.lang.String</type>
  </param>
</instruction>
```

An `InstructionQualifier` object represents a call of a script interface method. Any script interface method can be performed that way. Therefore, the `InstructionQualifier` is called "generic instruction qualifier". In some cases, it is quite difficult to implement an instruction as a script interface method. Then, a "specific instruction qualifier" has to be implemented. Such a specific qualifier must be derived from the generic qualifier (`InstructionQualifier`). At least, `perform` should be overridden. See Appendix D for a description of all implemented specific instruction qualifier.

[47] `actionset.InstructionQualifier`
[48] `actionset.Qualifier`

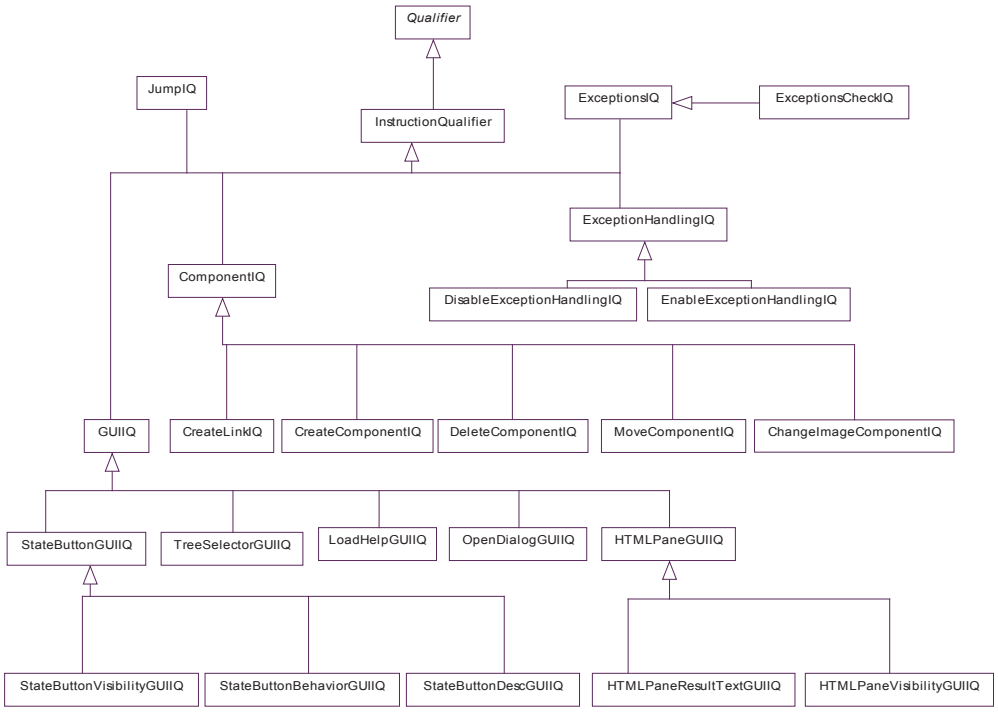

Figure 3.12: Hierarchy of instructions

3.4.3 Script language

Library vs. script language

There are two ways to implement instructions. They can be implemented as calls to methods of the objects in SimTool and can be offered in a Java API which is used for writing Java programmes that perform the desired tasks. Another possibility is to design a script language which contains exactly the desired features.

Both approaches have certain advantages and disadvantages. A Java API is easy to implement. As discussed above, in most cases the instructions mirror actions which a user can perform. That means methods to perform these tasks already exist and need to be called by the methods in the API only. Another advantage of this approach is that Java developers are already familiar with the language and can immediately start to write programmes using the API.

However, not all teachers who want to use the demonstration features of SimTool are Java developers. For those people, a script language is more appropriate because they do not have to learn a complete programming language in order to use the demonstration features. Furthermore, scripts do not need to be compiled. This is an advantage because scripts can be written with any text editor and are ready to use afterwards. Further software requirements (installed compiler, libraries, etc.) do not exist.

A major disadvantage of a script language is the need for a specific parser which must be implemented. Any change to the syntax of the script language means that you have to change the parser again. If a parser generator like CUP[49] or Parser[50] is used, this task can be done partly automatically. Parser generators use an unambiguous context free grammar as input and create a parser which accepts the given language.[51] However, if the semantics of the script language have changed, manual work is necessary.

XML as a format for a script language

Nowadays, XML is commonly used as a file format for data which are exchanged between applications. A huge number of tools exists to edit and process XML files. Nearly all important programming languages can deal with XML.[52] With JDK 1.4, a flexible API to process XML documents is available: JAXP (Java API for XML processing).[53] It includes the following standardised APIs:

1. Document Object Model (DOM) Level 2
2. Simple API for XML (SAX) 2.0
3. Extensible Stylesheet Language Transformations (XSLT) 1.0

The data structures in an XML document can be formally specified in an XML schema. Schema aware XML parsers (like Xerces2 Java Parser[54]) are

[49] see [Hud99]
[50] see [fok02]
[51] see [ASU86] p. 257
[52] see [Beh01]
[53] see [Sun01]
[54] see [Apa02]

able to check the contents of an XML document against its schema. Programmes using these parsers can rely on getting data from XML documents which matches the given XML schema.[55]

Although XML is a plain text format, it is hard to read for human beings. The structure of opening tags and closing tags is a bit confusing for some people. At the present time, this might be a problem. In the future, scripts shall be generated by an authoring tool (see Chapter 6.2.2). When the authoring tool is available, readability will not be an issue anymore.

Script file format

In this section, the file format of the script will be explained step by step using an example script. For a more formal description, see the XML schemas in Appendix E.

```
<rt:runtime xmlns:rt="http://www.simtool.de/scripts/runtime"
  xmlns:="http://www.simtool.de/scripts/simtool"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.simtool.de/scripts/runtime
                      http://www.simtool.de/scripts/runtime.xsd"
  name="example1">
```

The root element of the XML script file is named `runtime`. The attribute `name` can be used to name the script. In the example above, the script is named "example1". Its parameters `xmlns` are necessary to declare the namespaces (default namespace is `"http://www.simtool.de/scripts/simtool"`). This namespace contains useful data types which are used in the schema of the runtime script as well as in the schema of the exception set files (see Chapter 3.5.3). The whole schema can be found at Appendix E.1.

```
<rt:exceptionSets>
  http://www.simtool.edu/scripts/excepSet1
</rt:exceptionSets>
<rt:exceptionSets>
```

[55] see [Kle01]

```
  http://www.simtool.edu/scripts/excepSet2
</rt:exceptionSets>
```

A script can load one or more exception sets (see Chapter 3.5). They are specified inside `<rt:exceptionSets>` tags.

```
<rt:block label="main">
  <rt:action>
    ...
  </rt:action>
  <rt:action>
    ...
  </rt:action>
  ...
</rt:block>
<rt:block label="block2">
    ...
</rt:block>
```

A script consists of one or more execution blocks. Each block has a label (attribute `label`). The block with the label "main" will be executed when the script is started. The actions inside the block named "block2" are not performed unless they are the target of a `<jump>` instruction. An execution block consists of any number of instructions. If the block is executed, all its instructions will be performed sequentially.

```
<!-- first instruction -->
<rt:action>
  <newComponent name="MySource" className="Source"
                posx="10" posy="10"/>
</rt:action>
<!-- second instruction -->
<rt:action>
  <newComponent name="MyDestination" className="Destination"
                posx="600" posy="400"/>
</rt:action>
<!-- third instruction -->
<rt:action>
```

```
  <link src="MySource" dest="MyDestination" name="sillyLink"/>
</rt:action>
<!-- fourth instruction -->
<rt:action>
  <instruction srcId="MySource" function="setMaxME"
               classType="instance">
    <param name="maxME" type="stdType">
      <value>42</value>
      <type>java.lang.Long</type>
    </param>
  </instruction>
</rt:action>
<!-- fifth instruction -->
<rt:action>
  <enableExceptions/>
</rt:action>
```

Inside an `<rt:action>` tag, a single instruction can be specified. In the example, the first instruction performs a `<newComponent>` instruction which is used to add components to the simulation models. This instruction is associated with the specific instruction qualifier `CreateComponentIQ`[56] (see Chapter 3.4.2). Its attributes are the ID of the new component (`name`), the name of the component class without package information and without the prefix `Com` (`className`) and the coordinates in the main panel (`posx` and `posy`) where the component shall appear. In this example, a source component is added to the simulation model at the coordinates (10,10). It will receive the ID "MySource".

The second instruction adds a destination component. The third one links the source and the destination. The link will get the ID "sillyLink", because a source and a destination component should not be linked directly. Of course, `<newComponent>` and `<link>` are not the only available specific instruction. For a complete reference of all specific instructions see Appendix D.

A generic instruction qualifier is used in the fourth action. It can be distinguished from the specific instruction qualifiers through the `<instruction>` tag. This instruction represents a call to the script interface method `sc_setMaxME` (attribute `function`, represents the name of the method without the script prefix "sc_"). The attribute `scrId` is used to specify which ob-

[56] `actionset.CreateComponentIQ`

ject the method shall be performed on. In the example, the component with the ID "MySource" is the target of the method invocation. The `classType` attribute can be set to either "class" or "instance". If it is set to "instance" (like in the example), the method is performed on the given object as described. In case it is set to "class", the `srcId` attribute does not represent an object ID but a class name. The method is then called on all instances of the given class.

In the example, the generic instruction qualifier has one parameter (`maxME`), because the corresponding script interface method accepts one parameter: a `Long`[57] value. The attribute `type` of the `<params>` tag contains "stdType" if the class of the parameter has a constructor that accepts a `String`, so that the parameter object can be constructed from a `String` (see Chapter 3.4.5).

The way described above is the normal way of creating parameters. In addition to objects that can be constructed from a `String`[58], it is possible to use `Vector`[59] and `Class[]`[60]. If a special case applies, the `type` attribute has to be set to "vector" or "classArray", respectively. The application of non-standard parameters is shown in the following example. An instruction is presented there, which has two parameters: The first is a `Vector` and the second is a `Class[]`.

```
<instruction ...>
  <param type="vector">
  <!-- a vector containing a double and string value -->
    <value>20.0</value>
    <type>java.lang.Double</type>
    <value>number</value>
    <type>java.lang.String</type>
  </param>
  <param type="classArray">
  <!-- a class array with only one entry: Double -->
    <value>java.lang.Double</value>
    <type>java.lang.Class</type>
  </param>
</instruction>
```

[57] `java.lang.Long`
[58] `java.lang.String`
[59] `java.util.Vector`
[60] `java.lang.Class[]`

The fifth instruction contains the `<enableExceptions>` tag. It is associated with the `EnableExceptionHandlingIQ` and enables the exception handling. The last position of an execution block will be the best position for this instruction. If the exception handling is enabled, no further instructions of the script will be performed. Then the exception handling will have to determine where and when to continue (see Chapter 3.5).

3.4.4 Runtime engine

One or more instructions can be bundled into a block (`ExecutionBlock`[61]) that has a label which identifies it. There is a possibility to perform jumps from one block to another by specifying the label of the target block in a jump instruction (`JumpIQ`[62]). The runtime engine (`RunTimeEngine`[63]) is a collection of blocks (see Figure 3.13). The block labelled "main" will be executed if the method `run` of the runtime engine is called.

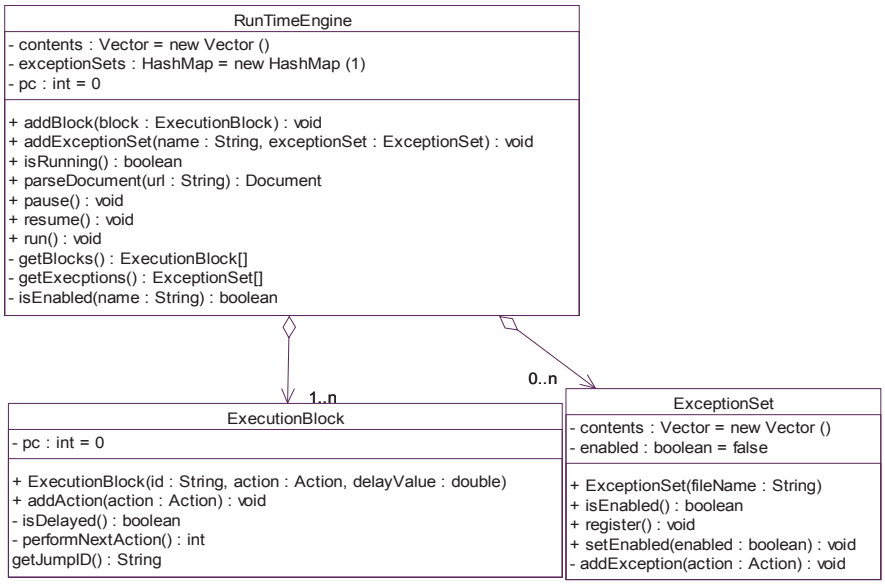

Figure 3.13: The runtime engine

[61] `runtime.ExecutionBlock`
[62] `actionset.JumpIQ`
[63] `runtime.RunTimeEngine`

3.4.5 Keeping extensibility in mind

Extensibility and modularity were major development aims of SimTool 1.0. These principles should be taken care of when adding new features. In this section, it is explained, how they influenced the design of the script language and the exception handling (see Chapter 3.5).

Adding script interface methods

Script interface methods are methods of simulation components or other objects of SimTool (system components) that can be called from scripts.[64] They must have a prefix (`sc_`) in order to be distinguished from other methods. For example, the method `sc_existsComponent` in class `simulation.Simulation` can be used to check if a specified simulation component exists in the simulation model.

```
public Boolean sc_existsComponent(String className) {
    Boolean result;
    Class compname=null;

    try {
        compname=Class.forName(className);
    }
    catch(ClassNotFoundException e) {
        System.err.println("sc_existsComponent: " +
                           "Cannot find class " +
                           className);
        result=new Boolean(false);
    }

    result=new Boolean(componentuse.containsKey(compname));
    return result;
}
```

The developer who wants to create new script interface methods for existing or new simulation components or static simulation objects has to observe

[64] This definition includes the use in condition parts of an exception.

the following requirement for script interface methods: The name of a script interface method must start with the prefix "sc_"[65].

A script interface method can be called with one or more parameters which are specified in the XML script. Due to the nature of the XML file, which is only able to store text (in Java represented by `String`) an additional type information has to be stored which is needed to cast the text value representing the value of the parameter to the proper object. The information has to be specified as the name of the corresponding Java class. A selection of the most important types will be shown in Table 3.4.

description	corresponding class in Java	example
text	`java.lang.String`	"Pizza"
floating point number	`java.lang.Double` `java.lang.Float`	5.03 5.045
whole nonnegative number	`java.lang.Integer`	10
constant value distribution (value=5.0)	`distribution.DistributionConst`	5.0
triangle distribution (upper bound=50, middle bound=10, lower bound=5)	`distribution.DistributionTriangle`	5,50,10

Table 3.4: Selected parameter types with their representations in Java

The mechanism described above is restricted to parameters that can be constructed from a text representation (`java.lang.String`). However, this mechanism is still very powerful, especially in terms of the extensibility of the system. Own objects can be passed to script interface methods as long as there exists a constructor which builds the object from a string.

Another way to pass parameters is the definition of special cases. Until now, there are two additional parameter types available: vector and class array. These additional cases are distinguished from the normal case in the script

[65] This prefix is specified by the static final variable `SCRIPTPREFIX` in the `SimTool` object. It is not recommended to change it.

via the `type` attribute of the `<params>` tag. The script file and its tags are explained in Chapter 3.4.3 in detail.

The return value of the script interface method will be compared to a specified value exactly in the same way the parameter comparison is handled. The return value is specified in the script as a text object. The value returned by the script method has to be of a type that is constructible from a string representation. Thus, the class of this return object needs a constructor that works on `String`.

If a script interface method is added to a component, neither the script parser nor the runtime engine for scripts will need to be custom-tailored. However, SimTool must be recompiled, because it is not possible to add them as a plug-in like simulation components.

3.5 Exceptions

3.5.1 Motivation

The aim of the system is helping teachers to teach complex facts about simulation by using interactive examples. These examples can be built as tutorials where the student has to perform tasks. A deeper understanding of the subject Simulation will require the learner to understand not only how the current example works. The learner must also understand the mathematical mechanisms that simulation is based on. To support the learning process, the system will work as an assistant for the user. Therefore, the system must be able to react on specific user actions.

A possible way to implement this desired behaviour is allowing to attach conditions to instructions. An instruction would be performed only if its conditions were fulfilled. Settings of the components and model properties could be verified. If they were not correct, an instruction would be performed. Of course, an instruction can also be performed in the opposite case, if the settings were correct.

There are two possibilities to handle the case of an unfulfilled condition. The first one is not performing the associated instruction and continuing with the

next one. That means the point in time of the condition check is fixed and the check would be performed only once. It would not be possible to express rules like "Every time the user sets the maximum capacity of the link 'Link2' to a value greater than x". The second alternative is waiting for the condition to be fulfilled. This could be used to specify the end of a tutorial ("Wait until the user has done X and display a final message"). However, the system would not be able to produce feedback to the user's actions because it would be occupied in the while loop. Further conditions could not be checked.

In order to support the user in a tutorial, a more powerful concept of rules is needed: exception. An exception is a combination of an instruction and conditions. If the conditions of an exception are fulfilled, the associated instruction will be performed. The process of checking the conditions of an exceptions is called performing or checking an exception.

One of the main design strategies of the system is extensibility. Therefore, the exception handling mechanism has to be implemented in a modular way. Exceptions can be seen as modules which themselves consist of smaller units (conditions and instruction). This will help to realise complex tutorials, because the teacher can choose from a variety of script interface methods which can be used in the condition tests. It is also possible to add more methods easily without having to change the implementation of the exception handling mechanism (see Chapter 3.4.5).

3.5.2 Working with more than one condition

An instruction of an exception with one condition will be performed if its condition is fulfilled. But an exception can consist of more than one condition, so a mechanism is needed to connect the different conditions to each other using logical operators. To guarantee the precedence of these logical operators if there are more than two conditions, a pre-eminence mechanism must be applied. The system uses brackets as known in the mathematics. For reflecting mathematical brackets, the different conditions are stored in a container (`ConditionContainer`) with the logical operator which has to be applied to all of these conditions. In terms of mathematics, this container reflects values between two brackets. Furthermore, a pre-eminence mechanism needs to work with nested brackets. Therefore, a container can store sub-containers (see Figure 3.14). To guarantee the correct order of the condition parts when performing a check, each container will be checked at

once. The check will be done recursively and does not differ from the normal mathematical behaviour. Provided logical connectors are:

- `OR`
- `AND`
- `NOT OR`
- `NOT AND`
- `XOR`

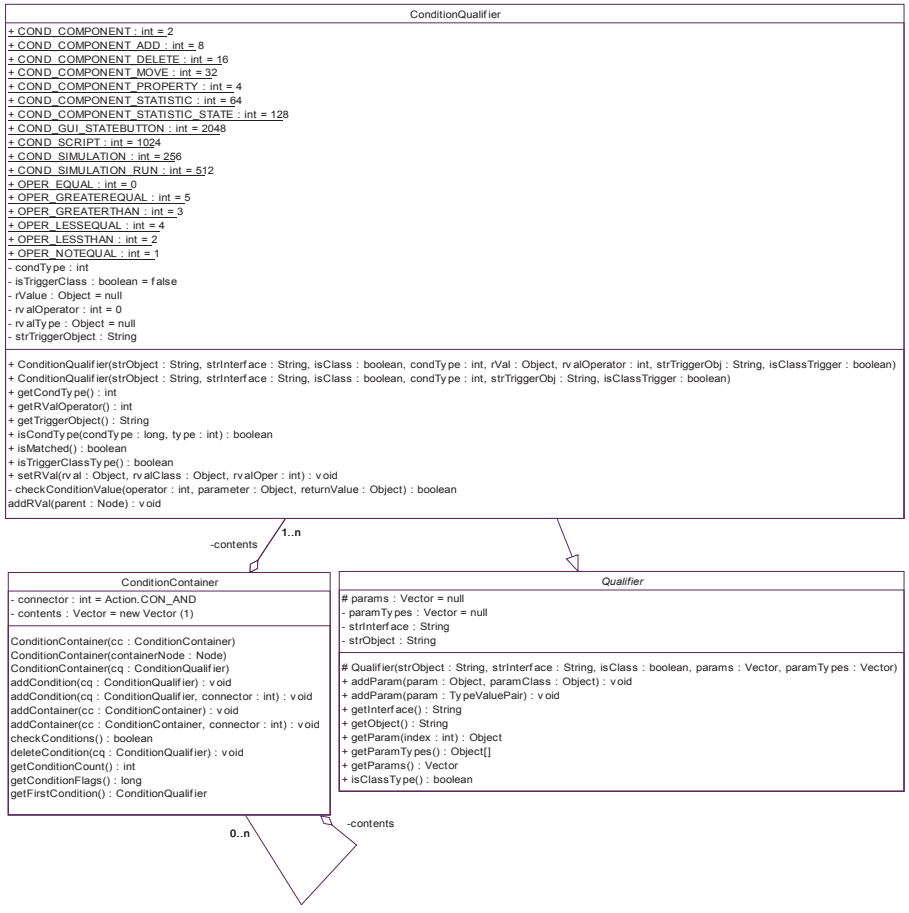

Figure 3.14: The objects that represents and stores conditions with some of the methods and attributes

```
<conditions connector="and">
<!-- c1 -->
  <condition srcId="SingleStation1" function="getCompName"
             classType="instance" condType="componentProperty">
    <returnValue compareOperator="==" type="stdType">
      <value>Station 1</value>
      <type>java.lang.String</type>
    </returnValue>
  </condition>
  <conditions connector="or">
    <!-- c2 -->
    <condition srcId="simulation.Simulation"
               function="existsComponent"
               classType="class" condType="componentAdd">
      <param>
        <value>simulation.ComDestination</value>
        <type>java.lang.String</type>
      </param>
      <returnValue compareOperator="==">
        <value>simulation.ComSource</value>
        <type>java.lang.String</type>
      </returnValue>
    </condition>
    <!-- c3 -->
    <condition srcId="simulation.Simulation"
               function="existsComponent"
               classType="class" condType="componentAdd">
      <param>
        <value>simulation.ComDestination</value>
        <type>java.lang.String</type>
      </param>
      <returnValue compareOperator="==">
        <value>true</value>
        <type>java.lang.Boolean</type>
      </returnValue>
    </condition>
  </conditions>
</conditions>
```

The script example above shows a nested structure of brackets in an XML script which specifies an exception set. It represents the structure in Equation 3.1. The learners task that has to be done in order to fulfil the condition would be changing the name of the single station represented by the ID "SingleStation1" to "Station 1" and adding a source or destination component. The condition c_1 represents a call to the script interface method `sc_getCompName()`. It checks whether the name of the single station is set to "Station 1". Conditions c_2 and c_3 check whether there is a source component or a destination component, respectively.

$$c_1 \wedge (c_2 \vee c_3) \tag{3.1}$$

The structure is displayed in Figure 3.15.

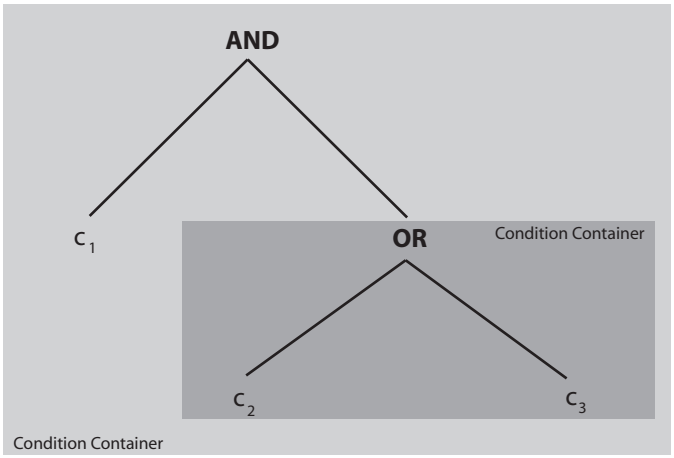

Figure 3.15: The internal structure of the example

As seen in the example, the connector will be specified as the attribute `connector` of the `<conditions>` element. There is a difference between the tags `<conditions>` and `<condition>`. The element with the `<conditions>` tag will describe a condition container whereas the element that is specified inside a `<condition>` tag will describe the actual condition itself.

3.5.3 Exception set file format

In this section, the file format of the exception set files is explained in detail using an example exception set. For a more formal description, see the XML schemas in Appendix E.

```
<es:exceptionSet
  xmlns:es="http://www.simtool.edu/scripts/exceptionSet"
  xmlns:="http://www.simtool.edu/scripts/simtool"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
   "http://www.simtool.edu/scripts/exceptionSet
    http://www.simtool.edu/scripts/exceptionSet.xsd"
  id="exampleSet1">
```

The root element of the XML exception set file is named `exceptionSet`. The parameter `id` can be used to give the script a unique ID. In the example above, the script has the id "exampleSet1". The parameters `xmlns` of the root element are necessary to declare the namespaces. The default namespace is `"http://www.simtool.de/scripts/simtool"`. This namespace contains useful data types which are used in the schema of exception set files as well as in the schema of the runtime script (see Chapter 3.4.3). The whole schema can be found at Appendix E.1.

An exception set file consists of one or more exceptions which are represented by the tag `<es:action>`.

```
<es:action name="checkEntranceSettings">
  <showDialog type="info"
              text="The source produces too few elements."/>
  <conditions connector="and">
    <condition srcId="Entrance" function="getMaxME"
               classType="instance"
               condType="componentProperty">
      <returnValue compareOperator="&lt;" type="stdType">
        <value>900</value>
        <type>java.lang.Long</type>
      </returnValue>
```

```
    </condition>
    <condition srcId="Entrance" function="getMaxME"
               classType="instance"
               condType="componentProperty">
      <returnValue compareOperator="&gt;" type="stdType">
        <value>0</value>
        <type>java.lang.Long</type>
      </returnValue>
    </condition>
  </conditions>
</es:action>
```

Inside an `<es:action>` tag, a single instruction can be specified. This instruction will be performed if the conditions are fulfilled. In the example, the exception performs a `<showDialog>` instruction which is used to show a dialogue. This instruction is associated with the specific action qualifier `OpenDialogGUIIQ` (see Chapter 3.4.2).

The conditions, which the exception are based on, are stored in a condition container (tag `<conditions>` and connected by `AND`). Therefore, the `connector` attribute is set to "and". Two conditions are stored inside the container.

The first condition will invoke `sc_getMaxME` on the component with the ID "Entrance". A single instance is involved here, so the `classType` attribute is set to "instance". The condition will be checked whenever the learner closes the property dialogue using the OK button, because the condition type (`condType` attribute) is set to "componentProperty". The result of the script interface method will be compared to the value specified by the `<returnValue>` tag (a `Long` with the value 900). The operator for that comparison is < (`compareOperator` attribute is set to "<").

The second condition uses the same script interface method of the same object. It checks whether `sc_getMaxME` returns a value greater than 0. Therefore, in the `<returnValue>` tag, the `compareOperator` attribute is set to ">".

Script interface methods used by conditions can have parameters. The specification of the parameters works in the same way as for the parameters of script interface methods used by instructions (see Chapter 3.4.3). The

mechanism of the construction the parameters described there ("stdType", "vector" and "classArray") apply to return values as well. In the example above, the standard mechanism has been used (`type` attribute: "stdType").

3.5.4 Point of time of checking conditions

One of the most important questions is the point of time of checking conditions. Conditions are triggered by other objects in the system. A system that has to react to user interactions has to be very powerful in terms of the exception handling. The user can act in a very unpredictable way and the system still has to be able to react to such a behaviour. As described in Chapter 3.4.3, objects can be accessed through its component IDs. A set of objects (of one type) can be accessed by its class name. The object representation mechanism is also used for the definition of the object that trigger the condition check, further on referred to as the triggering object. If a class name is used instead of a component ID, all objects that are instances of the class specified by the class name will perform the condition check. The triggering object triggers the check for fulfilled conditions at a special point of time which depends on the type of the condition. Especially, due to performance issues, it is not practicable to perform a condition test at every point of time for each possible condition. The following types are implemented:

- *Component property conditions*:
  Conditions of this type are checked when the property dialogue of an component is closed. Typical conditions of that type are conditions that compare the values describing the simulation component (specified in the property dialogue) with values specified in the script.
- *Component add conditions*:
  These conditions are checked when a new component is added to the simulation model.
- *Component delete conditions*:
  Conditions of this type are checked when a component is deleted from the simulation model.
- *Component move conditions*:
  The conditions that belong to this group are checked when a simulation component is moved.

- *Component statistic conditions*:
  This type allows the check of statistical values. Conditions of this type are verified when the statistical values of an component might change.

- *Simulation conditions*:
  Simulation conditions are checked during the simulation run. An example of such a condition is a check on the simulation time.

- *Script conditions*:
  Script conditions are checked by the runtime script.

- *State button conditions*:
  State button conditions will be checked when the state button is pressed. For further information about the state button see Chapter 3.6.2.

The system is planned as a tool which allows the user to "play" with simulation models. This is done by building and running simulation networks to determine dependencies that apply for simulation in general and for the simulation components in particular. Therefore, most of the time the triggering object will be a simulation component. The modularity and extensibility of the system lead to certain restrictions for simulation components (discussed in Chapter 3.3). That means anyone implementing new components must take care of the correct point in times when to trigger a condition test. The mechanism in the base class of all simulation components (`BaseComponent`) should not be changed unless there is a good reason to do so. Due to performance issues, other restrictions to this triggering concept apply.

For example, it does not make any sense to check conditions working on statistical information when the statistic is not running. At the present time, statistic values are not stored so that statistical values can only be checked if the simulation is running (and with it the statistic).

An assumption for the condition check is that the exception handling mechanism is enabled and the object that wants to trigger the check is registered. For information about the registration of exceptions and objects see Chapter 4.3. The biggest part of that mechanism is implemented in the base class of a simulation component (`BaseComponent`[66]). As there is the restriction that every new component added to the system has to be derived from

[66]simulation.BaseComponent

`BaseComponent`, a lot of work in the sense of the implementation process is already done. Table 3.5 will show the point of time when a condition test will be triggered and the different condition types implemented in the system for the simulation components.

Condition types have to be specified in the script as the attribute `condType` to the element with the `<condition>` tag. The data type of that attribute is a list of well defined strings, as the following excerpt of the XML schema will show. The whole schema can be found in Appendix E.1.

```
<xsd:element name="condition">
  ...
  <xsd:attribute name="condType">
    <xsd:simpleType>
      <xsd:list itemType="condTypeEnum"/>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:element>

<xsd:simpleType name="condTypeEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="component"/>
    <xsd:enumeration value="componentProperty"/>
    <xsd:enumeration value="componentAdd"/>
    <xsd:enumeration value="componentDelete"/>
    <xsd:enumeration value="componentMove"/>
    <xsd:enumeration value="componentStatistic"/>
    <xsd:enumeration value="componentStatisticState"/>
    <xsd:enumeration value="script"/>
    <xsd:enumeration value="simulation"/>
    <xsd:enumeration value="simulationRun"/>
    <xsd:enumeration value="guiStatebutton"/>
  </xsd:restriction>
</xsd:simpleType>
```

The following example will show two conditions. One has the condition type `componentProperty`, the other one has the condition types `componentAdd` and `componentDelete`. If more than one condition type is specified they must be separated by a blank character.

condition type	ID	trigger	point of time	description
component	2	BaseComponent	paint()	during a repaint of the window
componentProperty	4	BaseComponent	getDialog()	when the property dialog is closed
componentAdd	8	Simulation	addLink()	when a link is added to the simulation model
			addComponent()	when a component is added to the simulation model
componentDelete	16	Simulation	removeComponent()	when a component is deleted from the simulation model
componentMove	32	MainPanel	moveComponent()	when the user moves a component on the screen
componentStatistic	64	BaseComponent	paint()	during a repaint
componentStatisticState	128	BaseComponent	addStatValue()	during a statistic state change
simulation	256	SimRun	run()	after the simulation started
simulationRun	512	SimToolWin	runSimulation()	while the simulation runs
script	1024	ExceptionsCheckIQ	perform()	check exception instruction
guiStatebutton	2048	StateButton	actionPerformed()	if button pressed

Table 3.5: Mapping between condition types and the point of time of the condition check

```
<condition ... condType="componentProperty">
  ...
</condition>
<condition ... condType="componentAdd componentDelete">
  ...
</condition>
```

3.5.5 Invocation process

For understanding the internal structure of the conditions and the instructions that will be invoked when the condition is fulfilled, a closer look to different possibilities of invoking a condition test must be taken. In general, objects of the system can perform a condition test. Therefore, they compare their properties with the values specified in the condition. If they match according to a compare operator, specified in the script, an instruction will be performed. Such a statement could be a message to the user or some changes to the components in the simulation network. Any instruction that can be used in runtime scripts can be used in exceptions as well. However, it is not recommended to change anything in the simulation model without informing the user.

An exception is only able to perform one instruction. If more than one instruction shall be executed they should be bundled into an execution block stored in the runtime script. The instruction stored in that exception would be a jump to the block containing the bundled instructions. Figure 3.16 will illustrate the invocation process.

The exception handling will by started by a triggering object which asks the `ExceptionLookup`[67] object for a set of exceptions that need to be checked. That look-up object determines these exceptions. Therefore, the condition type that is associated with the actual point of time in the runtime of the object is used. The look-up object returns a set of exceptions that are based on conditions having that condition type.

The exception handling performs a condition test on each of the returned exceptions. Thus, the exception retrieves the object that has to be checked and invokes the script interface method, associated with the condition, on

[67] `exception.ExceptionLookup`

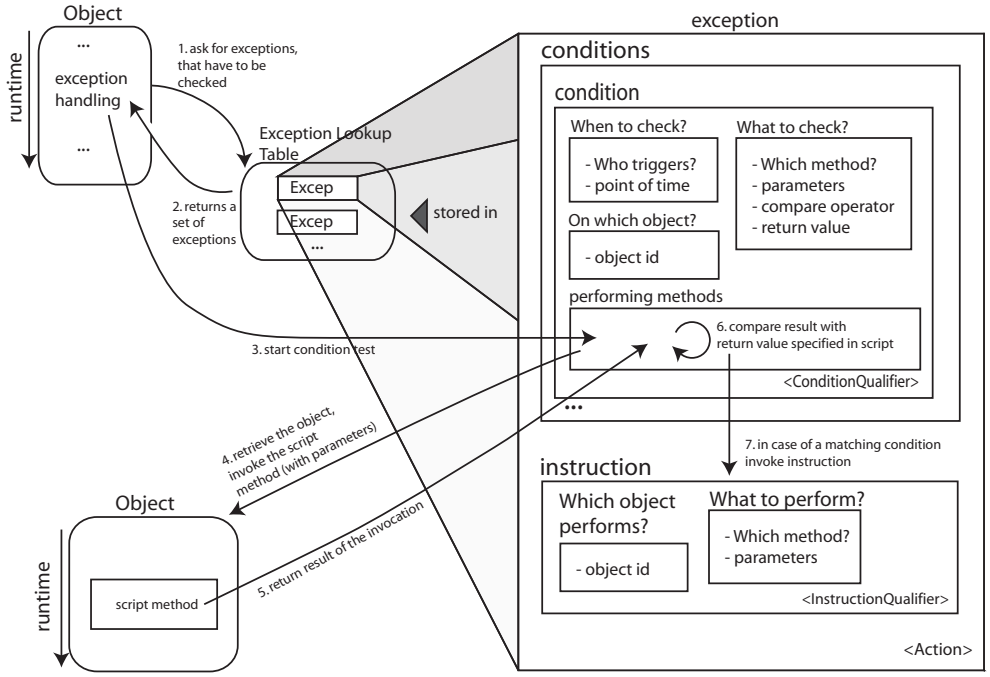

Figure 3.16: Exceptions - The invocation process

that object using the specified parameters. The object being checked will return the result of this operation. After that, this return value will be compared to the return value specified in the script.

If all the results of the different condition tests performed on the different conditions of the exception match according to the given compare operators the instruction of the exception will be performed (in case of an `AND` connection of the conditions). In case of an `OR` connection, the instruction will be performed if one of the conditions is fulfilled. Note that conditions can be structured in a more complex using nested condition containers (see Chapter 3.5.2).

3.5.6 Qualifying an object

The qualification of the object which performs the condition test[68] and contains the script interface method is a very important issue. An object is referenced by its ID or its class name (representation of the object) specified in a `ConditionQualifier`[69]. This representation is used to find the object and to perform the condition test on it. The condition test itself is done by invoking a script interface method (held by the object). In the sense of SimTool, there are two different object types. Objects can be either simulation components (derived from `BaseComponent`) or system components(for example the main window or the simulation engine). Furthermore, it should be possible to define sets of simulation components. This behaviour can be achieved by using an object representation that refers to a class: representation type class. The following example will show why this is useful and needed in order to provide exception support for user added components. Table 3.6 will show the important components that already contains script interfaces with its types and representation qualifiers.

type	object representation (ID)	class representation
simulation components		
	ComSingleStation$x$	simulation.ComSingleStation
	ComSource$x$	simulation.ComSource
	ComDestination$x$	simulation.ComDestination
	ComAssemblyStation$x$	simulation.ComAssemblyStation
	ComLink$x$	simulation.ComLink
system components		
	-	simulation.Simulation
	-	gui.SimToolWin

Table 3.6: The important components that provides script interfaces

Supposing that there is a simulation network that has four single stations connected to a source and destination component (see Figure 3.17). The learner has to add components so that the retention time of movable elements in the simulation is smaller than 10 minutes. The learner is free to act as long as he does not delete parts of the given simulation network. This tutorial will be assisted by a set of exceptions which tries to guide the learner to a proper solution of the problem. The exception set must be written in advance. Presuming that there is only one way to specify the object (in this case a simulation component), the ID. Then the question arises how to react

[68] Note that this object does not have to trigger the test.

[69] actionset.ConditionQualifier

on user interaction that involves new components which have been added by the user.

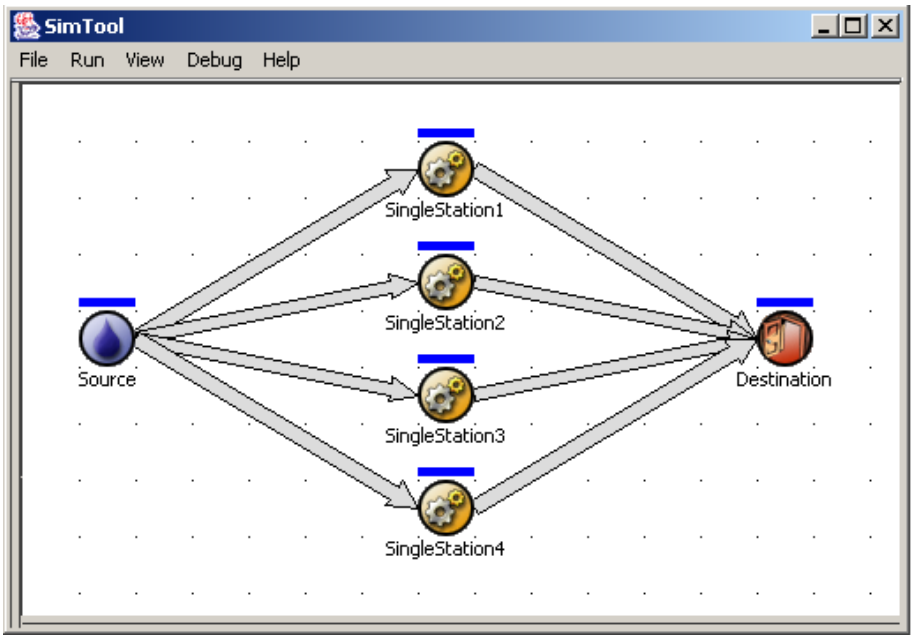

Figure 3.17: Example simulation network

This problem can be partly solved by knowing the ID naming process. User added components receive an ID from the system automatically. That enables the teacher to predict the next ID that will be given by the system. The IDs consist of the result of the `getDefaultPrefix`[70] method of the component and the number of instances of that class existing in the simulation model (including the added component). In the example, a fifth single station added by the user would receive the ID "SingleStation5". Understanding the naming process allows the teacher to add exceptions to the set that work on future components. This is possible because exceptions can be created, involving objects that do not yet exist. Only IDs are stored in the exceptions. The check process of a condition is started by a component. At this point of time, it is tried to resolve the IDs and to get object references from the `ObjectLookup` table. However, this prediction of IDs does not work well, because the teacher has to have a lot of background knowledge about the internal naming process. Normally, these internals are a subject to change in future. This would lead to unusable exception sets.

[70]This method will normally return the class name of the component.

Another possibility to solve the problem is to prohibit the adding of new components or the creation of exceptions during runtime. However, introducing more prohibitions does not seem to be useful, because it would restrict the user too much. The creation of new exceptions at runtime is not useful at all, because the teacher has to control the teaching process. That control will not longer be guaranteed if the student is able to define his own rules.

As a conclusion to that problem, another reference mode is introduced: class, which allows the specification of a class by its name. All objects that are instances of this class will have to perform the condition test. If one of these objects fulfil the condition, the exception will be raised and the instruction will be executed. However, the order in which the objects are checked cannot be guaranteed. This qualification mechanism is powerful because it allows the user to define sets of objects on which the conditions are verified.

In the example described above, a condition for all single stations would be desired. Therefore, the class name of the single station components (`simulation.ComSingleStation`) will be stored in the condition. A condition could be something like: "check if the process time is a constant value greater than 10". If the condition is fulfilled, a message could be displayed, which informs the user that it is not possible to reach a retention time for movable elements smaller than 10 if the process time of a single station is already set to 10.

3.5.7 Limiting the number of times an exception can be raised

A condition will trigger an instruction if it is fulfilled. There exist certain conditions that are always fulfilled after some time. Thus, after a certain time, whenever the condition is checked, the exception might be raised and the instruction might be performed. If the condition is verified too often, this can lead to problems as the following example will show.

Presuming that there is a single station a condition is based on, checking if the number of processed movable elements is greater than 10. Supposing that the simulation model is running, the exception is only based on this condition and there have already ten elements passed the station, the condition will be verified and the exception will be raised whenever a new element passes.

This behaviour can slow down the whole system and annoy the user. If the instruction associated with this exception is showing a message dialogue informing the user that 10 movable elements were processed by the simulation model, this message would be shown at every time an element leaves the single station.

As long as the simulation speed is set to slow this could be practical. But if the simulation is running at high speed the system will open a new dialogue immediately after the last one has been closed by the user. Therefore, it is necessary to be able to specify the maximum number of times that an exception should be fired. The problem in the example will be solved if the exception is only raising once. This maximum number is specified in the script as the `maxCalls` attribute to the exception (tag `<action>`) as an `int` value in the following way.

```
<es:action ...  maxCalls="3">
  ...
</es:action>
```

If an exception is performed `maxCalls` times, it will be deleted from the `ExceptionLookup` table. That guarantees that the condition of the exception will never be verified again as long as the `ExceptionLookup` object is not reinited. For further information about the look-up objects see Chapter 3.7.

The attribute `minCalls` specifies the minimum number of calls of an exception. This construct can be used to react to repeated errors of the learner. A message could be displayed if the user had repeated an error three times (`minCalls="3"`). The instruction will be performed if the conditions are fulfilled for the fourth time. After that invocation the exception will act in the normal way.

3.5.8 Conditions in detail

An exception has to be based on at least one condition. The main task of a condition is the invocation of a script interface method. The result of that invocation will be compared to a value specified in the script. This is the only difference to the execution of an instruction indeed. Therefore,

the condition can be seen as a special case of an instruction with an additional compare operator and return value. The condition is reflected in the `ConditionQualifier` object which is derived form the `Qualifier`[71] object which holds the representation of the task. Due to this object oriented modular structure new qualifiers can be derived based on the `ConditionQualifier` class. The matching-check of the condition is done in the `isMatched` method. This method can be overwritten for future condition qualifiers to ensure another checking behaviour.

The following XML code shows a condition.

```
<condition srcId="simulation.Simulation"
           function="existsComponent"
           classType="class" condType="componentAdd">
  <param type="stdType">
    <value>simulation.ComSource</value>
    <type>java.lang.String</type>
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>false</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

That condition acts on the `simulation.Simulation` object by the invocation of the `existsComponent` method. This script interface method will check whether a specified simulation component exists in the simulation model and will return a `java.lang.Boolean`. This value will be compared to the value specified in the script ("false"). A compare operator "==" will be used (all available compare operators are shown in Table 3.7). In other words, the condition asks the simulation if the simulation network contains a certain component ("Source"). In the example, the exception will fire if there is not such a component in the network.

[71] `actionset.Qualifier`

operator (in scripts)	operator (in Java)	operator (mathematical)	description
==	==	=	equal
!=	!=	≠	not equal
&lt;	<	<	less than
<=	<=	≤	less or equal
>=	>=	≥	greater or equal
&gt;	>	>	greater than

Table 3.7: Available compare operators

3.5.9 Instructions

An exception will perform a instruction if its conditions are fulfilled. Instructions are based on the `Qualifier` object and implemented in the `InstructionQualifier` object . The instructions stored in an exception are the same as the instructions used in the runtime script (see Chapter 3.4.2).

3.6 More user interaction

3.6.1 The help panel

Feedback is an essential issue in e-learning systems. The possibilities for the feedback that dialogues offer are quite limited. After having confirmed the dialogue the learner does not have the possibility to read the presented text again. In addition, it is not convenient to present long texts, such as the text describing the task the learner shall perform, in a dialogue.

The help panel at the bottom of the window of SimTool can be used not only to present information about the selected tool in the tool tree (see Chapter 3.1.3), it can also be used to present the tasks that the learner shall perform or to give additional feedback resulting from a user's action.

Not all information shall be displayed in the same pane of the help panel, so the panel is divided into three different panes:

1. *Help pane*:
   The help pane is used to display general help on components and tools or the task of the lesson.

2. *Result pane*:
   The result pane is used to display verbose result messages.

3. *Detailed help pane*:
   The detailed help pane is used to display detailed help. Links which appear in one of the panes will be opened in that pane.

Loading help files

The first step of a lesson should be the presentation of the tasks that the learners have to perform. The tasks should be displayed in the help panel of SimTool. This can be done by the `<loadDescriptionFile>` instruction whose usage will be shown in the XML excerpt below.

```
<rt:action>
  <loadDescriptionFile path="lesson/task"/>
</rt:action>
```

The instruction is able to load help files written in HTML in the help pane of the help panel. This can either be done by specifying a relative path (like in the example above) or by specifying a valid URL. Security restrictions are not relevant, as long as the files describing the task and assisting the lesson are a part of SimTool (accessible through the specification of a relative path). In the other case the system need read access to the directory. Therefore read permissions must be given in the policy file (see Section "Required permissions" on page 50). The URL referencing mode is especially useful as the script author is usually not the same person as the provider of the SimTool system. Thus, the author normally does not have any access to the directories where SimTool resides in. Therefore, script files are often located on another web space on another computer. The language localisation of help files is assisted in case of the usage of a relative path.

Following restrictions apply for the specification of a relative path:

- The HTML file needs to be named "index.html".
- The file has to be stored under the relative path "help/*language*/".

The example above refers to a HTML file named "index.html" that is stored under the relative path "help/*language*/lesson/task", where *language* is the actual language of the system. It is the responsibility of the script author to provide localised versions of the files in the desired directories. Note that the system will not provide any localisation help in case of the specification of help contents via a URL. The author is liable for this process.

The instruction that loads a specific HTML file into the help pane is implemented in the class `LoadHelpGUIIQ`[72]. This class is a direct subclass of `GUIIQ`, because the change of the contents of a GUI element (the help panel) can be considered as a GUI instruction.

Controlling the visibility of the help panes

Information are displayed in different panes of the help panel. Therefore, an instruction is needed that controls the visibility of these panes. This instruction is called `<visibilityHTMLPane>`. It is implemented in a subclass of `GUIIQ`[73] called `HTMLPaneVisibilityGUIIQ`[74]. See Appendix D.13 for further reference on the usage of this instruction.

Adding text to the result pane

For providing a higher feedback level `<setTextResultTab>` is used. It adds text or HTML statements to the result pane of the help panel and is implemented in the class `HTMLPaneResultTextGUIIQ`[75].

[72] `actionset.LoadHelpGUIIQ`
[73] `actionset.GUIIQ`
[74] `actionset.HTMLPaneVisibilityGUIIQ`
[75] `actionset.HTMLPaneResultTextGUIIQ`

The following example will show the usage of that instruction.

```
<rt:action>
  <setTextResultTab
    text="&lt;A HREF='component/simulation.ComSource'
          &gt;Entrance&lt;/A&gt;"
    mode="add"/>
</rt:action>
```

The example above adds the text "Entrance" to the result pane, coded as a hyper-link which will refer to a help document that describes the situation. The path of that help document can be set as a relative path or it can be defined as a valid URL. The contents of the help document will be displayed in the detailed help pane. The link used in the example will open the description of the source component when clicked on it.

3.6.2 The state button

The exception handling concept allows to specify exceptions that describe the solved task of a lesson. These exceptions can be checked all the time. If the user corrects the last wrong setting such that the task is solved the system will react immediately (by congratulating the learner, for example). However, sometimes it is desired not to verify if the task is solved until the learner explicitly declares to have finished. Therefore, a mechanism is used which provides a way for the learner to notify the system that the learner thinks to have solved the given task: the "finish" button.

The "finish" button can be connected to an exception. Thus, the button will trigger conditions which will lead to the execution of instructions if they are fulfilled. The button can be configured by the runtime script. Therefore, additional instructions which hide or show the button and change its label are necessary. The appearance of the button is not controlled by the lesson file, because the button is designed to change at runtime (script controlled). The `SimToolWin` object which holds the button in the toolbar will check for conditions with the condition ID `COND_GUI_STATEBUTTON` if the button is pressed.

The button is implemented in the class `StateButton`[76]. This class is called `StateButton` and not `FinishButton`, because the button can be used in other cases as well. For example, it is possible to create a demonstration in a way that the user can click on the button in order to pause or resume the demonstration.

It is possible to use the button more than once in a lesson with a different behaviour. Therefore, it is necessary to be able to change the behaviour of the button at runtime. That means it must be possible to change the exceptions that shall be triggered by the state button. Using `maxCalls` (see Chapter 3.5.7) does not suffice here, because the number of times the button is pressed before its behaviour shall change cannot always be predicted.

The range of exceptions that will be checked if the button is pressed will be specified using the name of an exception or the name of the exception set that the exceptions have to be stored in. Following instructions that control the state button are available:

- `<behaviorStateButton>`:
  This instruction specifies the exceptions that are triggered by the state button.
- `<descriptionStateButton>`:
  This instruction changes the label and the tooltip text of the state button.
- `<visibilityStateButton>`:
  This instruction changes the visibility of the state button.

The class `StateButtonGUIIQ`[77] is the base class of all state button related instructions. The class initialises the variables that are necessary to access the button through the `SimToolWin` object. It is derived from `GUIIQ`, because the task button is part of the GUI.

[76] `gui.StateButton`
[77] `actionset.StateButtonGUIIQ`

The <behaviorStateButton> instruction

The instruction <behaviorStateButton> is used to specify the behaviour of the button. It is implemented in the class StateButtonBehaviourGUIIQ[78]. If the button is pressed by the learner, a condition look-up will be started. The SimToolWin[79] is the triggering object and the relevant condition type is COND_GUI_STATEBUTTON. The resulting set of exceptions retrieved can be reduced to exceptions specified by its name or by the name of the exception sets they reside in. The specification of exception names or exception set names is optional. The following example will initiate a look-up of the exceptions stored in the exception set "firstExceptionSet". The set of resulting exceptions will be searched for the exception with the name "Excep67".

```
<rt:action>
  <behaviorStateButton>
    <excepSet>firstExceptionSet</excepSet>
    <excep>Excep67</excep>
  </behaviorStateButton>
</rt:action>
```

The <descriptionStateButton> instruction

The instruction that changes the description of the state button has two attributes: description and tooltip. The first one is used to specify the label of the button (the text that will appear on the button). The second one specifies a tooltip text. The following example will show how to use the instruction to label the state button with the text "finish" and attach a tooltip text to the button explaining that the button can be used to finish the lesson:

```
<rt:action>
  <descriptionStateButton description="finish"
    tooltip="This button will finish the lesson"/>
</rt:action>
```

[78] actionset.StateButtonBehaviourGUIIQ
[79] gui.SimToolWin

The associated class of this instruction is the class `StateButtonDescGUIIQ`[80] which is derived from the state button base class (`StateButtonGUIIQ`).

The `<visibilityStateButton>` instruction

The instruction which can be used to change the visibility of the state button is called `<visibilityStateButton>`. Its boolean attribute `visibility` specifies whether the visibility shall be set to `true` or `false`. The instruction is implemented in the class `StateButtonVisibilityGUIIQ`[81] which is derived from `StateButtonGUIIQ` and has therefore access to the state button. The following example will show how the state button can be made visible:

```
<rt:action>
  <visibilityStateButton visibility="true"/>
</rt:action>
```

3.7 Performance issues

One of Java's major drawbacks is the possible low runtime performance of programmes written in Java.[82] This is a problem especially for applications and applets using the Swing API. The problem is based on the Java architecture. The Java Virtual Machine is an abstract machine which interprets Java byte code. As the execution speed of machine code depends on the hardware design of the host machine, the performance of Java programmes depends on the design of the Java Virtual Machine. Nowadays, there are good approaches like just-in-time compilers (JIT-compilers) which are able to speed up Java programmes like programmes written in C.[83] Nevertheless, Java programmes are usually supposed to run on any platform (= JVM), so the developer must take care of its performance.

As mentioned above, performance may be weak, especially, when using GUI components provided by Swing. SimTool is a programme with a large graph-

[80] `actionset.StateButtonDescGUIIQ`
[81] `actionset.StateButtonVisibilityGUIIQ`
[82] see [Ven99] p. 17
[83] see [Ven99] loc. cit.

ical user interface written using the Swing API. There are not many possibilities to speed up the GUI. Therefore, it is even more important to do a lot of performance tuning on other components of the system. Especially, the look-up of objects and its methods via the reflection mechanism (see Chapter 2.2) are quite time consuming.

Sun Microsystems, as the provider of the Java Development Kit, explicitly warn of the usage of that mechanism if others are usable as well:

> First, a note of caution. Don't use the reflection API when other tools more natural to the Java programming language would suffice.[84]

To avoid time consumption of the reflection mechanism, SimTool does a lot of configuration at start up (during the initialisation phase). Especially, the look-up of the objects and its methods is done in advance and the results are stored in look-up objects which are containers that hold information reflecting the system. To benefit from these look-up objects, a passive approach is taken. The look-up objects do not trigger any actions, they are only used to deliver meta information quickly. They are initialised at the startup of the system. There are two look-up objects involved, one can be used to look up the different objects and their methods (`ObjectLookup`) and the other one serves to look up registered exceptions (`ExceptionLookup`) and their exception types.

To invoke a script interface method, the system needs to get a method object and the object on which the invocation has to be performed. The information are stored in the map of `ObjectLookup` which is filled during the initialisation of the different objects. When a new object is created (for example a simulation component), it has to register on the map (see Chapter 4.3 for details). At the point of registration, the object is stored in the map together with a representation of all its script interface methods. In case of the destruction of an object, the entry will be deleted from the map. The erasing of a simulation component is an example of such a case. If the system invokes a script interface method, the object and the method will be looked up in the map. This safes a lot of time, because the look-up process of the methods and the object via reflection does not have to be performed again and again. Due to the static nature of the objects at runtime, no problem arises out of the obsolescence of saved values.

[84] see [Gre98]

The other look-up object (`ExceptionLookup`) is used to provide a flexible and quick access to the different exceptions. The object contains a map which stores the loaded exceptions. The map can be asked for exceptions that are based on conditions of a certain condition type connected with a specific triggering object. The storage of exceptions in the map is necessary to speed up the invocation process. This allows the system to check only exception that are based on conditions that belongs to a specific object (triggering object). It is especially not necessary to check all conditions of all exceptions at every point of time. However, reducing the amount of conditions that have to been checked via the specification of the triggering object will not help. In a complex lesson, there can be a huge number of exceptions loaded and SimTool does not limit the number. Therefore, condition types are used. Each condition type represents a point of time. At any point of time, only those conditions are verified which have been assigned to the associated condition type. For further information on condition types see Chapter 3.5.4.

Chapter 4

Aspects of Implementation

4.1 Example

To provide a better explanation of the internal processes, an example will be used throughout all parts of this chapter. Presume there is a single station with the ID “SingleStation1” and a second single station with the ID “SingleStation2”. Both are connected in a simulation network, as shown in Figure 4.1. Furthermore, an exception set is a loaded which contains an exception with the ID “Exception1”.

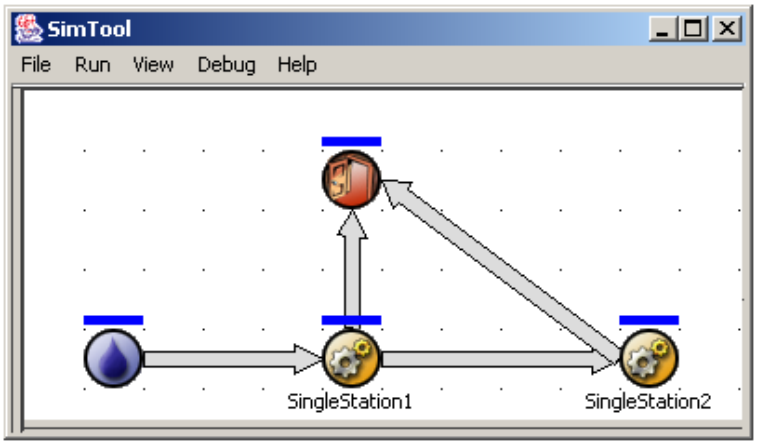

Figure 4.1: Simulation network used in the example

```
<es:action name="Exception1" maxCalls="1">
  <showDialog type="info"
              text="The throughput is greater than 10"/>
  <conditions connector="and">
    <condition srcId="SingleStation1"
               function="compProcTimeParams"
               classType="instance"
               condType="componentProperty">
      <param name="distClass" type="stdType">
        <value>distribution.DistributionConst</value>
        <type>java.lang.String</type>
      </param>
      <param name="params" type="vector">
        <value>5.0</value>
        <type>java.lang.Double</type>
      </params>
      <returnValue compareOperator="==" type="vector">
        <value>-1</value>
        <type>java.lang.Integer</type>
      </returnValue>
    </condition>
    <condition srcId="SingleStation1"
               function="getThroughPut">
               classType="instance"
               condType="componenStatistic">
    <returnValue compareOperator="&gt;">
      <value>10.0</value>
      <type>java.lang.Double</type>
    </returnValue>
    </condition>
  </conditions>
</es:action>
```

In the case that two conditions connected by `AND` are fulfilled, the exception will perform an instruction which will display a window which informs the user that "The throughput is greater than 10". The first condition will be satisfied if the distribution that describes the process time of the component with the ID "SingleStation1" is set to a constant value smaller than 5. The second condition verifies that the component "SingleStation2" has a throughput greater than 10. Both conditions are connected by the logical connector

AND. That means, the instruction will only be performed if both conditions are fulfilled.

The second condition of the exception will be verified every time the throughput changes. This would result in case the conditions are fulfilled in a continuous execution of the instruction. However, this behaviour is not desired, because a continuous popping up of a dialogue will annoy the user. To avoid this behaviour, the maxCalls attribute is set to 1 to ensure that the exception will not be raised more than once. After the exception has been raised, it will be deleted from the exception table. In this example, the object that will be checked by the invocation of one of its script interface methods (sc_compProcTimeParams, sc_getThroughPut) also triggers the condition test.

4.2 Initialisation

The Simtool[1] object is the main object of the simulation tool. This object does most of the initialisation and contains the entry point of the system. It enables the system to be started as a Java applet or Java application. A Java applet (subclass of Applet[2]) must implement the methods init, start and stop. The method init represents the entry point. In the case of an application the main method is the entry point.

The sequence diagram in Figure 4.2 will illustrate the whole initialisation process.

[1] gui.Simtool
[2] java.applet.Applet

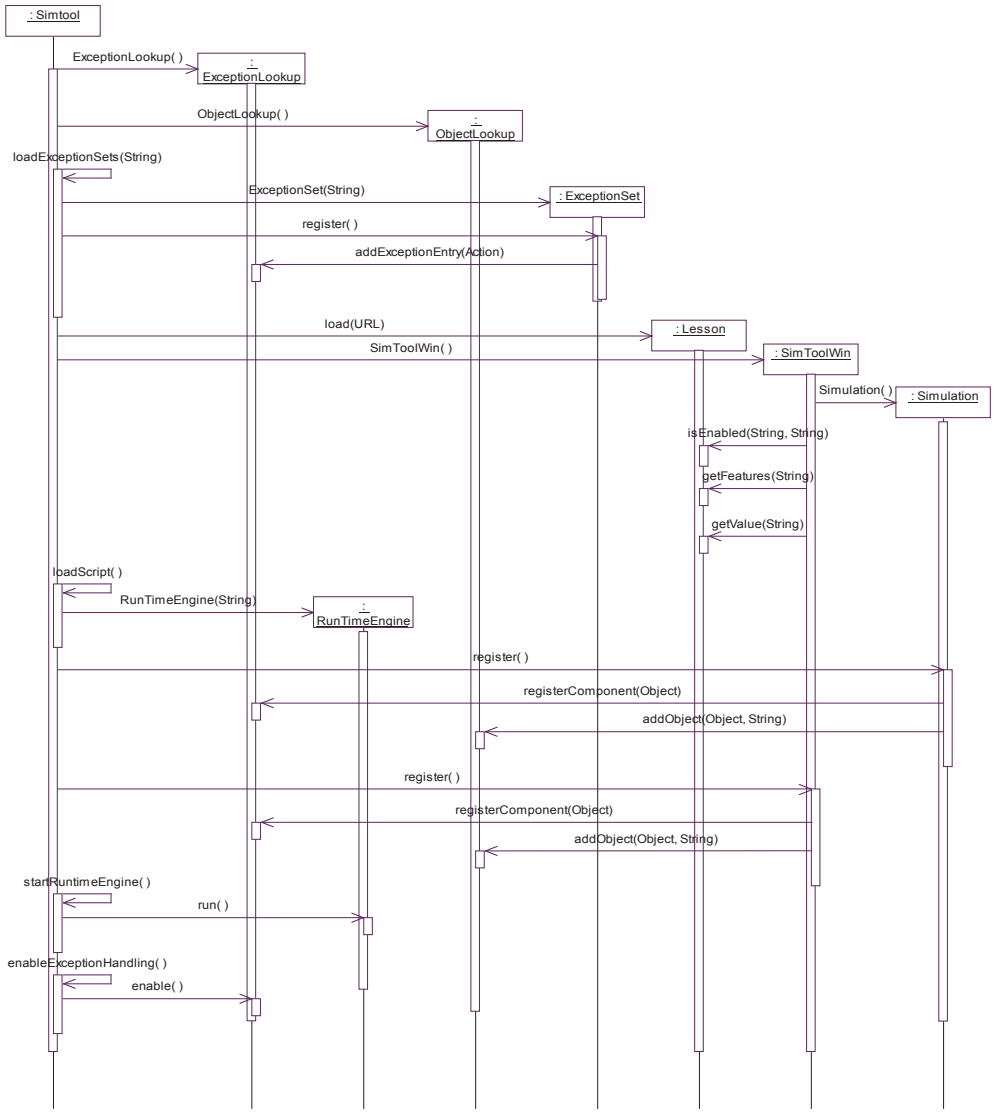

Figure 4.2: Initialisation process

The first step after starting the programme is the creation of the both look-up objects: `ExceptionLookup`[3] and `ObjectLookup`[4]. These objects have to be created during the initialisation phase of the system, because simulation and system components have to register on both look-up objects. These look-up

[3] `exceptions.ExceptionLookup`
[4] `exceptions.ObjectLookup`

objects are static[5] and ready to use after the initialisation process.

Basic exception sets will be loaded into `ExceptionSet`[6] objects.[7] These set objects register themselves via `register` on the exception look-up object (`ExceptionLookup`) during their initialisation phase (see Chapter 4.3).

Afterwards, the lesson file which specifies the basic settings of the system will be loaded. Basic settings are, for example the visibility of GUI elements and the visibility of simulation components and their properties. The lesson file can be accessed via a class called `Lesson`[8] which provides interfaces that can be used to ask for settings in the lesson file (see Chapter 3.2 for more information).

The main window (`SimToolWin`[9]) will be initialised next. An example of the initialisation performed by `SimToolWin` is the creation and positioning of GUI elements according to the values specified in the lesson file. Another important initialisation step is the creation of the simulation engine (`Simulation`[10]).

The `Simtool` object prepares the script engine by the initialisation of the `RunTimeEngine`[11] object. This object is static and will be constructed by parsing an XML file which contains the runtime script. The runtime engine object represents a script, which can be seen as a deserialisation of the different XML nodes in the runtime script. To achieve this, the script file will be parsed and `ExecutionBlock`[12] objects will be created. `ExecutionBlock` objects reflect blocks in the script which store any number of instructions. The script can have references to files that represent exception sets, which are internally stored in `ExceptionSet` objects (see concepts of execution block and exception sets in Chapters 3.4 and 3.5). An exception is called loaded if it is stored in the exception look-up map. Sets whose exceptions are loaded are stored in the `RunTimeEngine` object to provide the possibility of accessing them later when models are saved.

---

[5] In this context, static does not refer to the `static` keyword known in Java, it only indicates that there can only be one instance of each look-up objects.

[6] `exceptions.ExceptionSet`

[7] only possible if the system runs as an applet (applet parameters)

[8] `simulation.Lesson`

[9] `gui.SimToolWin`

[10] `simulation.Simulation`

[11] `runtime.RunTimeEngine`

[12] `runtime.ExecutionBlock`

Exceptions are implemented in `Action`[13] objects, which store an instruction (`InstructionQualifier`[14]) and conditions (`ConditionQualifiers` in a `ConditionContainer`). The instruction will be performed if the conditions are fulfilled. An object that is referred to as the triggering object in a condition of an exception will trigger the check of this condition. The object representations of these trigger objects are used as keys in the exception look-up map in order to provide a quick access to the exceptions and with them to the conditions that have to be checked.

Table 4.1 describes the exception look-up map after the initialisation phase of the example. The map holds the information that the object specified through its object representation (in this case the ID of the component) (key section) that the object triggers at least one of the conditions of the exceptions stored as values.

Key	**Value**
SingleStation1	<Exception1>
SingleStation2	<Exception1>

Table 4.1: Contents of the `ExceptionLookup` table after initialisation

Next, the simulation (`Simulation`) and the main window (`SimToolWin`) register themselves on the look-up objects, because they are important system components. An object which registers on the exception look-up object retrieves an enumeration of condition types referring to sets of conditions which need to be checked at a certain point of time during the runtime of the object (for further information see Chapter 4.5.1). Internally, these condition types will be determined by the table via an enumeration of the different conditions that store an object representation of the calling object.

However, at this point of time at the end of the basic initialisation process the look-up objects will not return any values, because their tables are still empty. The exceptions are not loaded yet. After the exceptions have been loaded, the registration of the simulation and main window object on the look-up objects has to be reapplied to return proper condition values.

[13] `actionset.Action`
[14] `actionset.InstructionQualifier`

The execution of the runtime script starts by the invocation of the `run` method of the `RunTimeEngine` object. Normally, the exception handling will be enabled in the script by a special instruction (`<enableExceptions>`). However, if basic exception sets are specified and a runtime script is not specified, the exception handling has to be enabled at the end of the initialisation phase for providing user interaction possibilities. The initialisation of other components of the programme will take place when they are created (for example, if a simulation component is added to the simulation model).

4.3 Registration of new objects

Objects can provide script interface methods. These methods are used in the runtime script or the files that describe the different exception sets. All objects providing script interface methods need to register at the `ObjectLookup`[15] and `ExceptionLookup`[16] objects. This registration process is diagrammed in Figures 4.3 and 4.4. Objects register themselves by calling `register` during their initialisation phase. A `deregister` method will be executed automatically before an object will be destroyed by the system. Both look-up objects provide methods which act in a supportive way for the (de)registration process. These methods are:

ExceptionLookup

1. `registerComponent(...)` (two implementations):
   This method exists in two different implementations: It takes either an `Object`[17] or a `BaseComponent`[18] as a parameter. The method with a `BaseComponent` as a parameter is used to register simulation components. The other implementation is used to register system components (for example `Simulation`[19] object).

[15] `exceptions.ObjectLookup`
[16] `exceptions.ExceptionLookup`
[17] `java.lang.Object`
[18] `simulation.BaseComponent`
[19] `simulation.Simulation`

ObjectLookup

1. `addObject(...)`:
   This method adds an object to the object look-up map. Parameters are the object itself and the ID of the object. The ID is used as a key in the map to ensure a quicker navigation through the different values.
2. `deleteObject(...)`:
   This method deletes the entry for the object specified by its ID.

Objects will register on `ExceptionLookup` to determine the condition types of the conditions that are relevant for them. A condition is relevant for an object if the object is the trigger of that condition. They store these condition types to determine the point of time when a condition check has to be raised. For more information on this topic see Chapter 4.3.1.

The developer who wants to add new simulation components does not have to care about the registration process because they register and deregister automatically.[20] The registration process is mainly done by the execution of one of the `registerComponent` implementations (on `ExceptionLookup`) and the execution of `addObject` (on `ObjectLookup`). The whole process ends with setting an internal flag of the component (`registered`) to `true`, to determine that the component has registered properly.

4.3.1 The determination of the condition types

A condition can be triggered at more than one point of time during the runtime of a component by specifying more than one condition type. The condition types are connected by `OR` and stored in a `long` value. This allows to question for a specific condition type by bitwise `AND`ing the long value (describing the condition types) with the desired type. The condition types determine the point in time at which the condition will be verified.

The following methods are useful for this process:

[20] This behaviour is implemented in `BaseComponent`

ExceptionLookup

1. `getCondTypes(...)`:
   This method is declared private. The parameter is the ID of the simulation component or in case of a system component its class name. The method iterates over all `Action`[21] objects (exceptions) which are relevant for the invoking object. In this context, a relevant exception is an exception that is stored under the key which represents this object (ID or class name). All conditions in the iteration will be checked whether their triggering object representation points to the triggering object that was passed to this method as a parameter. If the trigger object representation refers to the object, the condition types of that particular condition are added to the `long` value (consisting of `OR` connected condition types) which will be returned as the result of the method invocation.
2. `getbyFlag(...)`:
   Parameters are the ID or the class name of the object and the condition type. This method returns a vector of exceptions which belong to a specific condition type and an object. An `Action` object (exception) belongs to a specific condition type and an object if at least one condition in the exception contains the condition type and a representation of that object as the trigger. This method is used when an object verifies conditions of a specific type.

Condition Qualifier

1. `getCondType()`:
   This method returns the `long` value which represents the condition types of the condition.
2. `isCondType(...)`:
   This static method can be called to determine whether a specific condition type is set. In general, this will be used on the `long` value which represents the collection of condition types returned by the `getCondType` method of the `ConditionQualifier` or the `getCondTypes` method of `ExceptionLookup`. Objects which trigger an exception check will use this mechanism to determine the conditions that have to be verified.

[21] actionset.Action

3. `getConditionTypes`:
   Is a method which returns text that describes the different condition types.

The diagram in Figure 4.3 will show the steps that lead to a successful registration of a simulation component. The creation of system components will also be shown. In the following the diagram will be described in detail.

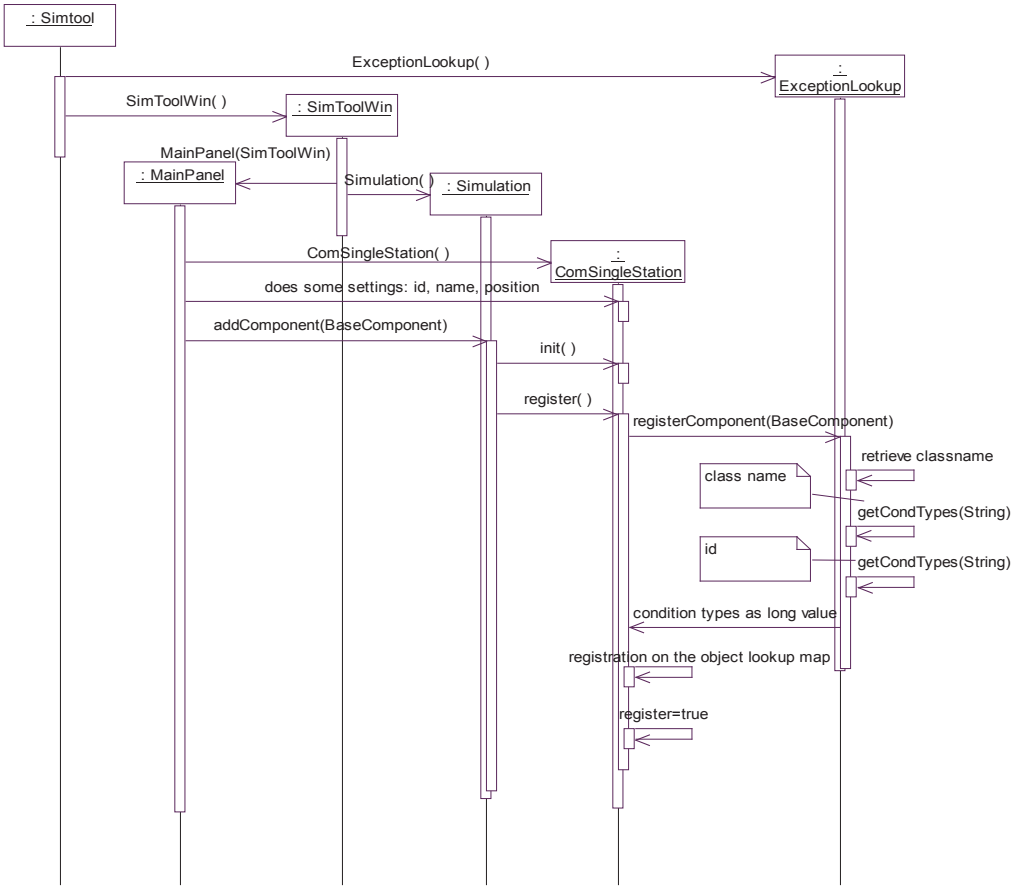

Figure 4.3: Registration of a new component on the exception table

As mentioned above, components need to register on the `ExceptionLookup` object to determine their relevant condition types. Therefore, they invoke the `registerComponent` method on the `ExceptionLookup` object. This method calls the `getCondTypes` method with the representation of the object (ID or class name). Note that only components derived from `BaseComponent` can be referenced by its ID and class name. All other objects (for example the `Simulation` object) can only be referred to by its class name. The `getCondTypes` method asks each `ConditionQualifier` stored in the exceptions for its condition type. The method will connect and return the condition types in a `long` value. The `ConditionQualifier`s are themselves stored under the key entry that represents the calling object of the method

In the case of the example, both components will call the `registerComponent` method on the `ExceptionLookup` object when they will be initialised. The components will pass themselves as parameters to the method. The method retrieves information about the components: the ID and the class name. A look-up of conditions that contain class and ID entries for the trigger object will be performed, because the components are simulation components based on `BaseComponent`. Therefore, `getCondTypes` will be invoked two times. Both executions of this method will return values which consists of different condition types (`OR` connected).

The internals of the `getCondTypes` method will be described using the example. Presume, the component with the ID "SingleStation1" wants to retrieve its relevant condition types. Therefore, it calls the `getCondTypes` method with its class name (with package information) as a parameter. In this context, relevant means that a condition needs to be checked at the point of time specified by the condition type.

First, the exception look-up map will be asked for a `Vector`[22] of all exceptions that are registered as values under the key "simulation.ComSingleStation" in the internal map of `ExceptionLookup` (class representation). The method will not return any value because the look-up map does not contain any exception stored under the key "simulation.ComSingleStation".

Next, the `getCondTypes` method will be invoked with the ID "SingleStation1" as the parameter. The exception look-up map will return a vector which contains the exception object with the name "exception1". Therefore, the triggering object representation of each condition stored in the exception

[22] `java.util.Vector`

will be compared to the ID of the object ("SingleStation1"). In the example the exception stores two conditions, but only in case of the first condition a matching occurs, because the trigger object equals the object with the ID "SingleStation1". As a consequence, the condition type of the first condition will be returned to the component (`COND_COMPONENT_PROPERTY`).

The whole process will be repeated for the other component (ID "SingleStation 2"). The result of the process is shown below:

Component 1

- *Look-up of the class representation "Simulation.ComSingleStation"*:
  - No exceptions registered
- *Look-up of the ID representation "SingleStation1"*:
  - `COND_COMPONENT_PROPERTY`

Component 2

- *Look-up of the class representation "Simulation.ComSingleStation"*:
  - No exceptions registered
- *Look-up of the ID representation "SingleStation2"*:
  - `COND_COMPONENT_STATISTIC`

4.3.2 The registration on the object table

The `ObjectLookup` object stores all objects used in the system. This look-up object is used for the quick and efficient look-up of objects and their script interface methods. Especially, this possibility of retrieving script interface methods helps keeping up with the aim of a good performance. This applies in particular to the invocation of the instructions and verification of conditions as they are done via the reflection mechanism which is known as to be quite a time consuming task (see Chapter 3.7). The object table is implemented as a nested map[23] and stores references to objects with all their

[23] `java.util.HashMap`

provided script interface methods as values in the map using the ID or class name of the objects as the keys.

An object starts the registration by calling `addObject`. There are a number of other methods described below, which are useful during the execution of the script and during the exception handling.

ObjectLookup

1. `addObject(...)`:
   This method is the key player for the whole registration process. An object that wants to provide script interface methods needs to register on the map by calling this method. The parameters are the representation of the object (ID or class name) and the object itself. This method stores the object representation as a key and the object itself with all its script methods as a value pair in a hash map. Additionally, it sets a flag that marks each script method to determine whether it is used in the currently loaded exception set.
2. `getObject(...)`:
   Returns an object stored in the map. The object is specified by its ID or class name. An ID is used in case of a simulation component, the class name in case of a system component.
3. `getMethod(...)`:
   This method returns a `Method`[24] object representing the script interface method which has to be invoked if a condition will be checked or an instruction will be performed. The parameters are the ID or class name of the object (the key in the hash map) that provides the script interface method and the name of this method.
4. `getMethodNames(...)`:
   Used to return the name of all script interface methods of an object (ID or class name as parameter).
5. `getMethods(...)`:
   This method returns a vector with all script interface methods of an object which is specified by its representation.
6. `isMarkedMethod(...)`:
   Can be used to determine if a script interface method is used in the

[24] `java.lang.reflect.Method`

loaded exception sets. Needs to be invoked with the representation of the object and the name of the method.

7. `getMarkedMethods(...)`:
   This method returns a vector that contains all script interface methods of an object (specified by its ID or class name) that are actually used in one of the loaded exception sets.

The sequence diagram in Figure 4.4 will illustrate the registration process on the object table. This process will be described in detail in the following.

In the case of the example, the following steps will be taken. At the moment when both single stations will be added to the main panel (main simulation area) the registration process will start by invoking the `register` method for registering the single stations on the `ExceptionLookup` object and the `ObjectLookup` object. The registration on the object look-up map is done by invoking `addObject` of `ObjectLookup`. As described above, this method takes the ID (in case of a simulation component) or the class name (in case of a system component) of the component and the component itself as parameters. In the example "SingleStation1" and "SingleStation2" are the IDs of the single stations.

The internal structure of the object can be described as a map which contain objects and another nested map. The value entry of the map stores an object called `ObjectValue`[25]. This object contains the actual object information. It stores two members: the object itself (in this case the component with the ID "SingleStation1" and "SingleStation2" respectively) and an object called `MethodValues`[26]. This object contains a map that stores another object called `MethodValue`[27] which serves as a container for information about the available script interface methods. The reflection mechanism is used during the registration process to retrieve a vector of methods from the class representation of the object. Script interface methods identify themselves by the script prefix[28]. Each `MethodValue` object reflects a script interface method with the information whether that method is used in one of the loaded exception sets.

[25] `exceptions.ObjectLookup.ObjectValue`

[26] `exceptions.MethodValues`

[27] `exceptions.MethodValues.MethodValue`

[28] The script prefix is specified in the static variable `gui.Simtool.SCRIPTPREFIX`. It is set to "sc_" and should not be changed.

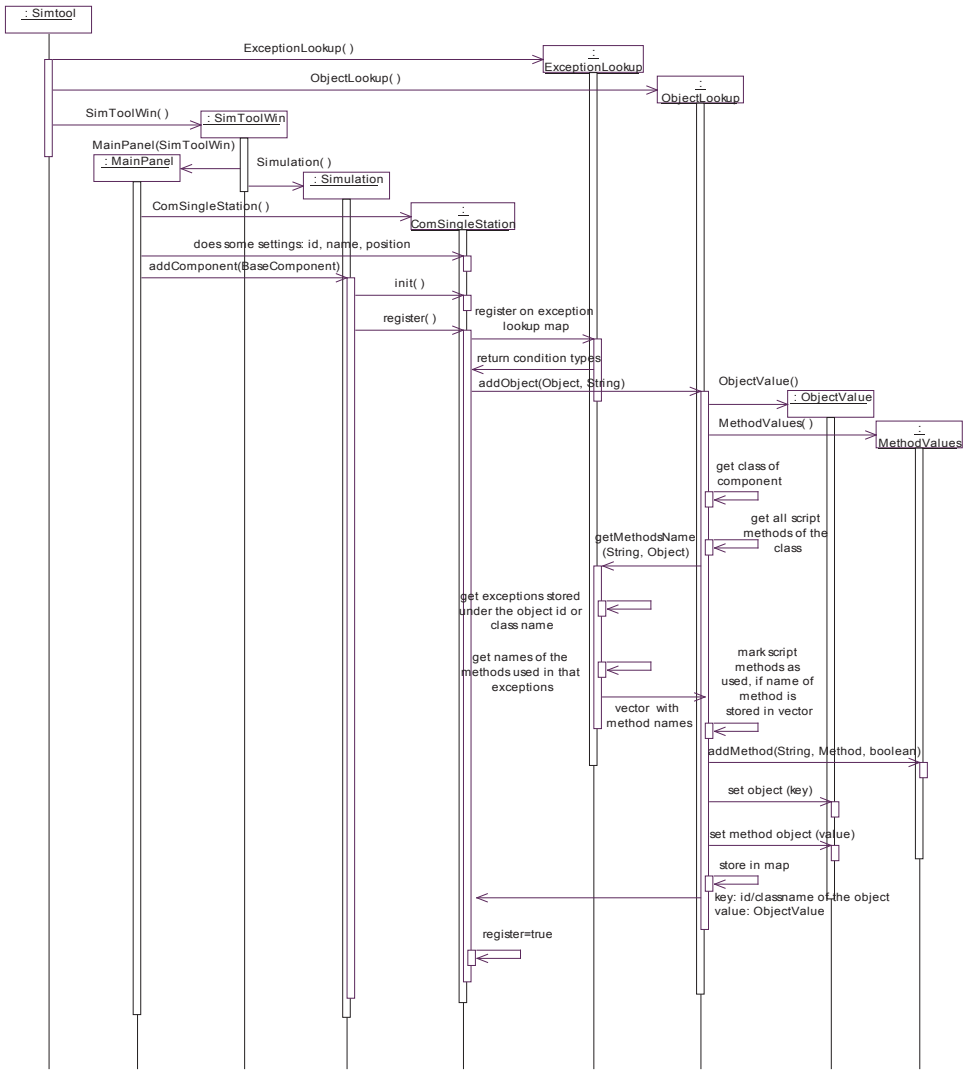

Figure 4.4: Registration of a new component on the object table

The name of the script interface method is used as the key in the map of `MethodValues` to refer to the `MethodValue` object. Table 4.2 shows the contents of the object look-up map after the registration process of the example.

Key	Object	methodname	method	ismarked
...				
SingleStation1	<SingleStation1>	...		
		sc_compProcTimeParams	<sc_compProcTimeParams>	true
		...		
...				
SingleStation2	<SingleStation2>	...		
		sc_getThroughPut	<sc_getThroughPut>	true
		...		
...				

Table 4.2: Object lookup table after initialisation

4.4 Performing script instructions

The `RunTimeEngine`[29] object performs script instructions which are stored in blocks (`ExecutionBlock`[30]). These blocks allow to structure the runtime script. `ExecutionBlock` objects can store one or more instructions (`Action`[31] without a condition part). An `Action` object represents a script instruction from the XML script, an `ExecutionBlock` represents a block defined there.

The block can be set to a delay mode. The value of the delay attribute in the script specifies the time that the system has to wait between the execution of the different instructions stored in the block. This mechanism makes it possible to watch a demonstration (for example, building a simulation network).

The `RunTimeEngine` can only be started if all initialisation as described in Chapter 4.2 is done. The essential part of the whole engine is the programme counter which points to the block that stores the current instruction to be executed. The execution of the runtime script will always start at the block labelled "main". Each `ExecutionBlock` object holds an own programme counter which points to the actual instruction that needs to be performed. The script engine supports jumps between blocks. Jumps are performed by setting the programme counter of the `RunTimeEngine` to the block that is a target of the jump instruction. This destination block needs to reset its own programme counter to its first instruction, because the execution starts at that instruction. Then the engine needs to be resumed. Following methods are useful for the control of the runtime scripts:

[29] `runtime.RunTimeEngine`
[30] `runtime.ExecutionBlock`
[31] `actionset.Action`

RunTimeEngine

1. `run()`:
   This method is the key player as it starts the whole engine and resets the programme counter to the start of the `ExecutionBlock` labelled "main".
2. `isRunning()`:
   This method returns a boolean variable that determines whether the script is running.
3. `pause()`:
   This method pauses the script.
4. `resume()`:
   This method resumes the script at the position specified by the programme counter.

ExecutionBlock

1. `isDelayed()`:
   This method determines whether the block is in delay mode.
2. `getDelayValue()`:
   This method returns the delay value for the block which has been specified in the script in seconds.
3. `reset()`:
   This method resets the programme counter of this block to the start position.

The process of performing a script instruction is illustrated in the sequence diagram in Figure 4.5. In the following this process will be described in detail.

The `RunTimeEngine` starts by calling the `run` method which sets the programme counter to the beginning of the execution block labelled "main". In addition to this, the programme counter of the main execution block needs to be reset. This is done by the execution of the `reset` method. After starting the engine a flag is set to determine that the script engine is in running mode. This flag is used to prevent the script engine from loading other instructions

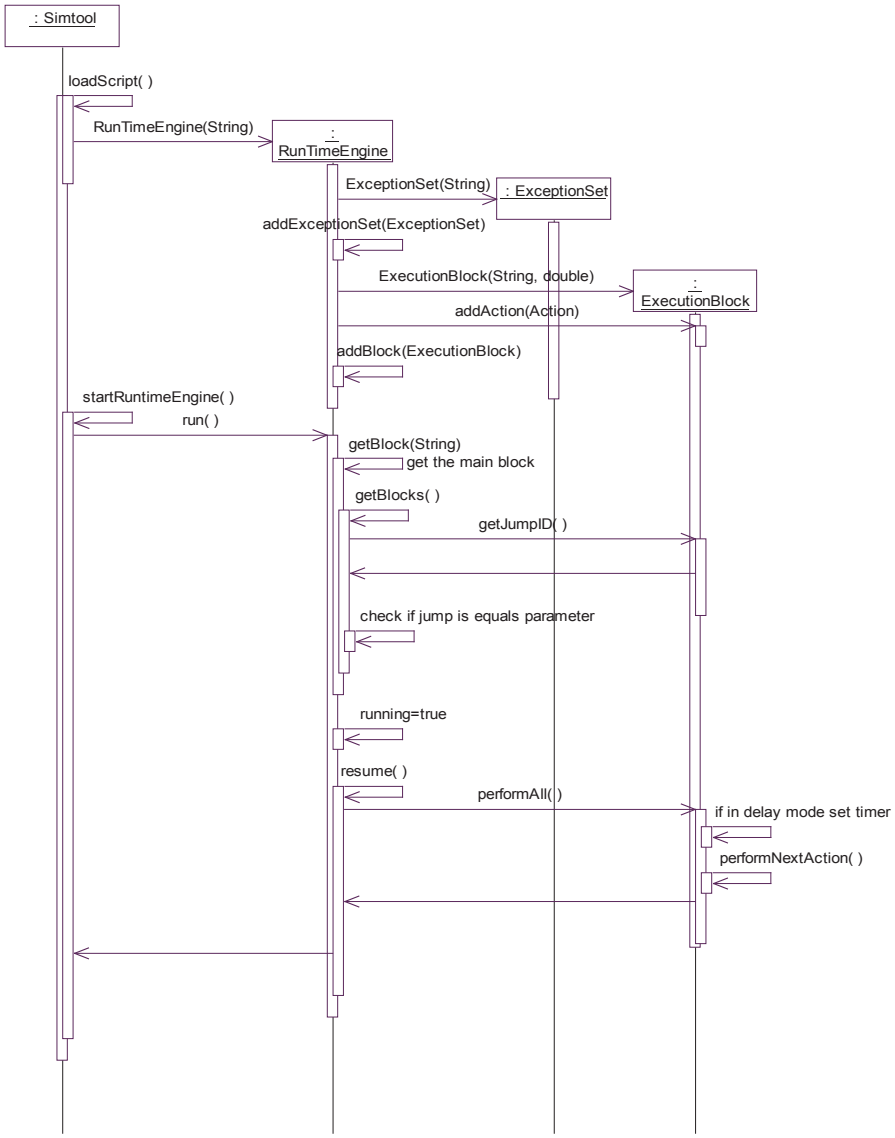

Figure 4.5: Initialise the script engine and perform script actions

during runtime. This behaviour reflects the concept of a static script which consists of a number of instructions that do not change. The script has to be defined in advance.

The script engine invokes `performAll` on the block specified by the programme counter (in this case the main block). If this block is in delay

mode a timer is set which executes the `performNextAction` method when the time span has passed. The time span is specified in the script by the `delay` attribute. After performing this method, the programme counter of the execution block will be forwarded to the next instruction in the block. If the delay is not set the instructions will be performed instantly. In both cases the instruction will be performed by executing the `checkandPerform` method.

The whole process will end after the execution of the last instruction in the script or if a break is raised (i.e. if an `EnableExceptionHandlingIQ`[32] instruction is performed). This instruction will enable the exception handling mechanism (described in detail in Chapter 4.5) and stop the execution of the script.

4.5 Raising exceptions

Several parts of the system are necessary to raise an exception. Following objects are involved:

- The object that triggers one of the conditions stored in the exception. This can be either a static object (like `Simulation`[33], also named a system component) or a non static object (simulation component). Non static objects (derived from `BaseComponent`[34]) are used by simulation model and can be referenced by the specification of their ID or the class name. If a class name representation is used all instances of the specified class can trigger the condition.
- The `ExceptionLookup`[35] object provides an efficient way to look up exceptions.
- The `Action`[36] object represents an exception.

[32] `actionset.EnableExceptionHandlingIQ`
[33] `simulation.Simulation`
[34] `simulation.BaseComponent`
[35] `exception.ExceptionLookup`
[36] `actionset.Action`

- The condition part of the exception stored in a `ConditionContainer`[37] which contains condition qualifiers (`ConditionQualifier`[38]). For further information see Chapter 3.5.2.
- The instruction of the exception is stored in the `InstructionQualifier`[39] object.

The process itself is divided into four smaller processes which will be described in detail in the following. Figure 4.6 will show the process in detail. In addition, Figure 4.7 displays the actual check of a condition with a class representation as an example of the invocation process.

4.5.1 Look up an exception

To check whether an exception must be raised, it is necessary to find the relevant exception. An exception is relevant for an object, if the object is registered as the triggering component of at least one of the conditions stored in the exception. Due to performance issues, not all conditions of all exceptions should be checked at once. Therefore, condition types are used to restrict the number of conditions that have to be checked during the condition look-up process. Every condition contains one or more condition types which specify when the condition has to be checked during the life-time of the object that triggers the condition test. An exception can be based on more than one condition which leads to the occurrence of more than one condition type. These different types are stored `OR` connected in a `long` value. See Chapter 4.3.1 for information regarding the implementation of condition types.

Objects that trigger conditions need to be registered on the object table and on the exception table. Furthermore, they need to be specified in the condition as the triggering object. This trigger object stores the value containing the condition type information. The different condition types do not only serve to appoint the point of time of triggering, they also help to isolate a subset of exceptions. This is done via the `lookupCond` method of `ExceptionLookup`.

[37] `actionset.ConditionContainer`
[38] `actionset.ConditionQualifier`
[39] `actionset.InstructionQualifier`

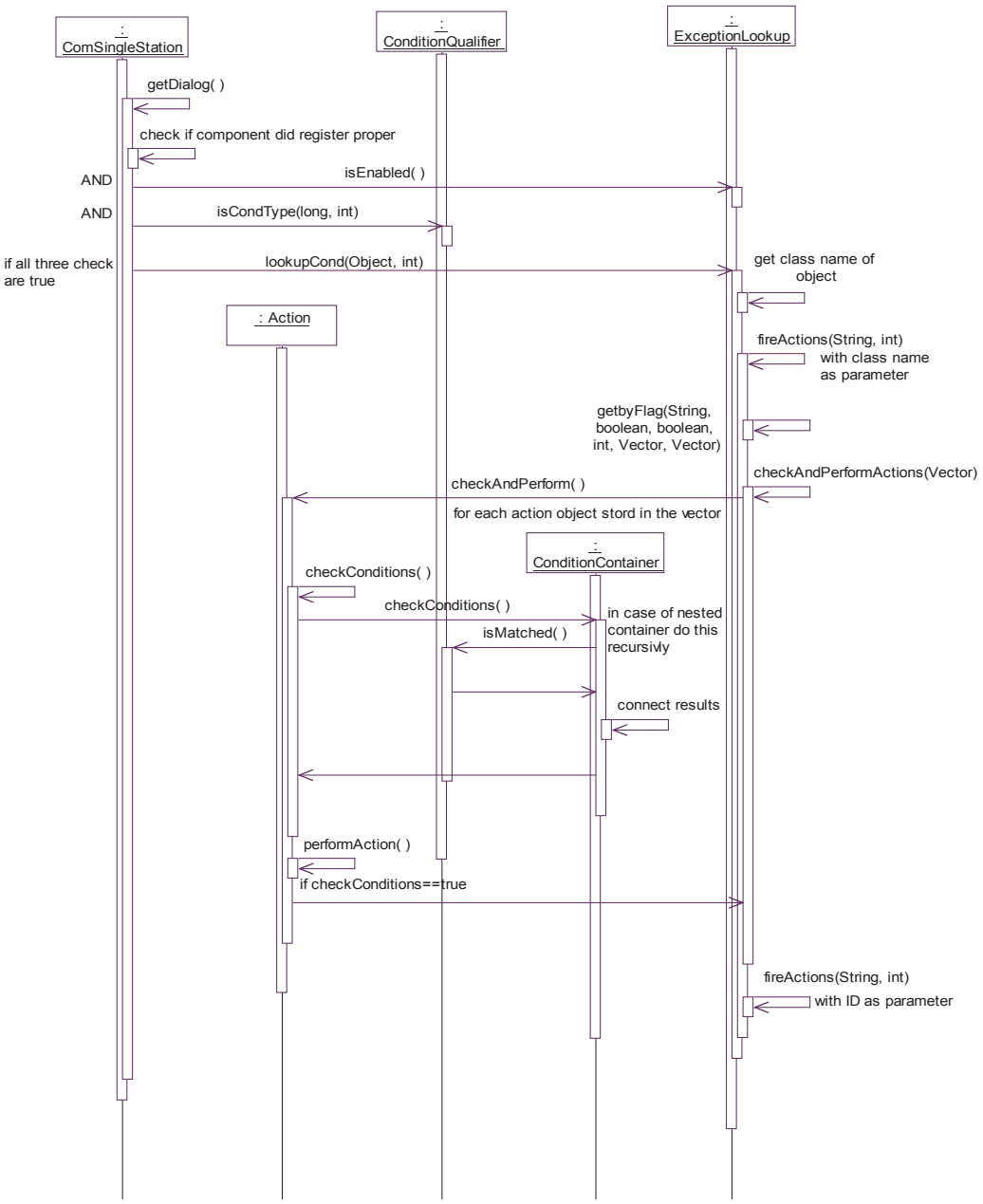

Figure 4.6: The entire invocation process

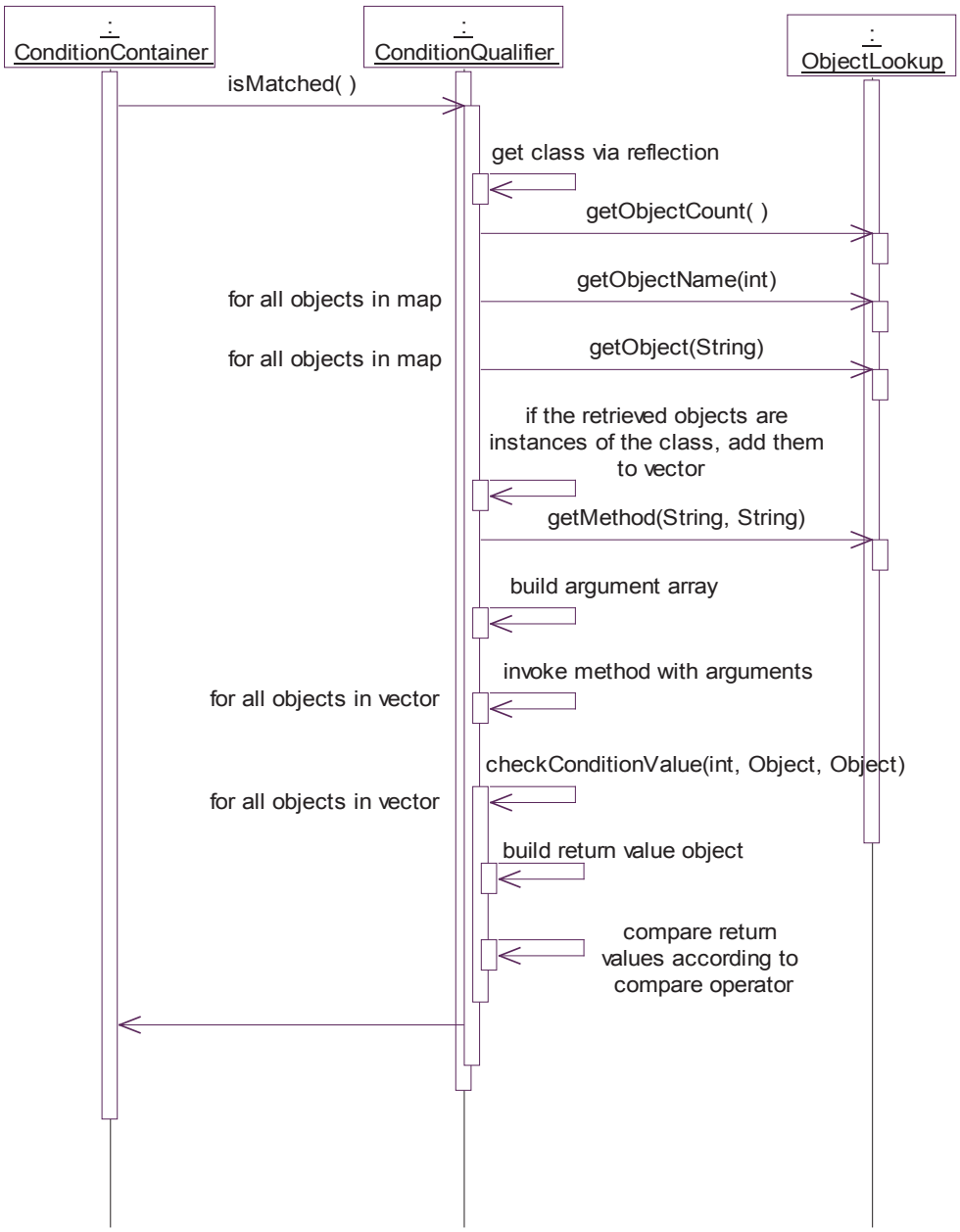

Figure 4.7: Condition test

The following example explains the process in detail.

Presume the user adds the second single station to the main simulation area, it will automatically get the ID "SingleStation2". The object (of type `simulation.ComSingleStation`) will register as described in Chapter 4.3. The result of the registration is an entry in the object look-up table as well as the condition types of the conditions the object is involved with (stored as a single `long` value). Then the actual life-time (spoken in terms of the simulation) of the component begins. The component asks the `ExceptionLookup` object for all exceptions which contain at least one condition that is connected to the specified condition type. But an exception is not only connected with the component via the condition types. Furthermore, the condition needs to have a matching reference to the triggering object stored as a representation. After the successful registration, the component with the ID "SingleStation2" has the condition type `COND_COMPONENT_STATISTIC`. The condition type `COND_COMPONENT_STATISTIC` represents a condition that works on statistical values. The point of time to check the condition is the `paint` method of the component. This leads to a permanent check during the life-time of the component which is critical to the performance. The following code was taken from the `paint` method of a component.

```
if (registered && statistic!=null) {
   if (ConditionQualifier.isCondType(
       condValues,
       ConditionQualifier.COND_COMPONENT_STATISTIC)
       && Simtool.getExceptionTable().isEnabled()) {
       Simtool.getExceptionTable().lookupCond(getComponent(),
           ConditionQualifier.COND_COMPONENT_STATISTIC);
   }
}
```

The check for fulfilled conditions (in the example above) will only start if the component has registered properly (flag `registered`) and the statistic is enabled (variable `statistic`). Presume both conditions in our example are fulfilled, the following steps will be taken by the component: The static method `isCondType` will be invoked to compare the `OR` connected condition types (acquired during the registration on the exception look-up map) to the condition type, representing this actual point of time. In case of the example, the condition value contains the condition type `COND_COMPONENT_STATISTIC`.

Next, it will be checked if the exception handling is enabled. The exception handling will be enabled by the script (`enableExceptionHandlingIQ` instruction). If the exception handling is enabled the `ExceptionLookup` object will be used to perform the next steps. The `lookupCond` method will be called with the component itself and the condition type as a parameter. As to the nature of a single station as a simulation component, this method retrieves the class name (with package information) and the ID of the component as a representation of the object.

As written above, components work on both representations (class and ID) whereas static objects (system components) work only on class representation. In the example, the set of exceptions involved with a class representation ("simulation.ComSingleStation") is null, because an exception containing conditions that are based on a class representation "simulation.ComSingleStation" does not exist.

Afterwards relevant exceptions will be looked for by ID. This time, the `getbyFlag` method of `ExceptionLookup` will return a vector that contains exceptions. Parameters of the method are the ID ("SingleStation1") and the condition type (`COND_COMPONENT_STATISTIC`). Note that there is also another entry point to that mechanism used by the `ExceptionsCheckIQ`[40] and `StateButtonBehaviorGUIIQ`[41] instruction (see Chapter 4.6 for further information). In case of the example, the return vector contains the exception with the ID "Exception1". If the component with the ID "SingleStation2" starts this process the same value will be returned.

The process ends here. The result of the process is a vector of exceptions which need to be checked for fulfilled conditions.

4.5.2 Determination of the different conditions

A `ConditionQualifier` object reflects a condition. Each exception can hold more than one condition. The conditions are stored in an object which acts as a container for conditions (`ConditionContainer`). The condition container holds different conditions as well as other condition container objects and contains an additional flag that specifies which logical connector is to be

[40] `actionset.ExceptionsCheckIQ`

[41] `actionset.StateButtonBehaviorGUIQ`

applied between the different conditions residing in that particular container. If the connector is not specified an `AND` connector will be chosen by default. The recursive storage of different condition containers allows the usage of different connectors. Following methods are provided to assist this process:

Condition Container

1. `addCondition(...)`:
   This method is provided in different versions. Parameters are the condition or the condition with the logical connector respectively.

2. `addContainer(...)`:
   This method adds a condition container. This method exists in two versions. The first version takes the condition container as a parameter. The other version has the container and the logical connector as parameters.

3. `getConditionFlags()`:
   This method returns a `long` value which describes the different condition types. All conditions stored in the container or sub containers will be iterated in order to obtain this value.

4. `getConditionCount()`:
   This method counts the number of conditions which are stored in and below the actual container and returns it.

5. `deleteCondition(...)`:
   This method looks for a condition in and below the actual container. The condition is specified as a parameter. If the condition is found it will be deleted.

An exception will be executed by invoking the `checkandPerform` method on the `Action` object that represents the exception. This method compares the specified return value to the value returned by the script interface method of the condition. Therefore, it invokes `checkCondition` which returns a `boolean` value determining whether all of the conditions connected with the logical connectors are fulfilled. The method asks the topmost condition container for the state of all conditions stored in that container or one of its sub containers recursively. The actual check of the different conditions is explained in Chapter 4.5.3.

There are two different conditions in the example. One condition acts on the script interface method `sc_compProcTimeParams`, the other one on the method `sc_getThroughPut`. As both conditions belong to the same exception and are connected by `AND` both have to be checked. The results of this check will be connected by the `AND` connector.

4.5.3 Perform a condition test

The `ConditionQualifier` object which represents a condition contains the following information:

- `strObject`:
  This attribute is a representation of the object that contains the associated script interface method.
- `isClass`:
  This attribute is a flag that indicates if the entry stored in `strObject` has to be interpreted as a class name or a component ID.
- `strTriggerObject`:
  This attribute is a representation of the object that triggers the condition (ID or class name).
- `isTriggerClass`:
  This attribute is a flag that indicates the type of the trigger object representation.
- `strInterface`:
  This attribute is the name of the script interface method.
- `params`:
  This attribute contains the parameters that are used to invoke the script interface method.
- `paramTypes`:
  This attribute contains the types of the parameters.
- `rValue`:
  This attribute is a value that will be compared to the returned value of the script interface method.

- `rvalType`:
  This attribute is the type of the return value.

- `rvalOperator`:
  This attribute is the compare operator ($=$, $\neq$, $<$, $\leq$, $\geq$, $>$).

A condition will be fulfilled if the compare operation (specified by the `rvalOperator` attribute) returns `true` when comparing the value stored in the qualifier to the return value of the invoked script interface method. Following methods are involved in the process:

ConditionQualifier

1. `isMatched()`:
   This method performs the invocation process of the script interface method. The `ObjectLookup` object is used to provide an efficient way to retrieve information about the objects. If the object does not deliver the needed information, the reflection mechanism is used to retrieve them (i.e. when looking for classes instead of instances). Furthermore, it starts the comparison of the return value specified in the script with the return value of the script interface method according to the compare operator. Therefore, it invokes the `checkConditionValue` method and returns the result of it.

2. `checkConditionValue(...)`:
   This method will be invoked from within `isMatched`. It returns a `boolean` value which indicates whether the return values matched according to the specified compare operator. The parameters of the method are the compare operator, the return value defined in the script and the actual return value returned by the invoked script interface method. Therefore, it is necessary to create an object of the type that is determined by the variable `rvalType` that represents the return value. To create this object, the constructor of the class specified in `rvalType` which takes a string argument is searched and invoked. The parameter for that invocation is the string value which represents the return value in the script. This mechanism restricts to data types which can be constructed from

strings. Return values of special, not `String`[42] conformed types have to be implemented separately.

For invoking a method of an object a (`Class`[43]) object is needed. Then a (`Method`[44]) object that reflects the script interface method is needed. After that, an instance of the class is needed on which to invoke the method. The last step is the invocation of the method with the specified parameters. This process will be described below.

The whole process begins with the determination of the type of the object that provides the script interface method for the condition. The type can be accessed through the `isClassType` method which returns a flag that specifies whether a class name or an ID is used at the object representation. The object representation is stored in the condition and can be accessed by the invocation of `getObject`. Both representation modes (ID or class name) are handled differently.

Class representation

In this case, the class will be found by invoking the `Class.forName` method with the class name stored in the condition.

```
cl=Class.forName(ref);
```

The `ref` variable contains the name of the class with package information which will be returned by `getObject`. An instance of the retrieved class is needed to invoke the script method. Due to the nature of the class representation, more than one object can be found.

For example, if the class representation is "simulation.ComSingleStation" all objects that are instances of the class are involved. In our example, the components with the ID "SingleStation1" and "SingleStation2" would be involved. For retrieving all instances, the map of objects stored in

[42] `java.lang.String`
[43] `java.lang.Class`
[44] `java.lang.reflect.Method`

`ObjectLookup` will be iterated. Each object will be compared to the class representation. If the object is an instance of that class, it will be stored in a vector.

Due to performance issues, the look-up of the script interface methods will be done through a method provided by `ObjectLookup` (`getMethod`). This method takes the name of the object and the name of the method as parameters and returns a `Method` object (of the script interface method). If this invocation fails, the backup-line will be taken and the reflection method `getMethod` will be invoked. Following code excerpt will show the important steps.

```
if (isClassType()) {
   cl=Class.forName(ref);
   ...
   String name=null;
   String objectname=null;
   Vector compObjVec=new Vector();

   // look-up objects
   for (int i=0; i<oiLookup.getObjectCount();i++) {
        name=oiLookup.getObjectName(i);

        // stores all objects in the vector
        Object compObj=oiLookup.getObject(name);
        if (cl.isInstance(compObj)) {
            compObjVec.add(compObj);
            objectname=name;
        }
    }
    obj=compObjVec;
    ...
    // try to retrieve the method representation
    m=oiLookup.getMethod(objectname,interf);
    // if this did not work try backup-line
    if (m==null) {
        m=cl.getMethod(interf, (Class[])getParamTypes());
    }
    ...
}
```

Instance representation

An instance representation implicates that the object must have registered on the map of `ObjectLookup`. In contrast to the class representation, only one object can be involved with that condition. The object will be retrieved via the invocation of the `getObject` method of `ObjectLookup`. The parameter of this method is the ID of the object as stored in the condition (accessible by the `getObject` method of the condition). After that, the method will be retrieved as already described in case of the class representation. The following code excerpt shows the procedure in detail.

```
// get the object
obj=oiLookup.getObject(ref);
if (obj!=null) {
  ...
  // look-up methods
  m=oiLookup.getMethod(ref,interf);
  ...
}
```

Both cases

Both cases merge after the `Method` object has been found. Script interface methods can have parameters. A parameter can be described as a pair of two values: the value of the parameter and its class type. The class type is necessary to cast the value of the parameter (coded as a text in the XML script) to the correct parameter object as needed by the script interface method.

With the information of the type, the corresponding class of the parameter will be found. This class is used to create an object that reflects the parameter. The instantiation of the object works similar to the one used for the creation of the return values. The same restrictions apply.

However, other parameter types can be used as well, but they need to be implemented. A special parameter type already implemented is `Vector`. The contents of the vector used as the parameter needs to be casted. Therefore, the variable that stores the type of the parameter (`rvalType`) does not store

the class `Vector` as it would usually do; instead it stores a vector which contains the class types of the different values that reside in the vector used as the parameter. With this information the values in the parameter vector will be casted. Another special parameter already implemented is an array of class entries (`java.lang.Class[]`). The following piece of code illustrates the parameter casting process.

```
...
if (params.get(u) instanceof java.lang.Class[]) {
    // parameter is of type Class Array
   args[u]=params.get(u);
}
else if (this.getParamTypes()[u] instanceof
         java.util.Vector)    {
    // parameter is of type vector
    ...
    // cast each parameter in vector
    for (int r=0; r<paramsVec.size();r++) {
       ...
       Class c=Class.forName("java.lang.String");
       Class[] argTypes2={c};
       Object[] argValues2={paramsVec..get(r)};
       Class castClass=(Class)classVector.get(r);
       Object storeObj=castClass.getConstructor(argTypes2).
                       newInstance(argValues2);
       paramVec.add(storeObj);
    }
    ...
    args[u] = paramVec;
}
else {
    // standard types
    Class[] argTypes={String.class};
    Object[] argValues = {params.get(u)};
    args[u] = ((Class)this.getParamTypes()[u]).
              getConstructor(argTypes).
              newInstance(argValues);
}
```

The parameters are stored in the `args` array. The invocation of the method is done by the execution of the reflection method `invoke`. This method takes the object and the array of parameters (`java.lang.Object[]`) as arguments. It returns the value returned by the executed script interface method. In case of a class representation this process has to be performed on each instances of the class stored in the vector (described above). This process does not consider the order of the objects. A condition working on a class representation will be satisfied if at least one of the existing instances fulfil it.

The `checkConditionValue` method of the `ConditionQualifier` compares the return value of the compare script interface method with the return value specified in the script. A compare operator (`rvalOperator`) is used for the comparison. The same problem as with the parameters occurs here: The return values that are specified in the script must be casted. This is done in the same way as with parameters (see above). This method returns a `boolean` value whether the values matched.

4.5.4 Execute the instruction

Performing an instruction works in the same way as performing a condition test. The difference is that there is no need for a return value, so `checkConditionValue` will not be invoked. As in the case of the condition handling, a script interface method will be called. However, this method does not check values, it performs instructions instead.

4.6 Adding a new instruction

This section explains the steps that are necessary to create a new instruction. This task might be necessary, because not all possible instructions are already implemented. The example used in this section reflects a real implementation done due to the demand of a greater flexibility in the interaction and communication process between the user and the system. The new instruction adds huge benefits to the exception and script handling system.

4.6.1 Demands

Interaction that the system provides should always be issues-related. The amount of feedback given by the system should be measured on the actual knowledge of the learner. Therefore, learners should be classified according to their knowledge level into different groups (novice, professional, etc.). To implement such different levels the teacher has to create a script file or exception set files which reflect the different knowledge levels. An example of such a scenario are a script and exception set files that are used during the first term of the semester and will be exchanged with files that contain less restrictive settings in the second term. To do this classification automatically, a mechanism is needed that will determines if the users reach a new knowledge level. A possible way to implement such a behaviour could be the creation of different exception sets. An example is one set defined for the novice and one for the professional.

The new instruction is able to address exceptions in a specific exception set. See Chapter 5 for an implementation of a lesson which uses automatic detection of different knowledge levels. The new instruction described in this section will add a new mechanism to the system for referring to exceptions. However, the normal way of qualifying conditions in exceptions via its triggering object should be used in most situations.

Presume, the user changes the property settings of a single station named "Machine1" which is related to a condition that will check whether the new settings are correct. If this condition is fulfilled an instruction will be raised which jumps back to the runtime script where a set of different instructions will be performed. An instruction that displays text that congratulates the user on the correct handling of the situation is an example. After that, the new instruction could be performed to check if components of one type (for example, all source components) have proper settings. All exceptions referring to that type should be defined in the same exception set file. The system can react on different knowledge levels by referring to sets that contain exceptions defined on that particular knowledge level (for example one set for beginners and another for professionals).

Furthermore, the new instruction can be used to implement nested condition tests. That means, an exception performs the new instruction which starts a condition test on another exception or set of exceptions if the conditions of the first exception are fulfilled. This instruction can be used in the runtime

script to trigger condition tests. A branching in the script according to different settings of the system can be achieved as well.

4.6.2 References of exceptions

An exception can be referred to by the condition object qualifiers, its name and the name of the exception set that stores it (see Figure 4.8 for details).

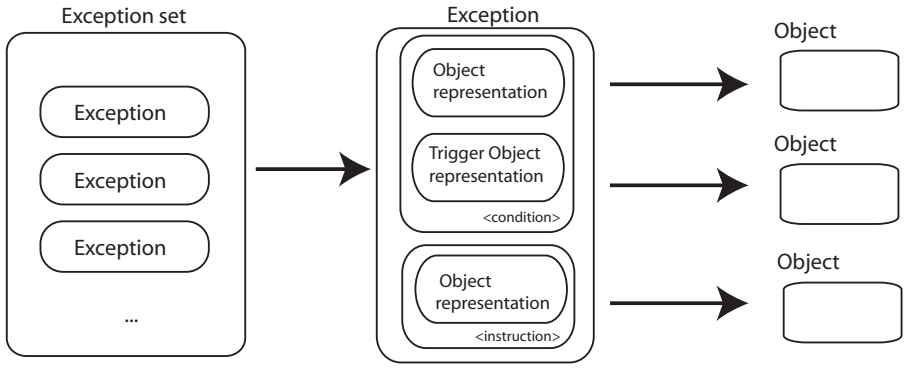

Figure 4.8: Referencing an exception

Another way of qualifying an exception could be the use of the object representation used in the instruction qualifier (instruction of the exception). However, the new instruction works by the invocation of condition tests. If the test is successful, it will result in the execution of the instruction of the exception. The instruction part of an exception is irrelevant for that mechanism. Therefore, it is not necessary and does not make sense to implement a referencing mechanism that is based on the instruction part.

The new instruction is able to work on more than one exception. Each of the qualification methods (exception set, exception name, object qualifications) can be used on its own or combined. If one of these methods is not used in the XML script, a restriction according to that method does not apply.

Exceptions can be referred to by the specification of the name of the exception set in which the exceptions are defined. More than one set can be specified. The following runtime script excerpt shows the usage of the new instruction (`<exceptionCheck>`) to perform a look-up of exceptions which are stored in exception sets named “ExceptionSet1” and “ExceptionSet2”.

```
<rt:action>
  <exceptionCheck>
   <excepSet>ExceptionSet1</excepSet>
   <excepSet>ExceptionSet2</excepSet>
  </exceptionCheck>
</rt:action>
```

In this context, the term "look-up of an exception" refers to a check of the condition parts (`ConditionQualifier`) of the exception and if they are fulfilled to an execution of the instruction (`InstructionQualifier`). Note that all qualification methods can be used more than once. All methods are connected with a logical `OR`. The example above will look up exceptions that reside in the set "ExceptionSet1" or "ExceptionSet2".

An exception can also be found by its name. The specification of a name results always in one exception because the name of the exception must be unique. It is possible to look up multiple exceptions by the specification of more than one exception name. The following example is a script excerpt that can be used to look up exceptions named "Exception1" and "Exception2".

```
<rt:action>
  <exceptionCheck>
    <excep>Exception1</excep>
    <excep>Exception2</excep>
  </exceptionCheck>
</rt:action>
```

Another possibility to specify an exception is the use of the representation of the objects involved with the condition. Two object representations are connected with the condition parts of the exception. One represents the object that triggers the condition test and the other one represents the object that contains the script interface method. Each of these representations can be used separately to refer to an exception. The kind of object representation is stored in the `mode` attribute of the qualifier tag.

Following modes are supported.

- `normal`:
  This mode refers to a representation that is used to get the objects that contains the script method.

- `trigger`:
  This mode refers to a representation that is used to retrieve the triggering object of a condition.

The type of the object representation has to be specified at the `type` attribute. The two known types (`instance` and `class`) are used in the same way as in the normal representation process, already known from the specification of a condition (see Chapter 4.5.1 for further information). The type `instance` refers to a representation used to look up a specific instance of a class. This type is used to look up all exceptions that are connected with exactly one object. In this context connected means that a representation of the object is stored in at least one of the conditions of an exception. Which of both object representations is used to look up exceptions depends on the setting of the `mode` attribute. The term "class" differs from the `class` flag that is used in the normal definition of a condition where all objects that are instances of the class are referred to by the flag. If a condition refers to a class, all instances of that class will also be related to this condition. For further information see Chapter 3.5.6.

The normal object representation is used in a conditions and an in instructions and refers to an object which contains the script interface method. The new instruction specifies the object used in the condition part of an exception to refer to the exception.

The `class` representation that is used in instructions that handle exceptions (`ExceptionsIQ`[45] and subclasses) refers to exceptions that are based on at least one condition part which is defined on that class. This does not include exceptions with condition parts defined on objects even if this objects are instances of the class. Figure 4.9 will illustrate the process.

However, these exceptions are included if the `classinclusive` representation type is used. It gets the exceptions defined for the class and the exceptions

[45] `actionset.ExceptionsIQ`

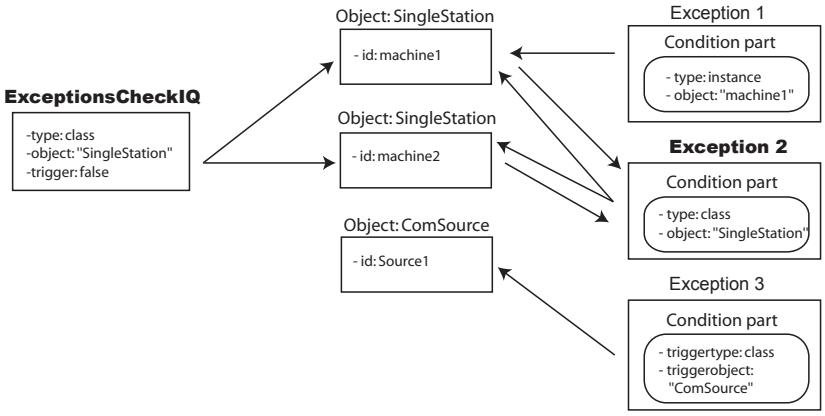

Figure 4.9: Example with mode="normal", type="class"

for all objects which are instances of the class. Figure 4.10 will illustrate the process.

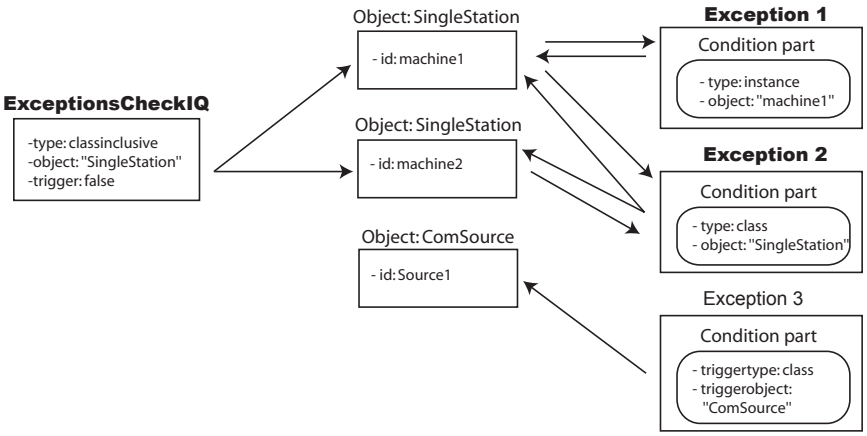

Figure 4.10: Example with mode="normal", type="classinclusive"

Figure 4.11 will show the result of a class/trigger representation.

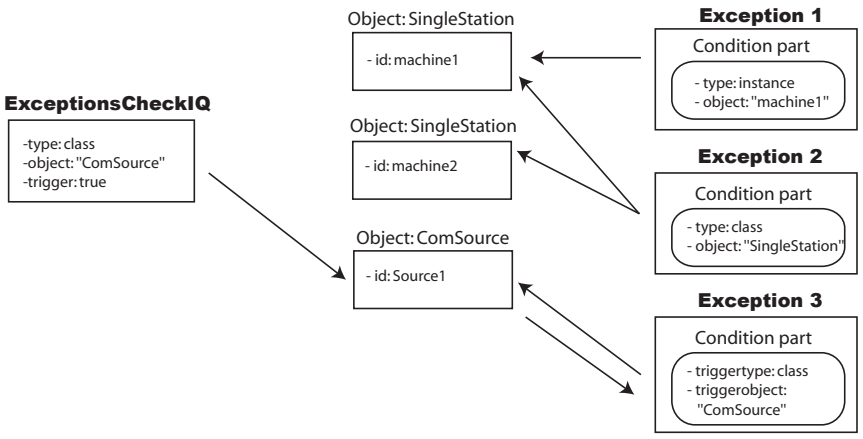

Figure 4.11: Example with mode="normal", type="class"

A better understanding of the different object referencing types requires background knowledge of the underlying mechanisms and structures. Figure 4.12 shows the involved classes with its main methods, involved in both representation processes (that one of the new instruction, and the normal one). Both kind of representation mechanisms are differently highlighted in the figure.

In the example (Figure 4.9), the new instruction asks the exception look-up object for exceptions that are relevant for a class representation with the class name "SingleStation". In the example "Exception2" would returned even if "Exception1" is also related to an instance of the class `simulation.ComSingleStation`. The exception with the ID "Exception1" will be returned as well by using the `classinclusive` mode instead of the `class` mode. Figure 4.10 shows this case.

An exception will only be found if at least one of the conditions have the new script condition type (`COND_SCRIPT = 256`). If the different referring mechanisms are used in a combined way, the restriction order is irrelevant, because it does not affect the result.

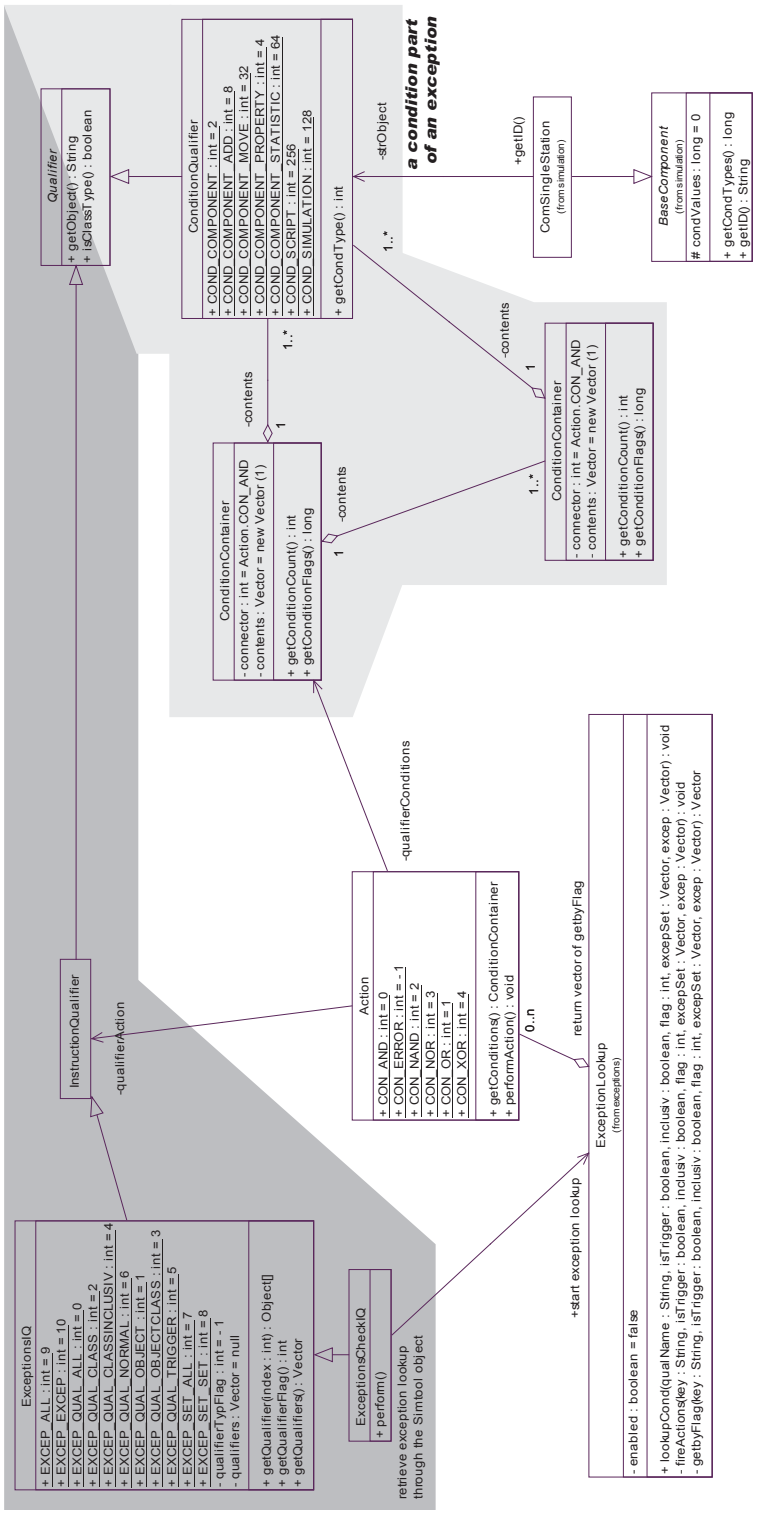

Figure 4.12: New instruction: `ExceptionsCheckIQ`

The excerpt of the runtime script shown below demonstrates the usage of two different representation mechanisms. The first one reflects objects of the type `simulation.ComSingleStation`. The other one represents a triggering object by its ID "MySource".

```
<rt:action>
  <exceptionCheck>
    <qualifier mode="normal" type="class">
       simulation.ComSingleStation
    </qualifier>
    <qualifier mode="trigger" type="object">
       MySource
    </qualifier>
  </exceptionCheck>
</rt:action>
```

4.6.3 Implementation issues

The introduction of a new condition type is necessary for enabling the exception to be found by the new instruction. This condition type called `COND_SCRIPT` acts as an indicator on a condition to indicate that the condition is relevant for script triggered exception look-ups (condition checks). The new instruction is implemented in the object `ExceptionsCheckIQ`[46]. This object is derived from the base class (`ExceptionsIQ`), which handles exceptions and is derived from the instruction qualifier base class (`InstructionQualifier`). `ExceptionsIQ` implements the different exception referring mechanisms. It defines a number of static flags which are described below.

1. *Qualifier specification types*:
    - `EXCEP_QUAL_ALL`: restrictions according to an object representation do not apply
    - `EXCEP_QUAL_CLASS`: class type
    - `EXCEP_QUAL_CLASSINCLUSIVE`: class inclusive type
    - `EXCEP_QUAL_OBJECT`: instance type

[46] `actionset.ExceptionsCheckIQ`

2. *Qualifier specification modes*:
    - `EXCEP_QUAL_NORMAL`: describes the object that actually provides the script interface method
    - `EXCEP_QUAL_TRIGGER`: describes the triggering object of the condition
3. *Exception specification*:
    - `EXCEP_ALL`: work on all exceptions
    - `EXCEP_EXCEP`: look up only exceptions of the given name
4. *Exception set specification*:
    - `EXCEP_SET_ALL`: work on all sets
    - `EXCEP_SET_SET`: look up sets specified by set names

These flags have to be set by the runtime engine which interprets the script and creates the instruction object with the parameters. New script commands have to be added to the XML schemas. The parsing and interpretation mechanism of the script engine have to be enhanced, to make use of the new instruction.

Every class that shall is created from an XML representation has a constructor which takes an XML node (`Node`[47]) as a parameter. This applies especially to the `Action` object which uses this mechanism to create the conditions and instruction specified in the script.

After the creation of `ExceptionCheckIQ` the `perform` method will be invoked when the programme counter of the runtime script reaches the `<exceptionCheck>` statement. That method starts the exception look-up process by the invocation of `lookupCond` on the `ExceptionLookup` object with the necessary information regarding the object representation (type and mode), the names of the exception sets and the exception names.

The look-up mechanism is mainly the same as during the normal exception handling. In case of the normal exception handling, an object (mostly a simulation component) will invoke the exception look-up on `ExceptionLookup`. In case of the new instruction the invocation of that process is done by the

[47] `org.w3c.dom.Node`

instruction. For aiding the added representation mechanisms, the internal look-up mechanism, mainly implemented in `getbyFlag` of `ExceptionLookup`, has to be extended. Presume, that the script defines an exception look-up based on objects and on a special exception set. The first step would be the look-up of all the exceptions that are relevant according to the specified object representation. In the second step, this choice will be shrunken according to exceptions that are stored in the exception set. Note that there is no guarantee for the execution order of these steps. An ordering mechanism is not necessary, because the result would not be effected. An ordering mechanism would only influence the performance of the operation. This is not critical because the new instruction is exceptional.

Chapter 5

A Sample Lesson

This chapter will describe the steps that are necessary to implement a simple lesson. The lesson that will be created is aimed for novices in the subject simulation. The students will learn to build and configure simulation networks by the help of this lesson. Therefore, the major focus is on the abstraction of real processes to simulation networks. The lesson consists of two parts. In the first part, the user has to configure a given simulation model. Later on the user will reconfigure and extend that model according to a given task. The first part is implemented whereas the concepts of the second one will be only described.

5.1 Concepts

5.1.1 The task

Edward Miller, the owner of a filling station, investigated the refuel habits of his customers. The filling station consists of 3 pumps working in parallel. Furthermore, there is a drive-in cash box that has two lanes. The filling station can be reached through an entrance lane and left via an exit lane. Figure 5.1 will show a map of Miller's filling station. Miller detected that if all three pumps are occupied, no more cars will turn into the lane leading to his filling station because the customers are always in a hurry and do not

want to waste time.

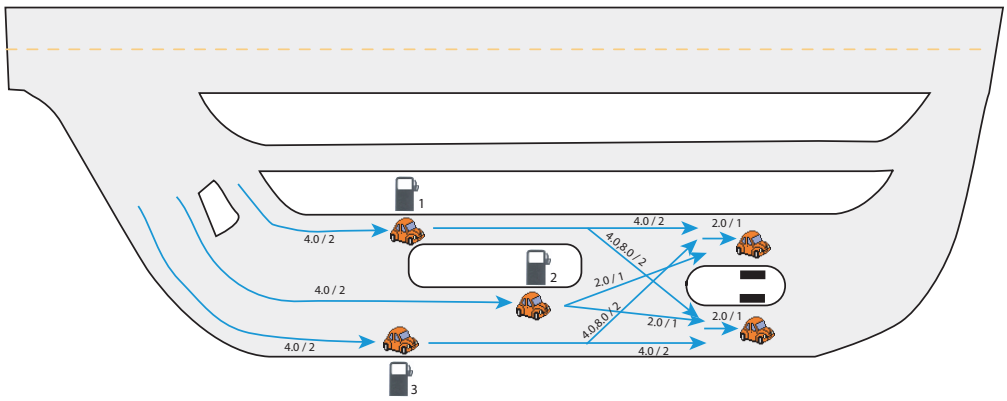

Figure 5.1: A map of the filling station

Miller discovered that his customers buy between 20 and 70 litres of fuel. Most customers fill 30 litres into their cars. Technical papers about the pump stations disclose that station 1 and 2 are quite new (built in 1999). Station 3 was built in 1980. According to the papers, station 1 and 2 deliver 12 litres per minute, station 3 can only deliver 8 litres per minute.

After having refilled their cars, the customers will enter one of the cash box lanes to pay for the fuel. Therefore, they choose the queue in which they will probably wait the least time. The queue entrances in the model are not real stations where the customers spend time. They are only there for routing reasons. The installation of a new computer system bought in 2000 allows the next consumer to use the pump 5 seconds after the previous user has left. He does not have to wait until the previous user of the station has payed.

Customers spend approximately one minute at the cash box. This time deviates about 50%. Between two customers, the clerks at the cash boxes need a recreation time between 10 seconds and 5 minutes. A normal length of a pause is 20 seconds. At the weekend, there is only one cash box open which leads to a traffic jams. Additionally, the clerk at the open cash-box needs longer pauses at the weekends (60,300,150).

The filling station is open 8 hours a day. Cars arrive on average every two minutes, but the interarrival times vary.

a.) Start the examination of the problem by the configuration of the given simulation model. The properties of the simulation components have to be set to reflect the real demands as described above. Press the "finish" button when you think you have completed the task. The model will run, if you have configured it correctly. The simulation will run from Friday to Monday.

b.) Reconfigure the given simulation model. Choose a solution that allows minimising the number of cars that turn around and do not enter the filling station. Keep the costs of the solution in mind. Some possible costs are displayed in the table below.

possibilities	**costs**
hire another clerk who works on the weekend	1000 Euro
update pump 3	200 Euro
buy a new pump	3000 Euro
...	

5.1.2 Thoughts about the learning contents

The learning process will be divided into two different parts. Both parts are defined on the same basic scenario. This enables the user to gain an understanding of the different steps that are necessary to abstract a real scenario to a simulation model.

The first part will start with the presentation of the situation. After that, a model of the filling station will be built as a demonstration. A proper delay is chosen, allowing the learners to follow each step in detail.

The learner's task will begin after the model is created. Therefore, the system has to turn on the exception handling mode for allowing user interaction. The aim of the first part of the lesson is the understanding of the different components and their settings. This part is designed for learners who are new to the subject Simulation. It is only assumed that the learners understand the different simulation components and their possible settings. The existence of a pre-built simulation model (built during the demonstration) will help the students to understand the basic structure of the given simulation model. To let the learner build the simulation from scratch would also be possible. However, it would be more difficult and therefore not desirable for novices.

The first part of the lesson will end if the learner presses the "finish" button. After that, the system will analyse the model and return whether the settings are correct. Furthermore, the system will give some feedback to the learner in case of wrong settings. A proper level of feedback based on the actual knowledge level of the learner has to be chosen. If a learner clicks the "finish" button, the system will not only display the result in a message box, it will also run the simulation to display the results of the settings in the case they are correct. If the learner does not reach the goal of the lesson, the system will allow a retry based on the actual settings. The learner can always run the simulation to verify the settings by comparing the number of served customers to the desired number. By doing so, the learner will notice that that the simulation runs in real time per default, which would lead to a simulation run that takes several days. To retrieve results more quickly the learner will have to use the speed slider of SimTool.

The second part of the lesson is on a more advanced level. The initial situation of this part is the simulation model from the first part with correct settings. The aim of the second part is the improvement of the refuelling process at the filling station. The learners who should have gained some basic knowledge on simulation will have to reconfigure the simulation model. Changes in the underlying structure are possible, for example, adding a new pump. The system will recommend some possibilities to solve the task. Each change will cause costs as in the real world. The learner has to consider this to reach a good solution of the problem. The throughput and the estimated costs of the solution will be used to measure the goodness of the student's solution.

Because of the more "game-like" nature of the second part, the system should only interact if major errors occur. Especially, when the students "play" with the simulation model they would be annoyed quickly if message boxes were popping up constantly.

The icons used in the simulation model should reflect the situation. All features of SimTool which the user should not use are disabled not to distract the user, for example the tool tree. Saving the model is possible at any time during the lesson.

The solution of the task can be found at Table 5.1.2.

component ID	property	value
Source:		
Entrance	maxMe	0
	create interval	exponential (120.0)
	make-ready time	constant (0.0)
	routing	best alternative
Single Station:		
FillingStation 1,2	processing time	triangular (100,350,150)
	make-ready time	constant (5.0)
	routing	best alternative
FillingStation 3	processing time	triangular (150,525,225)
	make-ready time	constant (5.0)
	routing	best alternative
QueueEntrance 1,2	processing time	constant (0.0)
	make-ready time	constant (0.0)
	routing	best alternative
CashBox 1,2	processing time	triangular (30,90,60)
	make-ready time	triangular (10,300,20)
	routing	best alternative
Link:		
Link l1,l2,l3,l4,l9	transport time	constant (4.0)
	maxBe	2
Link l6,l7	transport time	constant (2.0)
	maxBe	1
Link l5,l8	transport time	uniform (4.0,8.0)
	maxBe	2
Link l10,l11	transport time	constant (2.0)
	maxBe	1
Link l12,l13,l14	transport time	constant (3.0)
	maxBe	2

Table 5.1: Solution of the example lesson

5.2 Implementation

5.2.1 Necessary steps

The following enumeration will describe the tasks that must be performed during the implementation of a lesson. Normal items mark features or tasks that are quite easy to implement, emphasised items refer to parts which need more focus.

1. Draw the simulation model on a sheet of paper.
2. Create the lesson file.
3. *Create all instructions that are necessary for the demonstration mode.*
4. Create the block with demonstration steps in the runtime script.
5. Measure the knowledge level of the student. Define the amount of feedback that is needed.
6. *Think about the arrangement of exceptions in sets.*
7. Write exceptions.
8. *Create necessary interface methods for conditions.*
9. Create the exception set files and connect them with the runtime script.
10. Create other blocks in the runtime script.

5.2.2 The lesson file

The lesson file contains the overall settings of the system applying for the learner. These settings are, for example, the settings of the GUI, the components, distributions and routing algorithms that are used. The script author has always full access to the features of the system. Thus, a component can always be added to the simulation model by the script even if the use of that component is not allowed in the lesson file.

Furthermore, the lesson file defines the properties that will be displayed in the property dialogues of the components. It is divided into different sections. The global section is defined as:

```
# distributions
distributions = Const,Normal,Uniform,Triangle,Exponential

# routing algorithms
routing = Fixed,WeightedRouting,RoundRobin,Best

# tools in the tool tree
tools = Select

# simulation components
components = Source,SingleStation,Destination

# movable elements
movables = Car

# menus
menuFile = Save,SaveAs,Exit
menuHelp = About
menuView = Comment,Toolbar,Zel,Grid,GridOptions

# buttons in the tool bar
buttons = Save,Start,Init,Simspeed

# Localisation
language = en
country = GB
```

Some GUI elements will not be displayed because learners do not need them to solve the given task. They would only distract the student. The tool tree is disabled because there is only one thing that would be in the tool tree: the selection tool. This tool will be selected via an instruction in the demonstration block.

The second section will define component specific settings. The learner has to configure the different components in the model. The following components are used in the simulation model:

- *Source*:
  This component represents the entrance of the filling station.
- *Single station*:
  This component reflects a petrol pump.
- *Destination*:
  This component reflects the exit of the filling station.
- *Link*:
  Links are used to represent lanes in the filling station.

The learner has to apply the settings to the given simulation model. All properties regarding to the distributions and routing algorithms of the components have to be enabled. The property that allows renaming the components is disabled. All the other options in the pop menu like "delete" or "move" are disabled. Special restrictions for distributions or routing algorithms do not apply. That means all components will use all globally activated routing algorithms and probability distributions. The component specific section of the lesson file will be displayed below.

```
#source
Source.popup=Properties
Source.props.general=id,createelements,maxelements
Source.props.run=mkreadytimes,createInt
Source.props.routing=routing

#single station
SingleStation.popup=Properties
SingleStation.props.general=id
SingleStation.props.run=mkreadytimes,prctime
SingleStation.props.routing = routing

#destination
Destination.popup=Properties Destination.props.general=id
Destination.props.run=mkreadytimes,destdel

#link
Link.popup=Properties Link.props.general=id
Link.props.run=trtime,maxelements
```

Additional information about the syntax of the lesson files can be found in Chapter 3.2.1 and 3.2.2.

5.2.3 The instructions

Instructions are needed to place the simulation components into the working area of SimTool. The instruction that is necessary to do this is `<newComponent>`. The excerpt below will show its usage to add the entrance of the filling station to the model.

```
<rt:action>
  <newComponent name="Entrance" className="Source"
                posx="10" posy="180"/>
</rt:action>
```

For creating links there is a special instruction which will set a link between two simulation components (`<link>`).

```
<rt:action>
  <link src="Entrance" dest="FillingStation1" name="l1"/>
</rt:action>
```

The description of all implemented instructions can be taken from Appendix D.

However, the creation of the simulation network should not be the first step of a lesson. At the beginning of the lesson, the task of the learners should be displayed in the help panel of SimTool. Therefore, the `<loadDescriptionFile>` instruction is used.

```
<rt:action>
  <loadDescriptionFile path="lesson/task" />
</rt:action>
```

Furthermore, it is necessary to set the visibility of the different panes in the help panel. That can be done by using the `<visibilityHTMLPane>` instruction. The system can give detailed feedback via the `<setTextResultTab>`. For detailed information, see Chapter 3.6.1.

The simulation model reflects a situation in the reality. For aiding the abstraction and understanding process, the appearance of the components is important. An instruction is used that changes the image representing the component on the simulation workspace (`<setImageComponent>`). This instruction is used in the lesson for indicating that one cash box clerk is missing at the weekend, for example. Therefore, the image of the single station used to represent a cash box will change at the weekend. The path to the image can be specified as a relative path or a valid URL. For differences between both referring mechanisms see the explanation of `<loadDescriptionFile>` in Chapter 3.6.1. The following instruction is used for changing the image of the cash box at the weekend:

```
<rt:action>
  <setImageComponent path="images/singlestationoff.gif"
                     ID="CashBox1"/>
</rt:action>
```

The settings of the simulation model shall be checked when the user explicitly declares to have finished the configuration. Therefore, the state button (see Chapter 3.6.2) is used to implement a "finish" button. If the user presses that button a check of all settings will be triggered. This behaviour is defined by the `<behaviorStateButton>` instruction. At the beginning, a low feedback level is chosen. The feedback level will increase later, depending on the learner's behaviour.

5.2.4 Creating the main block

The simulation network that is used in the lesson will be created by the runtime script. All instructions that are necessary to create the network are stored in one block labelled "main". A delay value is specified to allow the learner to contemplate the steps that are taken to build the simulation model. Apart from the instruction that adds components to the simulation panel,

the instruction to display HTML files and the instructions which control the state button will be used in that block. Figure 5.2 will show a screen-shot of the system after the demonstration has taken place. The last instruction in the demonstration block will enable the exception handling. The entire XML file including the main block can be found in Appendix F.1.

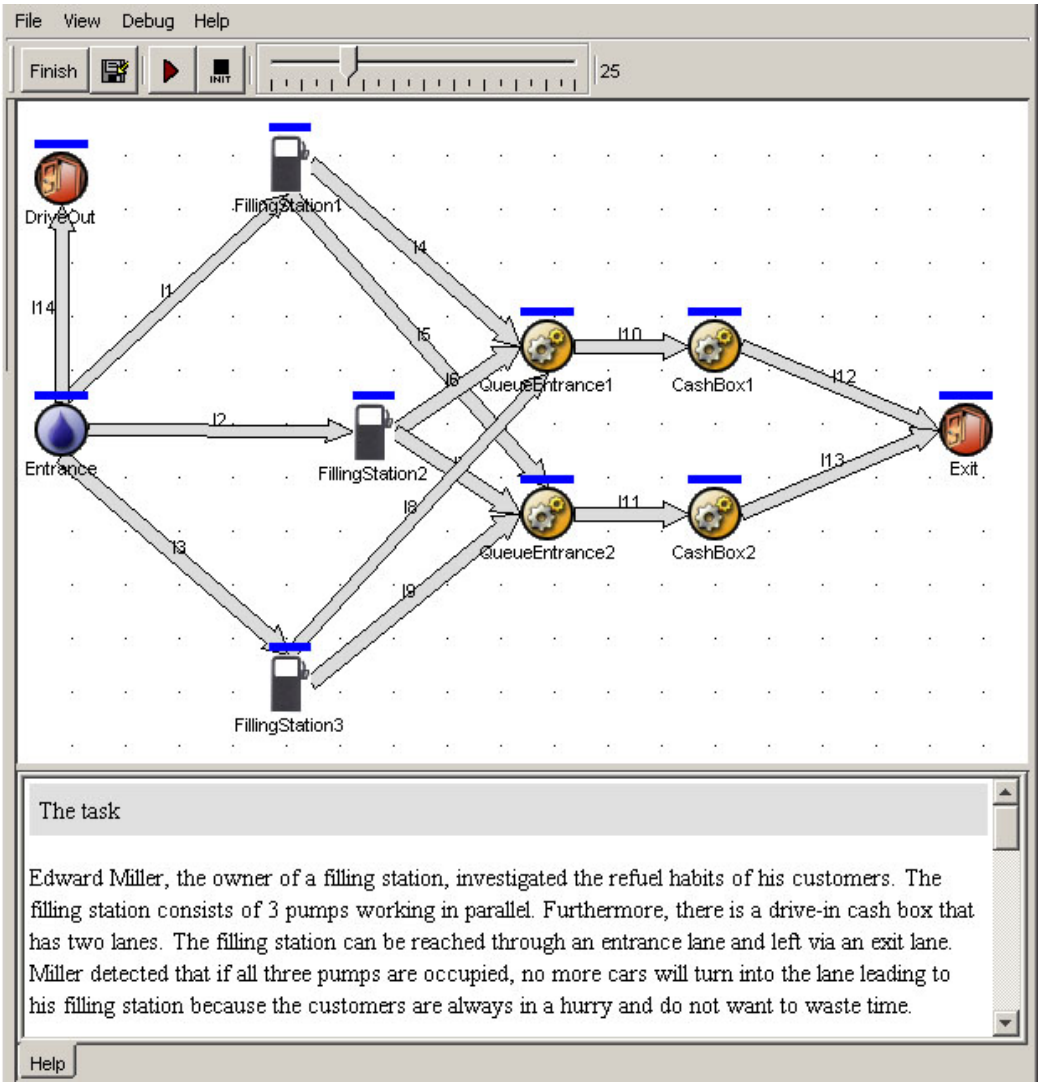

Figure 5.2: A screen-shot of the system after the demonstration

5.2.5 Structuring the exceptions

Different exceptions are structured by adding them to exception set objects. Exceptions can be classified according the items listed below and bundled in exception sets.

1. Exceptions that influence the runtime behaviour of the simulation model will offer a lot of design possibilities to the script author and allow more complex models. However, the use of exceptions in that way will come very close to a programming of a simulation model. Such a programming should be hidden from the learner. An example of the use of such an exception is an exception with a condition based on the simulation time and triggered by the simulation itself. The definition of exceptions defined in that way is further described below. Exceptions of that type are stored in set `Set1`.

2. Exceptions which will fire dynamically either during a simulation run or in the case that the user changes some setting and an immediate feedback is desired, are used to create more realistic models or to provide the user with information during the lesson. An example of such an exception is an exception which changes the icon of a simulation component according to its state. This kind of exceptions is stored in `Set2`.

3. Exceptions that define the solution of the task are the last type. In the sample lesson, more than one exception set is used to provide different feedback levels. One exception in each feedback level has to be defined as the entry point. Three different exception sets are defined `Set3SmallDetailedLevel`, `Set3MediumDetailedLevel` and `Set3HighDetailedLevel`. An additional set (`Set3Finish`) will be used to determine the steps of the system in the case that the learner configured the whole system in a correct way.

The state button will execute the entry point exceptions in the different `Set3` sets. The feedback level chosen by the system depends on the number of times that the learner presses the "finish" button (state button). If the settings of the simulation model are wrong, detailed help depending on the current feedback level will be provided by the system in the help panel.

However, the learner should not always press the "finish" button if he has changed the properties of a component. A better way to verify new settings will be to perform a simulation run to determine whether the desired throughput is reached. Statistical information about the different simulation components will be available to allow the learner to guess correct settings depending on the changes of statistical values. According to the script, the task will be solved if the desired throughput is reached (with a variance) during the simulation time. The simulation time is limited to 4 days.

`Set1` which alters the simulation model dynamically will be used to model the weekend and the business hours of the filling station. That helps creating more realistic models. The next section will explain some exceptions of this category in detail.

The activity diagrams will show the different exceptions and their interactions with the script file. Figure 5.3 will show the main activity diagram. Figure 5.4 will show the activities that occur if the "finish" button is pressed. The components of decision on top of Figure 5.4 reflect the actual condition test. If the conditions are fulfilled the system will perform the instruction that belongs to the exception. This will transform the system into a new state.

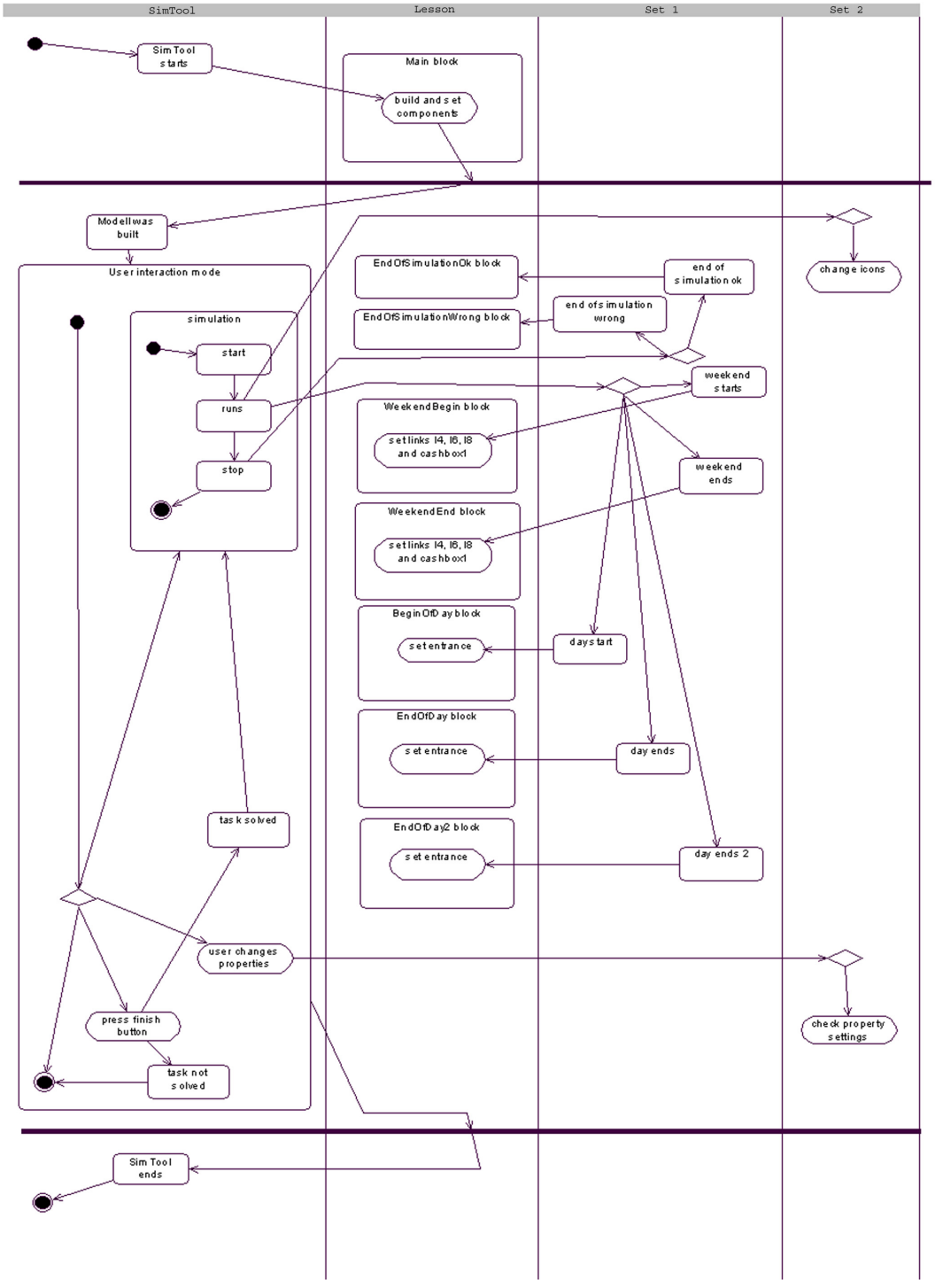

Figure 5.3: Activity diagram for the main lesson

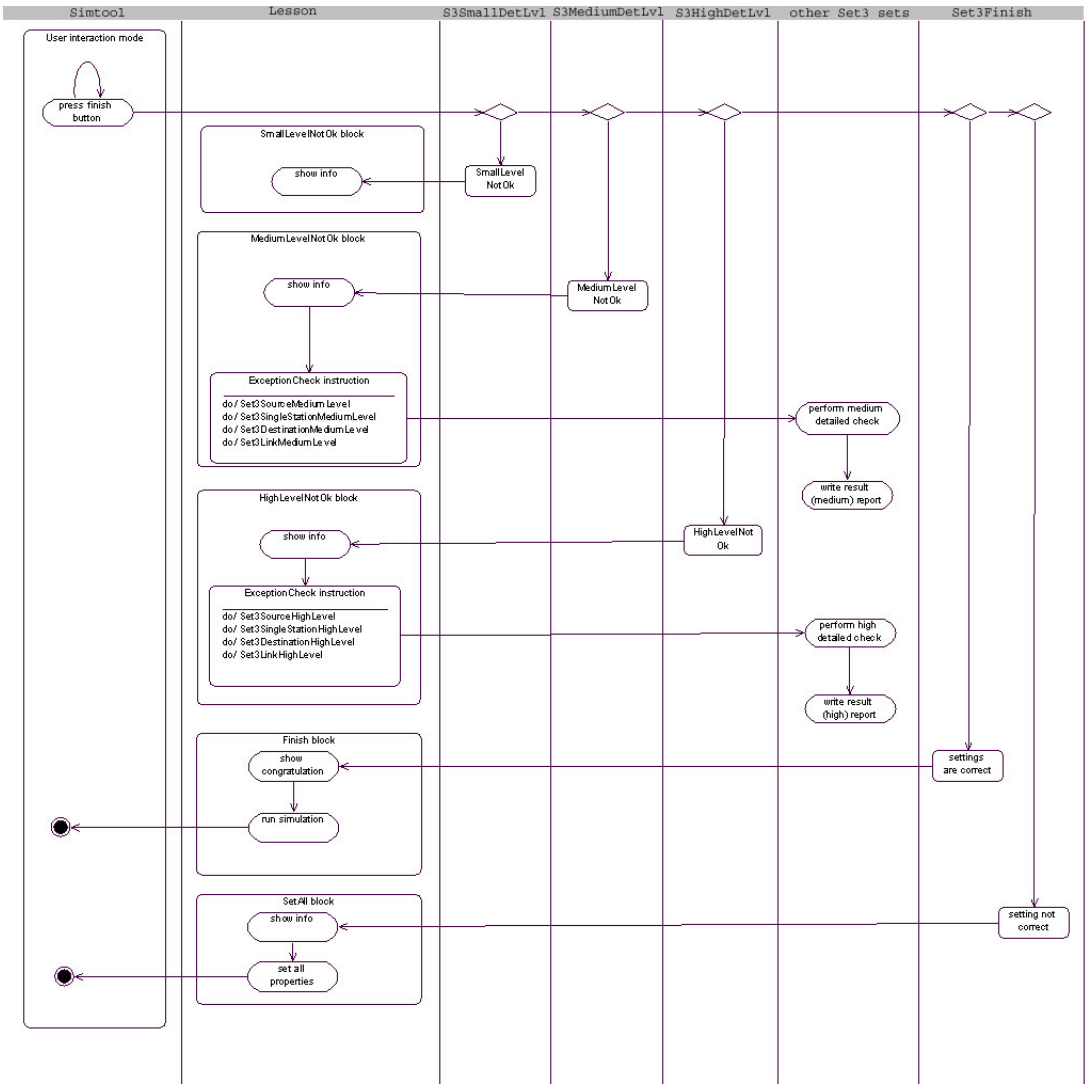

Figure 5.4: Activities after the finish button has been pressed

5.2.6 Example of an exception

The following exceptions named "WeekendBegin" and "WeekendEnd" define the start and the end of the weekend during a simulation run. They are defined in `Set1`, because they change the behaviour of the simulation model. The following XML excerpt will show the two exceptions:

```
<es:action name="WeekendBegin" maxCalls="1">
  <jump label="weekendBegin"/>
   <conditions>
     <condition srcId="FillingStation1"
                function="getSimulationTime"
                classType="instance"
                condType="simulationRun"
                triggerId="gui.SimToolWin"
                triggerclassType="class">
       <returnValue compareOperator="&gt;=" type="stdType">
         <-- the second day is saturday -->
         <value>1:00:00:00.000</value>
         <type>event.Time</type>
       </returnValue>
     </condition>
   </conditions>
 </es:action>

 <es:action name="WeekendEnd" maxCalls="1">
  <jump label="weekendEnd"/>
   <conditions>
     <condition srcId="FillingStation1"
                function="getSimulationTime"
                classType="instance"
                condType="simulationRun"
                triggerId="gui.SimToolWin"
                triggerclassType="class">
       <returnValue compareOperator="&gt;=" type="stdType">
         <-- the fourth day is monday -->
         <value>3:00:00:00.000</value>
         <type>event.Time</type>
       </returnValue>
     </condition>
```

```
    </conditions>
  </es:action>
```

The simulation model runs between Friday and Monday. Therefore, the `maxCalls` attribute of the exceptions is set to one. Thus, every exception will be raised once because there is only one weekend in the simulation time period. The exceptions contain of one condition and one instruction. The condition will be fulfilled if the desired simulation time is reached. In that cases the simulation time reflects the beginning and the end of the weekend. The conditions are triggered by the `SimToolWin`[1] object in case that the simulation is running (condition type `simulationRun`). If the conditions are fulfilled more than one instruction will be performed. A message dialogue will be displayed to inform the user of the event and some settings on the affected simulation components will be changed to ensure the desired behaviour at the weekend.

5.3 Summary of the sample lesson

Table 5.2 will show the different exceptions summarised according to their storage in the different sets.

The following items describe the blocks that are used in the runtime script together with the major tasks that they perform.

- *main*:
    1. Add the simulation components to the simulation workspace
    2. Connect the different components with links
    3. Set icons of the components
    4. Initialise the state button appearance and behaviour
    5. Enter select mode (selection tool) and show task in help panel
- *WeekendBegin*:
    1. Notify the user of the beginning of the weekend

[1] `gui.SimToolWin`

2. Change properties of the involved components ("CashBox2","l4","l6","ll8")
3. Change icon of "CashBox1" to reflect a closed cash box

- *WeekendEnd*:

1. Notify the user of the end of the weekend
2. Change properties of the involved components ("CashBox2","l4","l6","ll8")
3. Change icon of "CashBox1" to reflect an open cash box

- *SmallLevelNotOk*:

1. Show information window that the settings are not correct

- *MediumLevelNotOk*:

1. Show an information window that the settings are not correct and that the system runs at a medium feedback level
2. Enable the result pane of the help panel
3. Invoke exceptions that are stored in the following sets
   - Set3SourceMediumLevel
   - Set3SingleStationMediumLevel
   - Set3DestinationMediumLevel
   - Set3LinkMediumLevel

- *HighLevelNotOk*:

1. Notify the user that the settings are not correct and that the system runs at a medium feedback level
2. Enable the result pane of the help panel
3. Invoke exceptions that are stored in the following sets
   - Set3SourceHighLevel
   - Set3SingleStationHighLevel
   - Set3DestinationHighLevel
   - Set3LinkHighLevel

- *Finish*:

1. Show congratulation dialogue

2. Start the simulation

- *SetAll*:
  1. Show dialogue to notify the user that the settings are still wrong and the correct properties of the simulation components will be set by the system
  2. Set the properties of the components
- *BeginOfDay*:
  1. Notify the user of the beginning of a new day
  2. Change properties of the "Entrance"
- *EndOfDay*:
  1. Notify the user of the end of a day
- *EndOfDay2*:
  1. Change properties of the "Entrance"
- *EndOfSimulationOk*:
  1. Stop the simulation
  2. Notify the user that the property settings of the simulation components are correct
- *EndOfSimulationWrong*:
  1. Stop the simulation
  2. Notify the user that the property settings of the simulation components are not correct

Set1				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
weekendBegin	sim time≥1:00:00:00.000	jump weekendBegin	maxCalls=1	starts the weekend
weekendEnd	sim time≥3:00:00:00.000	jump weekendEnd	maxCalls=1	ends the weekend
EndofSimulationOk	sim time≥4:00:00:00.000 ∧ (throughput exit ≥ 640 ∧ ≤ 675)	jump EndOfSimulationOk	maxCalls=1	simulation ends in a proper way
EndofSimulationWrong	sim time≥4:00:00:00.000 ∧ (throughput exit ≤ 640 ∨ ≥ 675)	jump EndOfSimulationWrong	maxCalls=1	simulation did not end correct
BeginOfDay1	sim time≥0:00:00:00.000	jump BeginOfDay	maxCalls=1	start day1
EndOfDay1	sim time≥0:08:00:00.000	jump EndOfDay	maxCalls=1	end day1
EndOfDay1Reset	sim time∧0:09:00:00.000	jump EndOfDay2	maxCalls=1	reset
BeginOfDay2	sim time≥1:00:00:00.000	jump BeginOfDay	maxCalls=1	start day2
EndOfDay2	sim time≥1:08:00:00.000	jump EndOfDay	maxCalls=1	end day2
EndOfDay2Reset	sim time∧1:09:00:00.000	jump EndOfDay2	maxCalls=1	reset
BeginOfDay3	sim time≥2:00:00:00.000	jump BeginOfDay	maxCalls=1	start day3
EndOfDay3	sim time≥2:08:00:00.000	jump EndOfDay	maxCalls=1	end day3
EndOfDay3Reset	sim time∧2:09:00:00.000	jump EndOfDay2	maxCalls=1	reset
BeginOfDay4	sim time≥3:00:00:00.000	jump BeginOfDay	maxCalls=1	start day4
EndOfDay4	sim time≥3:08:00:00.000	jump EndOfDay	maxCalls=1	end day4
EndOfDay4Reset	sim time∧3:09:00:00.000	jump EndOfDay2	maxCalls=1	reset
Set2				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
checkEntranceSettings	maxMe entrance < 900 ∧ > 0	show dialogue		checks number of elements produced by the entrance
checkLinkSettings	$\vee_{i=0..9,10..14}$ Link l_i maxMe>0	show dialogue		check if link can process elements
Set3SmallDetailedLevel				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
SmallLevelNotOk	check property settings for errors	jump SmallLevelNotOk	maxCalls=2	check if settings are wrong
Set3MediumDetailedLevel				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
MediumLevelNotOk	check property settings for errors	jump MediumLevelNotOk	maxCalls=5, minCalls=3	check if settings are wrong
Set3HighDetailedLevel				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
HighLevelNotOk	check property settings for errors	jump HighLevelNotOk	maxCalls=8, minCalls=6	check if settings are wrong
Set3Finish				
Exception name	*Condition*	*Instructions/Jump*	*flags*	*Description*
Finish	check property settings	jump Finish		check if settings are correct
NotFinish	check property settings for errors	jump SetAll	maxCalls=10, minCalls=9	check if settings are wrong
other Set3 sets				
Exception set name	*Condition*	*Instructions/Jump*	*flags*	*Description*
Set3SourceMediumLevel	check source property settings	display results		check for errors
Set3DestinationMediumLevel	check destination property settings	display results		check for errors
Set3SingleStationMediumLevel	check single station property settings	display results		check for errors
Set3LinkMediumLevel	check link property settings	display results		check for errors
Set3SourceHighLevel	check source property settings	display results		check for errors
Set3DestinationHighLevel	check destination property settings	display results		check for errors
Set3SingleStationHighLevel	check single station property settings	display results		check for errors
Set3LinkHighLevel	check link property settings	display results		check for errors

Table 5.2: The exception sets

Chapter 6

Conclusion

6.1 Summary

This work is about improving the teaching in web-based virtual learning environments. A simulation tool has been developed that can be integrated in a web-based learning environment as a discipline specific module for the subject Simulation. This module, called SimTool, is not considered as a replacement for classical lectures or exercises, it should be used in addition to lectures in the subject Simulation.

Simulation is an important subject with high practical relevance. It is used for modelling in various disciplines. Especially, the operations research uses simulation and claims it to be part of their subject.[1] Unfortunately, courses in simulation are quite theoretical and lack practical examples. Existing simulation tools, for example eM-Plant[2], are too complicated for using in courses, because the learners would have to spend too much time in learning how to use these applications rather than learning how simulation work.

SimTool is designed as a simulation tool that is easy to use. Especially, the learners do not need any introduction for using the tool, because the GUI of SimTool is self-explanatory. It should be possible for the learner to build simple simulation models without having to programme. SimTool does not

[1] see [DD95] p. 206
[2] see [Tec02]

provide a script language for learners. The modelling facilities are therefore quite limited. However, the focus of the development is on the visualisation of basic simulation principles on small models.

As mentioned above, a script language for learners does not exist in SimTool. But there is a script language that can be used by teachers to create demonstration scenarios. It is possible to perform each action that a user can take via script instructions. However, not all possible instructions are implemented yet, because the aim of this work is not the creation of a fully functional application, it is rather the improvement of an existing prototype to implement learning supportive methods. But it is not difficult to add the missing instructions when they are needed. There are two possible ways for doing so: writing a script interface method or creating a specific instruction qualifier. Using the first way to implement a new instruction, a method with a special prefix ("sc_") is added to an object (a simulation component, for example). This method can be called from the script without having to change the script parser. The second way, the creation of a new instruction qualifier, leads to a simpler command syntax in the script file. However, the parser has to be changed to interpret the new command. The object involved with the new instruction usually does not have to be altered. The necessary steps for adding an instruction are described using an example in Chapter 4.6.

Extensibility and modularity are major design goals of SimTool. As described above, it is easy to add new instructions to the script language. Also, simulation components, movable elements and probability distributions can be added to SimTool easily as well. If they are implemented according to the implementation guidelines (see Chapter 3.3.1) they can be plugged in at runtime such that SimTool does not have to be recompiled.

For creating tutorials which give support the learning process by providing interaction possibilities, another feature of SimTool can be used: the exception handling. The exception handling concept enables teachers to configure SimTool such that it can react to of the learner's actions. For example, it is possible to specify a model solution for a given task which the actual model will be compared to. In the case that the learner's solution equals the model solution the task can be considered as solved. Furthermore, it is possible to inform the learner when he commits errors, which is especially useful for those cases where the learner changes settings of the simulation model to values that do not make any sense. An example of a tutorial lesson is described in Chapter 5 in detail.

In the sense of the learning theories described in Chapter 2.1.2, lessons used in SimTool can be seen as intelligent tutorials, because exceptions can be used to imitate decisions of a teacher. In addition, it can be seen as a simulation, because the scenarios which the learner is confronted to can be considered similar to real scenarios. Especially, as in practice people, who create simulation models, use similar tools which only differ to SimTool in their complexity. SimTool also allows the user to "play" with simulation models, so it can be considered as a learning game.

SimTool fulfils the demands of the cognitivism. It is not a behaviouristic practice and drill programme. The learner actively participates in the learning process via interaction with the system. The system is not only able to answer whether the input of the user is correct or not, it can furthermore adapt its behaviour according to the input (change the knowledge level and the verbosity of the feedback, for example). Feedback can be given either immediately or be deferred, until the user presses the "finish" button, for example.

It is possible to create lessons in SimTool that fulfil constructivistic demands. In these lessons an initial model is not presented. The learner has to start modelling from scratch. This situation is authentic, compared to the "real" work of creating simulation models. The demands of an independent control over the learning process and the order of the lessons do not apply to SimTool. The learning environment, where SimTool is integrated in, is responsible for the order of the lesson. If it allows the learners to define their own learn-paths, these demands will be fulfilled.

6.2 Possible extensions

6.2.1 User management

At the moment, SimTool treats all its users equally. Novices and professionals receive the same feedback. If SimTool knew who the actual learner is and which knowledge group the learner belongs to, it could adjust the feedback level according to the learner's knowledge level.

However, the feedback level can be adjusted at runtime according to the learner's behaviour. In the sample lesson (see Chapter 5) this feature will be used if the "finish" button has been pressed multiple times. For example, the feedback level is changed from "low" to "medium" after the second time the learner hits the finish button without having solved the task.

Virtual learning environments usually use databases for user management. A connection to that database should be made available for SimTool. Not only the knowledge level could be stored there. Furthermore, it would be appropriate to store configuration settings of the lesson file in the database as well. The lesson file is a standard Java property file. Thus, the settings are organised in key value pairs and can be easily transferred to a relational database. The lesson file is accessed via the class `Lesson`[3]. If the lesson file were transferred to a database, only the implementation of that class would have to be changed. The other classes of SimTool do not care about the location or the format of the lesson file.

In addition to a user management, a database could provide exceptions which describe the standard behaviour of components. These exceptions, which should depend on the learner's knowledge level, could be used directly by lesson authors who would not have to rewrite them. An exception based on a link that warns the learner that the link is not usable anymore, because no elements will be forwarded on it if the learner sets the maximum number of elements, allowed on this link to 0 is an example.

6.2.2 Authoring tool

The script files and especially the exception set XML files are hard to read. Even with an XML editor it is difficult to write these files. However, the use of XML has certain advantages. Probably in the next JDK release (1.5), the support for schema validation will be included.[4] The, it will be possible to verify if an instance XML document is valid according to an XML schema which represents a formal specification of the document.[5] This verification can be done independently from SimTool, so syntax errors can be detected very easily without using SimTool.

[3] `simulation.Lesson`

[4] see [SW02] p. 344

[5] Validation can already be done if external XML parsers such as Xerces (see [Apa02]) are used.

For creating more complex lessons, an authoring tool is needed. This tool should help the teacher for constructing scripts and exception sets. Therefore, specific instruction qualifiers and script interface methods for using in instructions and conditions should be offered as modules. The lesson author should have the possibility to connect these modules graphically via a GUI.

6.2.3 Modularity of scripts

In complex lessons, exception sets are usually quite large files. However, not every exception is based on different conditions. In contrast, there are a lot of exceptions that differ only in one if its conditions, whereas the other ones are the same. Therefore, having the possibility to define the condition once (using a unique ID) and using only references to this definition in the exceptions would help the authoring of exception sets. These sets would be easier to maintain since a change in the model solution of the lesson does not result in changing each occurrence of the same condition. Having a reference mechanism will allow to change only the defining occurrence. This mechanism should be implemented for instructions as well.

The branching facilities of the script should be extended. At the moment, it is possible to perform conditioned and unconditioned jumps (using a `<jump>` instruction). It would be useful to have a `<jumpSubroutine>` instruction. For using this instruction it would be possible to jump to another block and continue the execution of the script with the instruction that follows after the `<jumpSubroutine>` instruction. The normal `<jump>` instruction does not return to the original block after the execution of the target block.

Another useful feature in the sense of modularity would be the possibility to load more than one script. Additional scripts should be made executable by an instruction. This instruction would enable a separation of demonstration and the model itself, allowing the module to be reused easily in different lessons.

6.2.4 Variables

The script language does not have variables. However, it would be useful to have variables or registers that can be used to store values that indicate a

state or that contain the value of properties at a certain point of time.

6.2.5 Statistical evaluations

The implemented statistic is able to store a variety of different statistical values. Only a small amount of the provided functionality is used yet. The storage of values over a specific period of time is possible, for example. However, custom-tailored visualisations of these values have to be implemented (for example a graph showing the number of elements on a link in dependence of the time). These visualisation components should be implemented in a modular way to fit the underlying concepts of SimTool.

6.2.6 Components

It is possible to create vivid tutorial lessons with the implemented amount of components. However, the system will benefit from more components. With more components more complex models can be developed with less effort. An example of a useful new component would be a parallel station.

On the other hand the existing components should be enhanced. Thus, new properties should be added. An example of a useful property is the down time (mean time between failure and mean time to repair).

6.2.7 Dynamic recording of teachers' actions

The current save mechanism should be extended to enable the system to record a teacher's action. At the moment, the system will save a static snap-shot of the simulation model created by the teacher. The teacher could "play" this model by executing it as a script. A proper delay can be chosen to delay the different recorded actions and perform a demonstration. However, the order which the model has been created in is not reflected by the script. Therefore, a recording of the teacher's action as a video demonstration which reproduces exactly the performed actions should be implemented.

6.3 Outlook

This section will present some ideas of scenarios which could be reflected by lessons. Especially complicated coherences like the warm-up period and confidence intervals should be taught by using interactive tutorials in SimTool. These coherences can be visualised there in a more motivating and self-explanatory way than in books.

6.3.1 Warm-up period

A help desk can be used as a good example to introduce the term "warm-up period" by using a tutorial. That tutorial could ask the learner to determine the average time that a customer of the help desk has to wait in the phone queue. The learner shall determine the retention time by running the simulation model. The system will stop the simulation after a certain time (still at the warm-up period) and demonstrate that the resulting retention time differs from the retention time measured in the reality. These real values were determined in advance by examining a real help desk which is available 24 hours a day. The system will inform the user that this deviation results from the warm-up period. The demonstration will be assisted by diagrams which display the retention time in dependence of the time and in comparison to the real value. The learner will get access to help documents which describe the underlying concepts of the warm-up period. After that, the learner can restart the simulation to obtain better results. The system could support this process dynamically by warning the learner if he intends to stop the simulation too early.

Most of the needed exceptions and instructions are already implemented. The only change necessary would be the implementation of a new condition type which enables the system to look up exceptions when the simulation stops. However, this behaviour can already be implemented by using the state button. In this case the user would be asked to press the state button after a simulation run. The author of this tutorial, has to create a lesson file, the runtime script and the exception set file. The exception set file consists of (at least) the following exceptions:

- An exception that stops the simulation run after a certain time is needed for the first demonstration. This exception contains a condition which checks if the simulation time is greater than a specified time. This condition will be verified during a simulation run (condition type: `simulationRun`, maxCalls: 1).
- An exception that is based on a condition that will be verified when the simulation is stopped by the user. Then the actual retention time will be compared to the specified value of the real help desk.

Additional exceptions can be defined to support the learning process dynamically. The way of defining the runtime script and the lesson file is similar to the way already described in Chapter 5.

6.3.2 Random seeds

Another idea for a tutorial is the implementation of a lesson that illustrates the impact of different random seeds. A demonstration consisting of two nearly identical simulation models can be used. They only differ in the definition of the seed values used in the probability distributions. One network would use the same seed values for all simulation runs, whereas the other one uses different seed values. The seed value is stored in the different distributions. At the moment a possibility to change the seed values at runtime is not present. An additional property should be developed that allows the user to check whether the component uses always the same seed values and to specify the seed value, which will be used for the next simulation run. Furthermore, instructions have to be created which can change these settings via a script.

Appendix A

Sample Configuration Files

A.1 A sample lesson file

Sample files can be found at Appendix F.3 where the lesson file used in the sample lesson is presented and Appendix G.1 where the default lesson file is printed.

A.2 A sample `java.policy` file

This policy file grants permissions to an applet residing at the server `mare.home.de` in the hosted directory `simulation/simtool/`.

```
/* AUTOMATICALLY GENERATED ON Tue Jun 25 20:14:51 CEST 2002*/ /*
DO NOT EDIT */
grant codeBase "http://mare.home.de/simulation/simtool/" {
  permission java.util.PropertyPermission "user.language","write";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "user.home","read";
  permission java.io.FilePermission "/home/simulation","read,write";
  permission java.io.FilePermission "/home/simulation/-","read,write";
  permission java.net.SocketPermission "amiga.home.de","resolve";
  permission java.net.SocketPermission "192.168.0.6:80","connect,resolve";
  permission java.lang.RuntimePermission "modifyThread";
};
```

A.3 A sample applet file

```
<html>
  <head>
    <title>
      SimTool - an interaktive Simualtion Learning Tool
    </title>
  </head>
  <body>
    <applet codebase = "."
          code     = "gui/Simtool.class"
          name     = "SimToolWeb"
          width    = "800"
          height   = "600"
          hspace   = "0"
          vspace   = "0"
          align    = "top"
    >
      <param name="Menu" value="true">
      <param name="HelpWindow" value="true">
      <param name="TreeArea" value="true">
      <param name="Lesson"
             value="http://myserver.mydomain.net/~simulation/
                    simtool/defaultLesson">
      <param name="Script"
             value="http://myserver.mydomain.net/~simulation/
                    simtool/run1.xml">
      <param name="Exceptions"
             value="http://myserver.mydomain.net/~simulation/
                    simtool/basicSet1.xml;
                    http://myserver.mydomain.net/~simulation/
                    simtool/basicSet2.xml">
      <p>Sorry, but your browser does not support Applets</p>
    </applet>
  </body>
</html>
```

Appendix B

Comparison between Version 1.0 and 2.0

	SimTool Ver. 1.0	**SimTool Ver. 2.0**
basic functions		
type of programme		
application	*	*
applet		*
storage of files		
models		
serialisation	*	
XML scripts		*
web-based		*
lessons		
Java property file	*	*
global configuration options	*	*
component specific options		*
GUI visibility options		*
Localisation	*	*
XML files		
web-based		*
help files		
HTML files	*	*
web-based		*

	SimTool Ver. 1.0	**SimTool Ver. 2.0**
extended functions		
scripts		
delayed execution		*
block oriented		*
branching and jumps		*
XML files		*
web-based		*
exception handling		
exception sets		*
loadable exceptions		*
XML files		*
web-based		*
localisation		
German	*	*
German-Germany	*	*
English	*	*
English-USA	*	*
English-GB		*
extensibility		
distributions	*	*
components	*	*
routing algorithms	*	*
script interface methods		*
instructions		*
conditions		*

	SimTool Ver. 1.0	**SimTool Ver.2.0**
important packages		
components		
source	*	*
destination	*	*
single station	*	*
assembly station		*
link	*	*
properties		
`ChooseME`		*
`MaxElements`	*	*
`Quantities`		*
`RandomVariable`	*	*
`Routing`	*	*
`TextLine`	*	*
movable elements		
basic elements	*	*[1]
box element		*
pallet element		*
assembly element		*
distributions		
constant value	*	*
triangular	*	*
uniform	*[2]	*
gaussian normal	*	*
exponential		*
poisson	*	*
exception and script handling		
script interface methods		*[3]
condtions		
generic condition qualifier		*
specific condition qualifier		
actions		
generic action qualifier		*
specific action qualifier		*[4]

[1]: now usable as box element
[2]: formerly named equal distribution
[3]: see Appendix C
[4]: see Appendix D

Appendix C

Implemented Script Interface Methods

C.1 `BaseComponent`

`getAllValuesFromStatisticAttribute`

Description

Script interface method that returns all statistic values stored in the given statistic attribute.

Parameters `java.lang.Integer staticid`: the ID of the statistic attribute

Return value `java.util.Vector`

Example

The following condition will check if the component with the ID "comp1" has been in the state "blocked" at simulation time 5 and 10. The condition will be checked each time the statistic changes.

```
<condition srcId="comp1"
           function="getAllValuesFromStatisticAttribute"
           classType="instance" condType="componentStatistic">
  <returnValue compareOperator="==" type="vector">
    <value>5</value>
    <type>event.Time</type>
    <value>10</value>
    <type>event.Time</type>
  </returnValue>
</condition>
```

getCompID

Description

Script interface method which delivers the ID of the component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check if there exists a component with the ID "comp1" in the simulation model. The condition will be checked when a new component will be added to the model.

```
<condition srcId="simulation.BaseComponent" function="getCompID"
           classType="class" condType="componentAdd">
  <returnValue compareOperator="==" type="stdType">
    <value>comp1</value>
    <type>java.lang.String</type>
  </returnValue>
```

```
</condition>
```

getCompName

Description

Script interface method which delivers the name of the component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check if the name of the component with the ID "comp1" is set to "hugo". The condition will be checked when the property dialogue of the component will be closed using the OK button.

```
<condition srcId="comp1" function="getCompName"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="==" type="stdType">
    <value>hugo</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

getPercentageFromStatisticAttribute

Description

Script interface method which delivers the percentage value of the a statistic attribute which will be specified through it's ID.

Syntax

Parameter `java.lang.Integer statisticid`: the ID of the statistic attribute
The statistic IDs reside in the `StatisticAttribute` class will be shown below:

- `IDLE_TIME` = 0: the idle time of the component
- `PREPARE_TIME` = 1: the prepare time of the component
- `WORK_TIME` = 2: the working time of the component
- `BLOCKED_TIME` = 3: the blocked time of the component

Return value `java.lang.Double`

Example

The following condition will check if the component with the ID "comp1" has a work time value of "10(statistic attribute "WORK_TIME").

```
<condition srcId="comp1"
           function="getPercentageFromStatisticAttribute"
           classType="instance"
           condType="componentStatisticState">
  <returnValue compareOperator="==" type="stdType">
    <value>10.0</value>
    <type>java.lang.Double</type>
  </returnValue>
</condition>
```

getSimulationTime

Description

Script interface method which delivers the current simulation time.

Syntax

Parameters This method does not have parameters.

Return value `event.Time`

Example

The following condition will check if the simulation time is greater than 300. The condition will be checked at every simulation step.

```
<condition srcId="comp1" function="getSimulationTime"
           classType="instance" condType="simulation">
  <returnValue compareOperator="&gt;" type="stdType">
    <value>300</value>
    <type>event.Time</type>
  </returnValue>
</condition>
```

getStatus

Description

Script interface method which delivers the status of the component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Integer`

Example

The following condition will check if the component with the ID "comp1" is not blocked. The condition will be checked at every simulation step.

```
<condition srcId="comp1" function="getStatus"
           classType="instance" condType="simulation">
  <returnValue compareOperator="!=" type="stdType">
    <value>3</value>
    <type>java.lang.Integer</type>
  </returnValue>
</condition>
```

`getThroughput`

Description

Script interface method for getting the throughput of a component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Double`

Example

The following condition will check if the throughput of the component with the ID "comp1" is greater than 4.0. The condition will be checked when the statistic changes.

```
<condition srcId="comp1" function="getThrougput"
           classType="instance" condType="componentStatistic">
  <returnValue compareOperator="&gt;" type="stdType">
    <value>4.0</value>
    <type>java.lang.Double</type>
  </returnValue>
</condition>
```

getThroughputPerMinute

Description

Script interface method for getting the throughput per minute of a component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Double`

Example

The following condition will check if the throughput per minute of the component with the ID "comp1" is greater than 4.0. The condition will be checked when the statistic changes.

```
<condition srcId="comp1" function="getThrougputPerMinute"
```

```
          classType="instance" condType="componentStatistic">
  <returnValue compareOperator="&gt;" type="stdType">
    <value>4.0</value>
    <type>java.lang.Double</type>
  </returnValue>
</condition>
```

getTotalDurationFromStatisticAttribute

Description

Script interface method which returns the total time of the a statistic attribute which will be specified through it's ID.

Syntax

Parameter `java.lang.Integer statisticid`: the ID of the statistic attribute
The statistic IDs reside in the `StatisticAttribute` class will be shown below:

- `IDLE_TIME` = 0: the idle time of the component
- `PREPARE_TIME` = 1: the prepare time of the component
- `WORK_TIME` = 2: the working time of the component
- `BLOCKED_TIME` = 3: the blocked time of the component

Return value `event.Time`

Example

The following condition will check if the component with the ID "comp1" has already worked more than 10 minutes.

```
<condition srcId="comp1"
```

```
            function="getTotalDurationFromStatisticAttribute"
            classType="instance"
            condType="componentStatisticState">
  <returnValue compareOperator="&gt;" type="stdType">
    <value>0:00:10:00.000</value>
    <type>event.Time</type>
  </returnValue>
</condition>
```

getTotalNumberFromStatisticAttribute

Description

Script interface method which returns the total number of times a statistic attribute changed.

Syntax

Parameter `java.lang.Integer statisticid`: the ID of the statistic attribute
The statistic IDs reside in the `StatisticAttribute` class will be shown below:

- `IDLE_TIME` = 0: the idle time of the component.
- `PREPARE_TIME` = 1: the prepare time of the component.
- `WORK_TIME` = 2: the working time of the component.
- `BLOCKED_TIME` = 3: the blocked time of the component.

Return value `java.lang.Integer`

Example

The following condition will check if the component with the ID “comp1” has already changed it‘s working time more than ten times.

```
<condition srcId="comp1"
           function="getTotalNumberFromStatisticAttribute"
           classType="instance"
           condType="componentStatisticState">
  <returnValue compareOperator="&gt;" type="stdType">
    <value>10</value>
    <type>java.lang.Integer</type>
  </returnValue>
</condition>
```

hideStatistics

Description

Script interface method for hiding the statistic window of the component.

Syntax

Parameters This method does not have parameters.

Return value This method does not return a value.

Example

The following instruction will hide the statistic window of the component with the ID "FillingStation1".

```
<instruction srcId="FillingStation1" function="hideStatistics"
             classType="instance">
</instruction>
```

`setImage`

Description

Script interface method which changes the image of a component. It returns the old image. The image path must be specified relatively. Usually, component images will reside in the directory "images".

Syntax

Parameters `java.lang.String relPath`: the path to to the new image of the component

Return value `java.awt.Image`

Example

The following condition will set the image of the component with the ID "FillingStation1" to "images/fill1".

```
<instruction srcId="FillingStation1" function="setImage"
             classType="instance">
  <param name="relPath" type="stdType">
    <value>images/fill1</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

`setName`

Description

Script interface method which sets the name of a component. It returns the old name.

Syntax

Parameters `java.lang.String name`: the new name of the component

Return value `java.lang.String`

Example

The following condition will set the name of the component with the ID "comp1" to "hugo".

```
<instruction srcId="comp1" function="setCompName"
             classType="instance">
  <param name="name" type="stdType">
    <value>hugo</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

showStatistics

Script interface method for showing the statistic window of the component.

Description

Syntax

Parameters This method does not have parameters.

Return value This method does not return a value.

Example

The following condition will show the statistic window of the component with the ID "FillingStation1".

```
<instruction srcId="FillingStation1" function="showStatistics"
             classType="instance">
</instruction>
```

C.2 ComAssemblyStation

compMakeReadyTimeDist

Description

Compare the distribution for the make ready time to the given distribution.

Syntax

Parameters `distribution.Distribution d`: a distribution to compare with

Return value `java.lang.Boolean`

Example

The following condition will check if the selected distribution class of the make ready time of the component with the ID "comp1" is other than a constant value "42". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="compMakeReadyTimeDist"
```

```
          classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</condition>
```

compMakeReadyTimeParams

Description

Compare the parameters of the distribution for the make ready time to the given parameters.

Syntax

Parameters

1. `java.lang.String distClass`: class name of the distribution
2. `java.util.Vector params`: the vector with the parameters to compare with

   0 : first parameter

   1 : second parameter

   ...

Return value `java.util.Vector`

0 : compare result of the first parameter

1 : compare result of the second parameter

...

The compare result works in the following way:

1. If the specified value is greater than the value stored in the distribution, a value < 0 will be returned.
2. If the values are equal, 0 will be returned.
3. If the specified value is less than the value stored in the distribution, a value > 0 will be returned.

Example

The following condition will check if the selected distribution class of the component with the ID "comp1" is "distribution.DistribuutionUniform" with the parameters "0" (lower bound) and "1" (upper bound). The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="compMakeReadyTimeParams"
           classType="instance" condType="componentProperty">
  <param name="distClass" type="stdType">
    <value>distribution.DistributiionUniform</value>
    <type>java.lang.String</type>
  </param>
  <param name="params" type="vector">
    <value>0.0</value>
    <type>java.lang.Double</type>
    <value>1.0</value>
    <type>java.lang.Double</type>
  </param>
  <returnValue compareOperator="==" type="vector">
    <value>0</value>
    <type>java.lang.Double</type>
    <value>1</value>
    <type>java.lang.Double</type>
  </returnValue>
</condition>
```

compProcTimeDist

Description

Compare the distribution for the process time to the given distribution.

Syntax

Parameters `distribution.Distribution d`: a distribution to compare with

Return value `java.lang.Boolean`

Example

The following condition will check if the selected distribution class of the process time of the component with the ID "comp1" is other than a constant value "42". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="compProcTimeDist"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</condition>
```

compProcTimeParams

Description

Compare the parameters of the distribution for the process time to the given parameters.

Syntax

Parameters

1. `java.lang.String distClass`: class name of the distribution

2. `java.util.Vector params`: the vector with the parameters to compare with

   0 : first parameter

   1 : second parameter

   ...

Return value `java.util.Vector`

0 : compare result of the first parameter

1 : compare result of the second parameter

...

The compare result works in the following way:

1. If the specified value is greater than the value stored in the distribution, a value < 0 will be returned.
2. If the values are equal, 0 will be returned.
3. If the specified value is less than the value stored in the distribution, a value > 0 will be returned.

Example

The following condition will check if the selected distribution class of the component with the ID "comp1" is "distribution.DistribuutionUniform" with the parameters "0" (lower bound) and "1" (upper bound). The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="compProcTimeParams"
           classType="instance" condType="componentProperty">
  <param name="distClass" type="stdType">
    <value>distribution.DistributiionUniform</value>
    <type>java.lang.String</type>
  </param>
  <param name="params" type="vector">
    <value>0.0</value>
    <type>java.lang.Double</type>
    <value>1.0</value>
    <type>java.lang.Double</type>
```

```
  </param>
  <returnValue compareOperator="==" type="vector">
    <value>0</value>
    <type>java.lang.Double</type>
    <value>1</value>
    <type>java.lang.Double</type>
  </returnValue>
</condition>
```

getProcessTimeDistributionClassName

Description

Get the class name of the selected process time distribution.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check if the selected distribution class of the component with the ID "comp1" is other than "distribution.Const". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1"
           function="getProcessTimeDistributionClassName"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>distribution.DistributionConst</value>
    <type>java.lang.String</type>
  </returnValue>
```

```
</condition>
```

setMRTDist

Description

Script interface method to set the make ready time distribution to the specified value. The specified distribution will be returned.

Syntax

Parameters `distribution.Distribution dist`: the distribution of the component

Return value `distribution.Distribution`

Example

The following instruction will set the make ready time distribution of the assembly station with the ID "machine1" to a constant value of 42.

```
<instruction srcId="machine1" function="setMRTDist"
             classType="instance">
  <param name="dist" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </param>
</instruction>
```

setPTDist

Description

Script interface method to set the process time distribution to the specified value.

Syntax

Parameters `distribution.Distribution dist`: the distribution of the component

Return value `null`

Example

The following instruction will set the process time distribution of the assembly station with the ID "machine1" to a constant value of 42.

```
<instruction srcId="machine1" function="setPTDist"
             classType="instance">
  <param name="dist" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </param>
</instruction>
```

setQuantities

Description

Set the quantities necessary to produce a me. The quantity vector will be returned.

Syntax

Parameters `java.util.Vector quantities`: the quantities of movable elements that are necessary to produce an assembly element
The vector must have the following format:

0 : name of movable element 1

1 : quantity of movable element 1

2 : name of movable element 2

3 : quantity of movable element 2

...

Return value `java.lang.Vector`

Example

The following instruction will set the quantities of the assembly station with the ID "machine1" to 2× box element, 1× pallet element.

```
<instruction srcId="machine1" function="setQuantities"
             classType="instance">
  <param name="quantities" type="vector">
    <value>BoxElement</value>
    <type>java.lang.String</type>
    <value>2</value>
    <type>java.lang.Integer</type>
    <value>PalletElement</value>
    <type>java.lang.String</type>
    <value>1</value>
    <type>java.lang.Integer</type>
  </param>
</instruction>
```

`setRouting`

Description

Set the routing algorithm used by this component.

Syntax

Parameters `java.util.Vector params`: the parameter of the routing algorithm
The parameters must have the following order:

0 : routing method (`java.lang.Integer`)

- **0** Best
- **1** Round robin
- **2** Weighted routing
- **3** Fixed order

1 : The following cases are possible:

- *Weighted routing and round robin*: reroute (`java.lang.Boolean`)
- *Fixed order*: ID of the first component (`java.lang.String`)

2 : The following cases are possible:

- *Weighted routing*: ID of the first component (`java.lang.String`)
- *Fixed order*: ID of the second component (`java.lang.String`)

3: The following cases are possible:

- *Weighted routing*: weight of the first component (`java.lang.Integer`)
- *Fixed order*: ID of the third component (`java.lang.String`)

4: The following cases are possible:

- *Weighted routing*: ID of the second component (`java.lang.String`)
- *Fixed order*: ID of the fourth component (`java.lang.String`)

...

Return value `java.lang.Boolean`

Example

The following instruction will set the routing algorithm of the component with the ID "comp1" to "best".

```
<instruction srcId="comp1" function="setRouting"
             classType="instance">
  <param name="params" type="vector">
    <value>0</value>
    <type>java.lang.Integer</type>
  </param>
</instruction>
```

setStandardImage

Description

Script interface method to set the image back to the standard system provided image. The old image will be returned.

Syntax

Parameters This method does not have parameters.

Return value `java.awt.Image`

Example

The following instruction will set the standard image of the assembly station with the ID "machine1".

```
<instruction srcId="machine1" function="setStandardImage"
             classType="instance"/>
```

C.3 ComDestination

compDestroyTimeDist

Description

Compare the distribution for the destroy time to the given distribution.

Syntax

Parameters `distribution.Distribution d`: a distribution to compare with

Return value `java.lang.Boolean`

Example

The following condition will check if the selected distribution class of the destroy time of the component with the ID "comp1" is other than a constant value "42". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="compDestroyTimeDist"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</condition>
```

compMakeReadyTimeDist

see `compMakeReadyTimeDist` in `ComAssemblyStation` (Appendix C.2)

setDestrDist

Description

Script interface method to set the destroy time distribution to the specified value. Will return the old value.

Syntax

Parameters `distribution.Distribution dist`: the destroy time distribution of the component

Return value `distribution.Distribution`

Example

The following instruction will set the destroy time distribution of the destination component with the ID "exit1" to a constant value of 42.

```
<instruction srcId="exit1" function="setDestrDist"
             classType="instance">
  <param name="dist" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</instruction>
```

setMRTDist

see `setMRTDist` in `ComAssemblyStation` (Appendix C.2)

setStandardImage

see setStandardImage in ComAssemblyStation (Appendix C.2)

C.4 ComLink

compTransportTimeDist

Description

Compare the distribution for the transport time to the given distribution.

Syntax

Parameters distribution.Distribution d: a distribution to compare with

Return value java.lang.Boolean

Example

The following condition will check if the selected distribution class of the transport time of the link with the ID "l1" is other than a constant value "42". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="l1" function="compTransportTimeDist"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</condition>
```

getMaxME

Description

Get the maximum value for MEs which can be on the link at the same time.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Long`

Example

The following condition will check if the maximum number of movable elements that can be on the link with the ID "link1" at the same time is set to a value lesser than "4". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="link1" function="getMaxME"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="&lt;" type="stdType">
    <value>4</value>
    <type>java.lang.Long</type>
  </returnValue>
</condition>
```

getTransportTimeDistributionClassName

Description

Get the class name of the selected transport time distribution.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check if the selected distribution class of the transport time of the component with the ID "comp1" is other than "distribution.Const". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1"
           function="getTransportTimeDistributionClassName"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>distribution.DistributionConst</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

`setMaxME`

Description

Set the maximum value for MEs which can be on the link at the same time.

Syntax

Parameters `java.lang.Long` `maxME`: the maximum number of movable elements that can be on the link at the same time

Return value `java.lang.Long`

Example

The following instruction will set the maximum number of movable elements that can be on the link with the ID "link1" at the same time to "4".

```
<instruction srcId="link1" function="setMaxME"
           classType="instance">
  <param name="maxME" type="stdType">
    <value>4</value>
    <type>java.lang.Long</type>
  </param>
</instruction>
```

setStandardImage

see `setStandardImage` in `ComAssemblyStation` (Appendix C.2)

setTransportTimeDist

Description

Script interface method to set the transport time distribution to the specified value. The old distribution will be returned.

Syntax

Parameters `distribution.Distribution dist`: the distribution of the component

Return value `distribution.Distribution`

Example

The following instruction will set the transport time distribution of the the link with the ID "link1" to a constant value of 42.

```
<instruction srcId="link1" function="setTransportTimeDist"
             classType="instance">
  <param name="dist" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </param>
</instruction>
```

C.5 ComSingleStation

checkReRouting

Description

Script interface method which checks if the fixed order checkbox in the routing property dialogue has been selected.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Boolean`

Example

The following condition will check if the re-routing checkbox of the component with the ID "comp1" is activated. The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="checkReRoute"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

checkRoutingFixedUserOrder

Description

Script interface method checking if the chosen routing algorithm is "fixed order" and the order is the same as the specified order list.

Syntax

Parameters `java.util.Vector orderList`: a vector with IDs of the components which represents the routing order

Return value `java.lang.Boolean`

Example

The following condition will check if the routing of the component with the ID "comp1" is set to fixed order ("comp2", "comp4"). The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="checkRoutingFixedUserOrder"
           classType="instance" condType="componentProperty">
  <param name="orderList" type="vector">
    <value>comp2</value>
    <type>java.lang.String</type>
    <value>comp4</value>
    <type>java.lang.String</type>
```

```
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

checkRoutingPositionFixedUserOrder

Description

Script interface method checking if the chosen routing algorithm is "fixed order" and the given component is at the given position in the routing order list.

Syntax

Parameters

1. `java.lang.String component`: ID of the component that is looked for
2. `java.lang.Integer position`: the desired position of the component in the order list

Return value `java.lang.Boolean`

Example

The following condition will check if the routing of the component with the ID "comp1" is set to fixed order and "comp2" is on position 4 in the order list. The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1"
function="checkRoutingPositionFixedUserOrder"
```

```
          classType="instance" condType="componentProperty">
  <param name="component" type="stdType">
    <value>comp2</value>
    <type>java.lang.String</type>
  </param>
  <param name="position" type="stdType">
    <value>4</value>
    <type>java.lang.Integer</type>
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

checkRoutingUserWeightedRoute

Description

Script interface method checking whether weighted routing is chosen and a given component has a given weight.

Syntax

Parameters

1. `java.lang.String component`: ID of the component that is looked for
2. `java.lang.Integer weight`: the desired weight of the component

Return value `java.lang.Boolean`

Example

The following condition will check if the routing of the component with the ID “comp1” is set to weighted routing and the weight of the component

with the ID "comp2" is set to "4". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1"
           function="checkRoutingUserWeightedRoute"
           classType="instance" condType="componentProperty">
  <param name="component" type="stdType">
    <value>comp2</value>
    <type>java.lang.String</type>
  </param>
  <param name="weight" type="stdType">
    <value>4</value>
    <type>java.lang.Integer</type>
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

compMakeReadyTimeDist

see `compMakeReadyTimeDist` in `ComAssemblyStation` (Appendix C.2)

compMakeReadyTimeParams

see `compMakeReadyTimeParams` in `ComAssemblyStation` (Appendix C.2)

compProcTimeDist

see `compProcTimeDist` in `ComSource` (Appendix C.2)

compProcTimeParams

see `compProcTimeParams` in `ComAssemblyStation` (Appendix C.2)

compRouting

Description

Script interface method that compares the setted routing algorithm with a given set of parameters.

Syntax

Parameters `java.util.Vector params`: the vector with the parameters to compare with
The parameters must have the following order:

0 : routing method (`java.lang.Integer`)

- **0** best
- **1** round robin
- **2** weighted routing
- **3** fixed order

1 : The following cases are possible:

- *Weighted routing and round robin*: reroute (`java.lang.Boolean`)
- *Fixed order*: ID of the first component (`java.lang.String`)

2 : The following cases are possible:

- *Weighted routing*: ID of the first component (`java.lang.String`)
- *Fixed order*: ID of the second component (`java.lang.String`)

3 : The following cases are possible:

- *Weighted routing*: weight of the first component (`java.lang.Integer`)
- *Fixed order*: ID of the third component (`java.lang.String`)

4 : The following cases are possible:

- *Weighted routing*: ID of the second component (`java.lang.String`)
- *Fixed order*: ID of the fourth component (`java.lang.String`)

...

0 : first parameter

1 : second parameter

...

Return value `java.util.Boolean`

Example

The following condition will check if the routing algorithm of the component with the ID "comp1" to "best". The condition will be checked when the property dialogue is closed by pressing the OK button.

```
<condition srcId="comp1" function="compRouting"
           classType="instance" condType="componentProperty">
  <param name="params" type="vector">
    <value>0</value>
    <type>java.lang.Integer</type>
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

getMakeReadyTimeDistributionClassName

Description

Get the class name of the selected make ready time distribution.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check if the selected distribution class of the make ready time of the component with the ID "comp1" is other than "distribution.Const". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1"
           function="getMakeReadyTimeDistributionClassName"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>distribution.DistributionConst</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

getProcessTimeDistributionClassName

see Appendix C.2

getRoutingChoice

Description

Get the ID of the chosen routing method of the component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Integer`

- **0** best
- **1** round robin
- **2** weighted routing
- **3** fixed order

Example

The following condition will check if the routing of the component with the ID "comp1" is set to "3" (fixed order). The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="comp1" function="getRoutingChoice"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="==" type="stdType">
    <value>3</value>
    <type>java.lang.Integer</type>
  </returnValue>
</condition>
```

`setMRTDist`

see `setMRTDist` in `ComAssemblyStation` (Appendix C.2)

`setPTDist`

see `setPTDist` in `ComAssemblyStation` (Appendix C.2)

`setRouting`

see `setRouting` in `ComAssemblyStation` (Appendix C.2)

`setRoutingFixedUserOrder`

Description

Script interface method setting the routing algorithm to "fixed order" and the order according to the specified order list.

Syntax

Parameters `java.util.Vector orderList`: a vector with IDs of the components which represents the routing order

Return value `java.lang.Vector`

Example

The following condition will set the routing of the component with the ID "comp1" to fixed order ("comp2", "comp4").

```
<instruction srcId="comp1" function="setRoutingFixedUserOrder"
           classType="instance">
  <param name="orderList" type="vector">
    <value>comp2</value>
    <type>java.lang.String</type>
    <value>comp4</value>
    <type>java.lang.String</type>
  </param>
</instruction>
```

setStandardImage

see setStandardImage in ComAssemblyStation (Appendix C.2)

C.6 ComSource

compCIDist

Description

Compare the distribution for the create interval to the given distribution.

Syntax

Parameters distribution.Distribution d: a distribution to compare with

Return value java.lang.Boolean

Example

The following condition will check if the selected distribution class of the create interval of the source component with the ID "source1" is other than a constant value "42". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="source1" function="compCIDist"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="!=" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</condition>
```

`compMakeReadyTimeDist`

see `compMakeReadyTimeDist` in `ComAssemblyStation` (Appendix C.2)

`compMakeReadyTimeParams`

see `compMakeReadyTimeParams` in `ComAssemblyStation` (Appendix C.2)

`compRouting`

see `compRouting` in `ComSingleStation` (Appendix C.2)

`getMakeReadyTimeDistributionClassName`

see `getMakeReadyTimeDistributionClassName` in `ComSingleStation` (Appendix C.5)

`getMaxME`

Description

Get the maximum value of MEs which the source will produce.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Long`

Example

The following condition will check if the maximum number of movable elements that that the source with the ID "source1" will produce is set to a value lesser than "4". The condition will be checked when the property dialogue is closed by pressing OK.

```
<condition srcId="source1" function="getMaxME"
           classType="instance" condType="componentProperty">
  <returnValue compareOperator="&lt;" type="stdType">
    <value>4</value>
    <type>java.lang.Long</type>
  </returnValue>
</condition>
```

setCIDist

Description

Script interface method to set the create interval distribution to the specified value.

Syntax

Parameters `distribution.Distribution dist`: the distribution of the component

Return value `distribution.Distribution`

Example

The following instruction will set the create interval distribution of the assembly station with the ID "source1" to a constant value of 42.

```
<instruction srcId="source1" function="setCIDist"
             classType="instance">
  <param name="dist" type="stdType">
    <value>42</value>
    <type>distribution.DistributionConst</type>
  </returnValue>
</instruction>
```

setMaxME

Description

Set the maximum value of MEs which the source will produce.

Syntax

Parameters `java.lang.Long maxME`: the maximum number of MEs the source will produce

Return value `java.lang.Long`

Example

The following instruction will set the maximum number of movable elements that that the source with the ID “source1” produces to “4”.

```
<instruction srcId="source1" function="setMaxME"
             classType="instance">
  <param name="maxME" type="stdType">
    <value>4</value>
    <type>java.lang.Long</type>
  </param>
</instruction>
```

`setMEType`

Description

Script interface method to set the kind of ME produced by this source component.

Syntax

Parameters `java.lang.String meType`: the type of MEs the source will produce
The following types are available:

PalletElement a pallet element

BoxElement a box element

AssemblyElement an assembly element

CarElement a car

Return value `java.lang.String`

Example

The following instruction will set kind of movable elements that that the source with the ID "source1" produces to pallet elements.

```
<instruction srcId="source1" function="setMEType"
             classType="instance">
  <param name="meType" type="stdType">
    <value>PalletElement</value>
    <type>java.lang.String</type>
  </param>
</instruction>
```

setMRTDist

see `setMRTDist` in `ComAssemblyStation` (Appendix C.2)

setRouting

see `setRouting` in `ComAssemblyStation` (Appendix C.2)

setRoutingFixedUserOrder

see `setRoutingFixedUserOrder` in `ComSingleStation` (Appendix C.5)

setStandardImage

see `setStandardImage` in `ComAssemblyStation` (Appendix C.2)

C.7 Simulation

existsComponent

Description

Script interface method checking whether there exist a component with the given class name in the simulation model.

Syntax

Parameters `java.lang.String` `className`: name of the class

Return value `java.lang.Boolean`

Example

The following condition will check whether there exists a source component in the simulation model. The condition will be checked when a component is added to the simulation model.

```
<condition srcId="simulation.Simulation"
           function="existsComponent"
           classType="class" conditionType="componentAdd">
  <param name="className" type="stdType">
    <value>simulation.ComSource</value>
    <type>java.lang.String</type>
  </param>
  <returnValue compareOperator="==" type="stdType">
    <value>true</value>
    <type>java.lang.Boolean</type>
  </returnValue>
</condition>
```

`lastAddedComponentClassName`

Description

Script interface method that returns the class name of the last added component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check whether the last added component is an instance of `ComDestination`. The condition will be checked when a component is added to the simulation model.

```
<condition srcId="simulation.Simulation"
           function="lastAddedComponentClassName"
           classType="class" conditionType="componentAdd">
  <returnValue compareOperator="==" type="stdType">
    <value>simulation.ComDestination</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

`lastAddedComponentID`

Description

Script interface method that returns the ID of the last added component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check whether the ID of the last added component is "SingleStation4". The condition will be checked when a component is added to the simulation model.

```
<condition srcId="simulation.Simulation"
```

```
           function="lastAddedComponentName"
           classType="class" conditionType="componentAdd">
  <returnValue compareOperator="==" type="stdType">
    <value>SingleStation4</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

lastAddedComponentName

Description

Script interface method that returns the name of the last added component.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.String`

Example

The following condition will check whether the name of the last added component is "hugo". The condition will be checked when a component is added to the simulation model.

```
<condition srcId="simulation.Simulation"
           function="lastAddedComponentName"
           classType="class" conditionType="componentAdd">
  <returnValue compareOperator="==" type="stdType">
    <value>hugo</value>
    <type>java.lang.String</type>
  </returnValue>
</condition>
```

C.8 SimToolWin

getTreeTool

Description

Script interface method that returns the selected tool from the tool tree.

Syntax

Parameters This method does not have parameters.

Return value `java.lang.Integer`

- **0** *init*: initialise
- **1** *move*: move a component
- **2** *delete*: the eraser tool is selected
- **3** *set*: set a component
- **4** *select*: the selector tool is chosen
- **5** *setME*: drop a movable element onto the model

Example

The following condition will check if the eraser tool is selected. The condition will be checked on every simulation step.

```
<condition srcId="simulation.SimToolWin" function="getTreeTool"
           classType="class" condType="simulation">
  <returnValue compareOperator="==" type="stdType">
    <value>2</value>
    <type>java.lang.Integer</type>
  </returnValue>
</condition>
```

`reset`

Description

Script interface method that resets the simulation. Has to be performed to restart the execution of a simulation model. This interface method will also reset the exception handling.

Syntax

Parameters This method does not have parameters.

Return value This method does not return any values.

Example

The following instruction will reset the simulation.

```
<instruction srcId="simulation.SimToolWin" function="reset"
             classType="class"/>
```

`runOrPause`

Description

Script interface method that imitates the pressing of the run/pause button. If the simulation is stopped or paused, the simulation will be started. Otherwise, if the simulation runs, it will be paused.

Syntax

Parameters This method does not have parameters.

Return value This method does not have a return value.

Example

The following instruction will start the simulation if it is paused or stopped.

```
<instruction srcId="simulation.SimToolWin" function="runOrPause"
             classType="class"/>
```

setTreeTool

Description

Script interface method that sets the selected tool from the tool tree.

Syntax

Parameters `java.lang.String` `mode`: the tree tool

select the selection tool

erase the eraser tool

Return value This method does not return a value

Example

The following instruction will set the eraser tool as selected tool in the tool tree.

```
<instruction srcId="simulation.SimToolWin" function="setTreeTool"
             classType="class>
  <param name="mode" type="stdType">
    <value>delete</value>
    <type>java.lang.String</type>
  </param>
</instruction>
```

setTreeToolDelete

Description

Script interface method that sets the eraser tool as the selected tool in the tool tree.

Syntax

Parameters This method does not have parameters.

Return value This method does not return a value

Example

The following instruction will set the eraser tool as selected tool in the tool tree.

```
<instruction srcId="simulation.SimToolWin"
function="setTreeToolDelete"
             classType="class/>
```

setTreeToolSelect

Description

Script interface method that sets the selection tool as the selected tool in the tool tree.

Syntax

Parameters This method does not have parameters.

Return value This method does not return a value.

Example

The following instruction will set the selection tool as selected tool in the tool tree.

```
<instruction srcId="simulation.SimToolWin"
function="setTreeToolSelect"
             classType="class/>
```

Appendix D

Implemented Instructions

D.1 <deleteComponent>

Description

Remove a component from the simulation model.

Syntax

Attributes

name Specifies the ID of the component which is the shall be deleted. The usage of this attribute is required.

Example

The following example shows how to delete the simulation component with the ID “Machine1” from the simulation model.

```
<deleteComponent name="Machine1"/>
```

D.2 <disableExceptions>

Description

Disable the exception handling.

Syntax

The <disableExceptions> instruction does not have attributes nor elements.

Example

In the following example, the exception handling is disabled.

```
<disableExceptions/>
```

D.3 <enableExceptions>

Description

Enable the exception handling. This command should be used as the last command in a block because it implicitly suspends the execution of the script.

Syntax

The <enableExceptions> instruction does not have attributes nor elements.

Example

In the following example, the exception handling is enabled.

```
<enableExceptions/>
```

D.4 <exceptionCheck>

Description

This instruction can be used to force a condition test. The addressed conditions must have the conditonType `script`. It is possible to specify more than one exception. Furthermore, whole sets can be triggered. Another possibility to specify an exception is the usage of the representation of the objects involved with the condition. Two object representations are connected with the condition parts of the exception. One represents the object that triggers the condition test and the other one represents the object that contains the script interface method. Each of these representations can be used separately to refer to an exception.

Syntax

Elements

excep Name of the exception that shall be triggered.

excepSet Name of the exception set that shall be triggered.

qualifier ID of a component or class name of the related object. The following attributes are available:

mode "normal" or "trigger"

normal Trigger conditions that are based on that object.

trigger Trigger conditions that have the specified object as the triggering object.

type "class" or "object"
Used to distinguish if the specified value is a component ID or a class name.

Example

The following example will trigger conditions that are based on the class `simulation.ComSingleStation`, triggered by the object with the ID "MySource", stored in exception set "firstExceptionSet", or that have the name "Excep67".

```
<exceptionCheck>
  <qualifier mode="normal" type="class">
    simulation.ComSingleStation
  </qualifier>
  <qualifier mode="trigger" type="object">
    MySource
  </qualifier>
  <excepSet>firstExceptionSet</excepSet>
  <excep>Excep67</excep>
</exceptionCheck>
```

D.5 <jump>

Description

Jump to the specified block in the script. The given block will be perform all its actions from the beginning.

Syntax

Attributes

label The label of the block which is the target of the jump. The usage of this attribute is required.

Example

In the following example, a jump to the block with the label "hugo" will be performed.

```
<jump label="hugo"/>
```

D.6 <link>

Description

Link to components using a link component (`simulation.ComLink`).

Syntax

Attributes

src Specifies the ID of the component which is the source of the link. The usage of this attribute is required.

dest Specifies the ID of the component which is the target of the link. The usage of this attribute is required.

name Specifies the ID of the link as a string. The name of the link will be set to the same value. The usage of this attribute is required.

Example

The following example creates a link between the component with the ID "MySource" and "Machine1". The link will get the ID "link1".

```
<link src="MySource" dest="Machine1" name="link1"/>
```

D.7 <newComponent>

Description

Add a new component to the simulation model.

Syntax

Attributes

name Specifies the ID of the component as a string. The name of the component will be set to the same value. The usage of this attribute is required.

className Specifies the class name of the component. Package info must be omitted as well as the prefix `Com`. To add a source component (`simulation.ComSource`), for example, the value of the attribute must be `className="Source"`. The usage if this attribute is required.

posx The x coordinate of the position where the component shall be located is specified as an integer. The usage of this attribute is required.

posy The y coordinate of the position where the component shall be located is specified as an integer. The usage of this attribute is required.

Example

The following example adds a single station (`simulation.ComSingleStation`) with the ID "Machine1" to the simulation model. It will be displayed in the main panel at position $(200, 100)$.

```
<newComponent name="Machine1" className="SingleStation"
              posx="200" posy="100"/>
```

D.8 <setTextResultTab>

Description

Set the text in the result panel. It is possible to specify whether to add the text to the current contents of the panel or to set the text as a new text an delete the old contents. The text can be coded as HTML. This makes it possible to define links to other HTML documents which will open in the detailed help pane.

Syntax

Attributes

text Specifies the text that shall be displayed in the result panel.

mode Specifies whether to add the text or overwrite the current contents of the panel. This attribute is optional, its default mode is set.

- "add": add text
- "set": overwrite text

Example

The following instruction in the example adds a line of text ("There are errors in the settings of the source component") to the result panel.

```
<setTextResultTab
  text="Errors in the settings of the source component detected"
  mode="add" />
```

D.9 <selectTool>

Description

Set the selected tool in the tool tree.

Syntax

Attributes

name The name of the tool that shall be selected. The following tools can be used:

- "delete": the eraser tool
- "select": the selection tool

D.10 <setImageComponent>

Description

Changes the image of a component.

Syntax

Attributes

ID The ID of the component, which image has to be changed.

path The relative path to the image. The path resides under `codebase`/simulation/. In case of a missing path attribute the standard image will be set.

mode The mode that is used to find the components:

- "instance": a component will be found by it's ID
  The image will be changed on one component.
- "class": the ID have to be a full qualified class representation (with package info)
  The image will be changed on all instance of the class reflects through the ID.

Example

The following example changes the image of the component with the ID "Station1".

```
<setImageComponent ID="Station1" path="images/Station1.gif"
mode="instance"/>
```

D.11 `<loadDescriptionFile>`

Description

Loads a HTML file into the help pane of the help panel.

Syntax

Attributes

path The path to the HTML file which will be displayed in the help pane. The HTML file can be specified by using a:

- *Relative path*:
  The HTML file has to be named "index.html" to be found in case of a relative path specification. The system will support localised files when using this specification mode. The relative path resides under `codebase`/help/`language`/.
- *Valid URL*:
  The full qualified URL of HTML file.

Example

The following example loads the tasks of a lesson by using a relative path.

```
<loadDescriptionFile path="lesson/task"/>
```

The file that describes the lesson will be accessed via a relative path specification and is named "index.html".

D.12 `<showDialog>`

Description

Shows a message dialogue. The following types of dialogues are available:

- Information dialogue
- Error dialogue

- Warning dialogue
- Plain dialogue

Syntax

Attributes

type Specifies the type of the dialogue. The following values are allowed:

- "info"
- "error"
- "warning"
- "plain"

The usage of this attribute is required.

text Specifies the text which shall be displayed in the dialogue. The usage of this attribute is required.

Example

The following example shows an information dialogue with the text "Congratulations! You have successfully completed the tutorial!".

```
<showDialog type="info" text="Congratulations!
  You have successfully completed the tutorial!"/>
```

D.13 <visibilityHTMLPane>

Description

Changes the visibility of a pane which will appear (if set to visible) at the bottom of the main panel in the help panel. The following help panes are

available:

1. The standard help pane which is used to display the component information.
2. The result pane which is used to display results of exception checking.
3. A pane to display further help. Links in the displayed HTML files on all panes will be opened in this pane when clicked.

Syntax

Attributes

tabFlag Specifies the pane of the panel which shall be displayed or hidden.

- "help": help pane
- "result": result pane
- "helpdetail": extra help pane

visibility Specifies the pane's visibility.

- "true": make the pane visible
- "false": hide the pane

Example

The following example shows the help panel at the bottom of the main panel.

```
<visibilityHTMLPane tabFlag="0" visibility="true" />
```

D.14 <visibilityStateButton>

Description

Changes the visibility of the state button.

Syntax

Attributes

visibility Specifies the visibility of the button.

- "true": make the button visible
- "false": hide the button

D.15 <descriptionStateButton>

Description

Changes the description and tool tip of the state button.

Syntax

Attributes

description The text that is to display on the button.

tooltip The text that will displayed as the tool tip of the button.

Example

The following example changes the button description to “Finish lesson” and sets the tool tip to “Click this button to finish the lesson”.

```
<descriptionStateButton description="Finish lesson"
  tooltip="Click this button to finish the lesson." />
```

D.16 <behaviorStateButton>

Description

Changes the behaviour of the state button. The state button will invoke a exception look-up when the button was pressed. Normally, this exception look-up will retrieve all exceptions which hold at least one condition of the condition type `guiStatebutton`. However, the look-up process can be configured by using this instruction.

Syntax

Elements

excep Name of the exception that shall be triggered.

excepSet Name of the exception set that shall be triggered.

Example

The following example will configure the state button such that the exceptions named “Finish”, “SmallLevelNotOK”,”MediumLevelNotOK” and ”HighLevelNotOK” will be checked when the user presses the button.

```
<behaviorStateButton>
  <excep>Finish</excep>
  <excep>SmallLevelNotOk</excep>
  <excep>MediumLevelNotOk</excep>
  <excep>HighLevelNotOk</excep>
</behaviorStateButton>
```

Appendix E

XML Schemas

E.1 simtool.xsd

The following XML schema is the basic schema for SimTool. It includes basic definitions that are used in both the schema for runtime scripts (see Appendix E.2) and exception sets (see Appendix E.3).

```
<?xml version="1.0"?>

<!-- the base schema with type definitions
     for all other simtool schemas -->
<xsd:schema
  targetNamespace="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
  xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xml:lang="en"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- define type type value pair -->
  <xsd:complexType name="typeValuePair">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="value" type="xsd:string"/>
      <xsd:element name="type" type="xsd:NMTOKEN"/>
    </xsd:sequence>
```

```
    <xsd:attribute name="name" type="xsd:NMTOKEN"/>
    <!-- only used in params -->
    <xsd:attribute name="compareOperator">
    <!-- only used in return values -->
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="=="/>
              <xsd:enumeration value="!="/>
          <xsd:enumeration value="&lt;="/>
          <xsd:enumeration value="&gt;="/>
          <xsd:enumeration value="&lt;"/>
          <xsd:enumeration value="&gt;"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="type" default="stdType">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="stdType"/>
          <xsd:enumeration value="vector"/>
          <xsd:enumeration value="classArray"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <!-- enumeration type of classType -->
  <xsd:simpleType name="classTypeType">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="class"/>
      <xsd:enumeration value="instance"/>
    </xsd:restriction>
  </xsd:simpleType>

  <!-- enumeration type of condType -->
  <xsd:simpleType name="condTypeEnum">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="component"/>
      <xsd:enumeration value="componentProperty"/>
      <xsd:enumeration value="componentAdd"/>
      <xsd:enumeration value="componentDelete"/>
      <xsd:enumeration value="componentMove"/>
      <xsd:enumeration value="componentStatistic"/>
```

```
    <xsd:enumeration value="componentStatisticState"/>
    <xsd:enumeration value="simulation"/>
    <xsd:enumeration value="simulationRun"/>
    <xsd:enumeration value="script"/>
    <xsd:enumeration value="guiStatebutton"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- define list of condition types -->
<xsd:simpleType name="condTypeList">
                  <xsd:list itemType="condTypeEnum"/>
                </xsd:simpleType>
<xsd:complexType name="qualifierType">
<!-- Type for (action) qualifier.
     Condition qualifier uses extension. -->
  <xsd:sequence>
    <xsd:element name="param" type="typeValuePair"
                 minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="srcId" type="xsd:NMTOKEN"
                 use="required"/>
  <!-- id of object (or class) -->
  <xsd:attribute name="function" type="xsd:NMTOKEN"
                 use="required"/>
  <!-- the name of the scipt interface method
       (w/o "sc_" prefix) -->
  <xsd:attribute name="classType" type="classTypeType"
                 use="required"/>
</xsd:complexType>

<!-- define type condition container -->
<xsd:complexType name="conditionContainer">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:choice>
      <xsd:element name="conditions"
                   type="conditionContainer"/>
      <xsd:element name="condition">
        <xsd:complexType>
          <xsd:complexContent>
            <xsd:extension base="qualifierType">
              <xsd:sequence>
                <xsd:element name="returnValue"
                             type="typeValuePair"/>
```

```
              </xsd:sequence>
              <xsd:attribute name="condType" use="required"
                             type="condTypeList"/>
              <xsd:attribute name="triggerId"
                             type="xsd:NMTOKEN"/>
              <xsd:attribute name="triggerClassType"
                             type="classTypeType"/>
            </xsd:extension>
          </xsd:complexContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:choice>
  </xsd:sequence>
    <xsd:attribute name="connector">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
        <xsd:enumeration value="and"/>
        <xsd:enumeration value="or"/>
        <xsd:enumeration value="nand"/>
        <xsd:enumeration value="nor"/>
        <xsd:enumeration value="xor"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="actionType">
<!-- represents actionset.Action -->
  <xsd:choice>
    <!-- generic instruction qualifier -->
    <xsd:element name="instruction" type="qualifierType"/>

    <!-- new component instruction -->
    <xsd:element name="newComponent">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:NMTOKEN"
                       use="required"/>
        <xsd:attribute name="className" type="xsd:NMTOKEN"
                       use="required"/>
        <xsd:attribute name="posx"
                       type="xsd:nonNegativeInteger"
                       use="required"/>
        <xsd:attribute name="posy"
```

```
                    type="xsd:nonNegativeInteger"
                    use="required"/>
  </xsd:complexType>
</xsd:element>

<!-- link instruction -->
<xsd:element name="link">
  <xsd:complexType>
    <xsd:attribute name="src" type="xsd:NMTOKEN"
                    use="required"/>
    <xsd:attribute name="dest" type="xsd:NMTOKEN"
                    use="required"/>
    <xsd:attribute name="name" type="xsd:NMTOKEN"
                    use="required"/>
  </xsd:complexType>
</xsd:element>

<!-- show dialogue instruction -->
<xsd:element name="showDialog">
  <xsd:complexType>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="info"/>
          <xsd:enumeration value="error"/>
          <xsd:enumeration value="warning"/>
          <xsd:enumeration value="plain"/>
          <xsd:enumeration value="other"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="text" type="xsd:string"
                    use="required"/>
  </xsd:complexType>
</xsd:element>

<!-- enable exception handling instruction -->
<xsd:element name="enableExceptions"/>

<!-- disable exception handling instruction -->
<xsd:element name="disableExceptions"/>

<!-- delete component instruction -->
```

```
<xsd:element name="deleteComponent">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NMTOKEN"/>
  </xsd:complexType>
</xsd:element>

<!-- jump instruction -->
<xsd:element name="jump">
  <xsd:complexType>
    <xsd:attribute name="label" type="xsd:NMTOKEN"
                   use="required"/>
  </xsd:complexType>
</xsd:element>

<!-- select tree tool instruxtion -->
<xsd:element name="selectTool">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NMTOKEN"
                   use="required"/>
  </xsd:complexType>
</xsd:element>

<!-- instruction for the visibility of help panes -->
<xsd:element name="visibilityHTMLPane">
  <xsd:complexType>
    <xsd:attribute name="tabFlag">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="help"/>
          <xsd:enumeration value="result"/>
          <xsd:enumeration value="helpdetail"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="visibility" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:element>

<!-- load descritption file instruction -->
<xsd:element name="loadDescriptionFile">
  <xsd:complexType>
    <xsd:attribute name="path" type="xsd:anyURI"/>
  </xsd:complexType>
```

```
</xsd:element>

<!-- set text in result pane instruction -->
<xsd:element name="setTextResultTab">
  <xsd:complexType>
    <xsd:attribute name="text" type="xsd:string"/>
    <xsd:attribute name="mode">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="add"/>
          <xsd:enumeration value="set"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>

<!-- exception check instruction -->
<xsd:element name="exceptionCheck">
  <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="excep" type="xsd:NMTOKEN"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="excepSet"  type="xsd:NMTOKEN"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="qualifier"
                   minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:NMTOKEN">
              <xsd:attribute name="mode">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="normal"/>
                    <xsd:enumeration value="trigger"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:attribute>
              <xsd:attribute name="type">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="class"/>
                    <xsd:enumeration value="object"/>
```

```
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:attribute>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- set image instruction -->
  <xsd:element name="setImageComponent">
    <xsd:complexType>
      <xsd:attribute name="path" type="xsd:anyURI"/>
      <xsd:attribute name="mode" default="instance">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="instance"/>
              <xsd:enumeration value="class"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="ID" type="xsd:NMTOKEN"
                     use="required"/>
    </xsd:complexType>
  </xsd:element>

  <!-- set description of the state button -->
  <xsd:element name="descriptionStateButton">
    <xsd:complexType>
      <xsd:attribute name="description" type="xsd:string"
                     use="required"/>
      <xsd:attribute name="tooltip" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

  <!-- set visibility of the state button -->
  <xsd:element name="visibilityStateButton">
    <xsd:complexType>
      <xsd:attribute name="visibility" type="xsd:boolean"/>
    </xsd:complexType>
  </xsd:element>
```

```
      <!-- set behaviour of the state button -->
      <xsd:element name="behaviorStateButton">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="excep" type="xsd:NMTOKEN"
                         minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="excepSet" type="xsd:NMTOKEN"
                         minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:NMTOKEN"/>
    <xsd:attribute name="maxCalls" type="xsd:int"/>
    <xsd:attribute name="minCalls" type="xsd:int"/>
  </xsd:complexType>

  <!-- define type for exceptions -->
  <xsd:complexType name="exceptionType">
    <xsd:complexContent>
      <xsd:extension base="actionType">
        <xsd:sequence>
          <xsd:element name="conditions"
                       type="conditionContainer"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

</xsd:schema>
```

E.2 runtime.xsd

The following XML schema describes the file format used in the runtime scripts.

```
<?xml version="1.0"?>

<!-- the schema for scripts -->
<xsd:schema
  targetNamespace="http://www.ibr.cs.tu-bs.de/~tklie/runtime"
  xmlns="http://www.ibr.cs.tu-bs.de/~tklie/runtime"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sim="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
  xml:lang="en"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- import the simtool schema -->
  <xsd:import
    namespace="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
    schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/simtool.xsd"/>

  <xsd:element name="runtime">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="exceptionSet" type="xsd:anyURI"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="block"
                     minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="action" type="sim:actionType"
                           minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="label" type="xsd:NMTOKEN"
                           use="required"/>
            <xsd:attribute name="delay" type="xsd:double"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:NMTOKEN"
                     use="required"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

E.3 exceptionSet.xsd

The following XML schema describes the file format used in the exception set files.

```
<?xml version="1.0"?>

<xsd:schema
  targetNamespace="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
  xmlns="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sim="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
  xml:lang="en"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- import the simtool schema -->
  <xsd:import
    namespace="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
    schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/simtool.xsd"/>

  <xsd:element name="exceptionSet">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="action" type="sim:exceptionType"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:NMTOKEN" use="required"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Appendix F

Files for the Sample Lesson

In this chapter, the files used in the sample lesson in Chapter 5 are presented.

F.1 The runtime script

```
<?xml version="1.0"?>
<rt:runtime xmlns:rt="http://www.ibr.cs.tu-bs.de/~tklie/runtime"
            xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/runtime
                                http://www.ibr.cs.tu-bs.de/~tklie/runtime.xsd"
            name="Lesson1">

<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set1.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set2.xml</rt:exceptionSet>

<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3SmallDetailedLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3MediumDetailedLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3HighDetailedLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3Finish.xml</rt:exceptionSet>

<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3SourceHighLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3SingleStationHighLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3DestinationHighLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3LinkHighLevel.xml</rt:exceptionSet>

<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3SourceMediumLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3SingleStationMediumLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3DestinationMediumLevel.xml</rt:exceptionSet>
<rt:exceptionSet>file:///d:/development/cschalong/simtool/doc/Set3LinkMediumLevel.xml</rt:exceptionSet>

<rt:block label="main">
    <rt:action>
        <visibilityHTMLPane tabFlag="help" visibility="true" />
    </rt:action>
    <rt:action>
        <newComponent name="Entrance" className="Source" posx="10" posy="180"/>
    </rt:action>
    <rt:action>
        <newComponent name="Exit" className="Destination" posx="550" posy="180"/>
    </rt:action>
    <rt:action>
        <newComponent name="FillingStation1" className="SingleStation" posx="150" posy="20"/>
```

```
</rt:action>
<rt:action>
    <newComponent name="FillingStation2" className="SingleStation" posx="200" posy="180"/>
</rt:action>
<rt:action>
    <newComponent name="FillingStation3" className="SingleStation" posx="150" posy="330"/>
</rt:action>
<rt:action>
    <newComponent name="QueueEntrance1" className="SingleStation" posx="300" posy="130"/>
</rt:action>
<rt:action>
    <newComponent name="QueueEntrance2" className="SingleStation" posx="300" posy="230"/>
</rt:action>
<rt:action>
    <newComponent name="CashBox1" className="SingleStation" posx="400" posy="130"/>
</rt:action>
<rt:action>
    <newComponent name="CashBox2" className="SingleStation" posx="400" posy="230"/>
</rt:action>
<rt:action>
    <newComponent name="DriveOut" className="Destination" posx="10" posy="30"/>
</rt:action>
<rt:action>
    <link src="Entrance" dest="FillingStation1" name="l1"/>
</rt:action>
<rt:action>
    <link src="Entrance" dest="FillingStation2" name="l2"/>
</rt:action>
<rt:action>
    <link src="Entrance" dest="FillingStation3" name="l3"/>
</rt:action>
<rt:action>
    <link src="Entrance" dest="DriveOut" name="l14"/>
</rt:action>
<rt:action>
    <link src="FillingStation1" dest="QueueEntrance1" name="l4"/>
</rt:action>
<rt:action>
    <link src="FillingStation1" dest="QueueEntrance2" name="l5"/>
</rt:action>
<rt:action>
    <link src="FillingStation2" dest="QueueEntrance1" name="l6"/>
</rt:action>
<rt:action>
    <link src="FillingStation2" dest="QueueEntrance2" name="l7"/>
</rt:action>
<rt:action>
    <link src="FillingStation3" dest="QueueEntrance1" name="l8"/>
</rt:action>
<rt:action>
    <link src="FillingStation3" dest="QueueEntrance2" name="l9"/>
</rt:action>
<rt:action>
    <link src="QueueEntrance1" dest="CashBox1" name="l10"/>
</rt:action>
<rt:action>
    <link src="QueueEntrance2" dest="CashBox2" name="l11"/>
</rt:action>
<rt:action>
    <link src="CashBox1" dest="Exit" name="l12"/>
</rt:action>
<rt:action>
    <link src="CashBox2" dest="Exit" name="l13"/>
</rt:action>

<rt:action>
    <setImageComponent path="images/gasStationfree.gif" mode="instance" ID="FillingStation1"/>
</rt:action>
<rt:action>
    <setImageComponent path="images/gasStationfree.gif" mode="instance" ID="FillingStation2"/>
</rt:action>
<rt:action>
    <setImageComponent path="images/gasStationfree.gif" mode="instance" ID="FillingStation3"/>
</rt:action>
<rt:action>
    <descriptionStateButton description="Finish" tooltip="This button will finish the lesson"/>
</rt:action>

<rt:action>
    <visibilityStateButton visibility="true" />
</rt:action>

<rt:action>
```

```
        <behaviorStateButton>
            <excep>Finish</excep>
            <excep>SmallLevelNotOk</excep>
            <excep>MediumLevelNotOk</excep>
            <excep>HighLevelNotOk</excep>
        </behaviorStateButton>
    </rt:action>

    <rt:action>
          <selectTool name="select" />
    </rt:action>

    <rt:action>
        <loadDescriptionFile path="lesson/task"/>
    </rt:action>

    <!-- set some wrong setting to unsettle the user -->
    <rt:action>
        <instruction srcId="FillingStation1" function="setRouting" classType="instance">
            <param name="params" type="vector">
                <value>3</value>
                <type>java.lang.Integer</type>
                <value>QueueEntrance2</value>
                <type>java.lang.String</type>
                <value>QueueEntrance1</value>
                <type>java.lang.String</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="FillingStation2" function="setRouting" classType="instance">
            <param name="params" type="vector">
                <value>1</value>
                <type>java.lang.Integer</type>
                <value>true</value>
                <type>java.lang.Boolean</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="FillingStation3" function="setRouting" classType="instance">
            <param name="params" type="vector">
                <value>3</value>
                <type>java.lang.Integer</type>
                <value>QueueEntrance2</value>
                <type>java.lang.String</type>
                <value>QueueEntrance1</value>
                <type>java.lang.String</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- WEEKEND BEGINS-->
<!-- ================================================================================== -->

<rt:block label="WeekendBegin">
    <rt:action>
        <disableExceptions/>
    </rt:action>
    <rt:action>
        <showDialog type="info" text="The weekend begins."/>
    </rt:action>

    <rt:action>
        <instruction srcId="l4" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="l6" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </param>
```

```
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="l8" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <instruction srcId="CashBox2" function="setPTDist" classType="instance">
            <param name="dist" type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
          <instruction srcId="CashBox2" function="setMRTDist" classType="instance">
            <param name="dist" type="stdType">
                 <value>60.0,300.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <setImageComponent path="images/singlestationoff.gif" mode="instance" ID="CashBox1"/>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ============================================================================== -->
<!-- WEEKEND ENDS-->
<!-- ============================================================================== -->

<rt:block label="WeekendEnd">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <showDialog type="info" text="The weekend ends."/>
    </rt:action>
    <rt:action>
        <instruction srcId="l4" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="l6" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>1</value>
                <type>java.lang.Long</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
        <instruction srcId="l8" function="setMaxME" classType="instance">
            <param type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <instruction srcId="CashBox2" function="setPTDist" classType="instance">
            <param name="dist" type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
        </instruction>
    </rt:action>
    <rt:action>
```

```
        <instruction srcId="CashBox2" function="setMRTDist" classType="instance">
            <param name="dist" type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
        </instruction>
   </rt:action>

    <rt:action>
        <setImageComponent ID="CashBox1" mode="instance"/>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- LESSER LEVEL NOT OK-->
<!-- ================================================================================== -->

<rt:block label="SmallLevelNotOk">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <showDialog type="info" text="The settings are not correct. Please retry!"/>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- Medium LEVEL NOT OK-->
<!-- ================================================================================== -->

<rt:block label="MediumLevelNotOk">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="false" />
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="helpdetail" visibility="false" />
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="true" />
    </rt:action>

    <rt:action>
        <showDialog type="info" text="Settings are still wrong! Using a medium feedback level."/>
    </rt:action>

    <rt:action>
        <setTextResultTab text="&lt;h3&gt;Following components have wrong settings.&lt;/h3&gt;"
                          mode="set"/>
    </rt:action>

    <rt:action>
        <exceptionCheck>
            <excepSet>Set3SourceMediumLevel</excepSet>
            <excepSet>Set3SingleStationMediumLevel</excepSet>
            <excepSet>Set3DestinationMediumLevel</excepSet>
            <excepSet>Set3LinkMediumLevel</excepSet>
        </exceptionCheck>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- HIGH LEVEL NOT OK-->
<!-- ================================================================================== -->
```

```
<rt:block label="HighLevelNotOk">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="false" />
    </rt:action>
    <rt:action>
        <visibilityHTMLPane tabFlag="helpdetail" visibility="false" />
    </rt:action>
    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="true" />
    </rt:action>

    <rt:action>
        <showDialog type="info" text="Settings are still wrong! Using a high feedback level."/>
    </rt:action>

    <rt:action>
        <setTextResultTab text="&lt;h3&gt;Properties resisting on the following tabs are not correct&lt;/h3&gt;"
                          mode="set"/>
    </rt:action>

    <rt:action>
        <exceptionCheck>
            <excepSet>Set3SourceHighLevel</excepSet>
            <excepSet>Set3SingleStationHighLevel</excepSet>
            <excepSet>Set3DestinationHighLevel</excepSet>
            <excepSet>Set3LinkHighLevel</excepSet>
        </exceptionCheck>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- FINISH-->
<!-- ================================================================================== -->

<rt:block label="Finish">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="false" />
    </rt:action>

    <rt:action>
        <showDialog type="info" text="Congratulation you solved the task. The model will be started now."/>
    </rt:action>

    <rt:action>
        <instruction srcId="gui.SimToolWin" function="runOrPause" classType="class">
        </instruction>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- SET ALL-->
<!-- ================================================================================== -->

<rt:block label="SetAll">
    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <visibilityHTMLPane tabFlag="result" visibility="false" />
    </rt:action>

    <rt:action>
        <showDialog type="info" text="System sets properties of the different components."/>
    </rt:action>

    <!-- RESULT SET-->
```

```
<!-- ENTRANCE -->
<rt:action>
    <instruction srcId="Entrance" function="setMEType" classType="instance">
        <param name="meType" type="stdType">
            <value>Car</value>
            <type>java.lang.String</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="Entrance" function="setCIDist" classType="instance">
        <param name="dist" type="stdType">
            <value>120.0</value>
            <type>distribution.DistributionExponential</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="Entrance" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="Entrance" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>0</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- EXIT -->
<rt:action>
    <instruction srcId="Exit" function="setDestrDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="Exit" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>

<!-- FS 1 -->
<rt:action>
    <instruction srcId="FillingStation1" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>100.0,350.0,150.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>

<rt:action>
    <instruction srcId="FillingStation1" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>5.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>

<!-- FS 2 -->
<rt:action>
    <instruction srcId="FillingStation2" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>100.0,350.0,150.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="FillingStation2" function="setMRTDist" classType="instance">
```

```
        <param name="dist" type="stdType">
            <value>5.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>

<!-- FS 3 -->
<rt:action>
    <instruction srcId="FillingStation3" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>150.0,525.0,225.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="FillingStation3" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>5.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>

<!-- QE 1 -->
<rt:action>
    <instruction srcId="QueueEntrance1" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="QueueEntrance1" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
            </param>
    </instruction>
</rt:action>

<!-- QE 2 -->
<rt:action>
    <instruction srcId="QueueEntrance2" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="QueueEntrance2" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>0.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>

<!-- CashBox 1 -->
<rt:action>
    <instruction srcId="CashBox1" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>30.0,90.0,60.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="CashBox1" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>10.0,300.0,20.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>

<!-- CashBox 2 -->
<rt:action>
    <instruction srcId="CashBox2" function="setPTDist" classType="instance">
        <param name="dist" type="stdType">
```

```
            <value>30.0,90.0,60.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="CashBox2" function="setMRTDist" classType="instance">
        <param name="dist" type="stdType">
            <value>10.0,300.0,20.0</value>
            <type>distribution.DistributionTriangle</type>
        </param>
    </instruction>
</rt:action>

<!-- L 1 -->
<rt:action>
    <instruction srcId="l1" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l1" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 2 -->
<rt:action>
    <instruction srcId="l2" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l2" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 3 -->
<rt:action>
    <instruction srcId="l3" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l3" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 4 -->
<rt:action>
    <instruction srcId="l4" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l4" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
```

```
    </instruction>
</rt:action>


<!-- L 5 -->
<rt:action>
    <instruction srcId="l5" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0,8.0</value>
            <type>distribution.DistributionUniform</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l5" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 6 -->
<rt:action>
    <instruction srcId="l6" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>2.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l6" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 7 -->
<rt:action>
    <instruction srcId="l7" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>2.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l7" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 8 -->
<rt:action>
    <instruction srcId="l8" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0,8.0</value>
            <type>distribution.DistributionUniform</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l8" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>


<!-- L 9 -->
<rt:action>
    <instruction srcId="l9" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>4.0</value>
            <type>distribution.DistributionConst</type>
```

```
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l9" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 10 -->
<rt:action>
    <instruction srcId="l10" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>2.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l10" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 11 -->
<rt:action>
    <instruction srcId="l11" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>2.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l11" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 12 -->
<rt:action>
    <instruction srcId="l12" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>3.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l12" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- L 13 -->
<rt:action>
    <instruction srcId="l13" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>3.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l13" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>
```

```
<!-- L 14 -->
<rt:action>
    <instruction srcId="l14" function="setTransportTimeDist" classType="instance">
        <param name="dist" type="stdType">
            <value>2.0</value>
            <type>distribution.DistributionConst</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="l14" function="setMaxME" classType="instance">
        <param name="maxME" type="stdType">
            <value>5</value>
            <type>java.lang.Long</type>
        </param>
    </instruction>
</rt:action>

<!-- Routing -->
<rt:action>
    <instruction srcId="Entrance" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="FillingStation1" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="FillingStation2" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="FillingStation3" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="QueueEntrance1" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="QueueEntrance2" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="CashBox1" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
</rt:action>
<rt:action>
    <instruction srcId="CashBox2" function="setRouting" classType="instance">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
    </instruction>
```

```
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

<!-- ============================================================================================ -->
<!-- SET ENTRANCE AT BEGIN OF DAY-->
<!-- ============================================================================================ -->

<rt:block label="BeginOfDay">
    <rt:action>
        <showDialog type="info" text="Begin of day."/>
    </rt:action>
</rt:block>

<!-- ============================================================================================ -->
<!-- SET ENTRANCE AT END OF DAY-->
<!-- ============================================================================================ -->

<rt:block label="EndOfDay">
    <rt:action>
        <showDialog type="info" text="End of day."/>
    </rt:action>

    <rt:action>
        <instruction srcId="Entrance" function="setCIDist" classType="instance">
            <param name="dist" type="stdType">
                <value>57600.0</value>
                <type>distribution.DistributionConst</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <instruction srcId="Entrance" function="setMRTDist" classType="instance">
            <param name="dist" type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
        </instruction>
    </rt:action>
</rt:block>

<!-- ============================================================================================ -->
<!-- RESET ENTRANCE AT END OF DAY-->
<!-- ============================================================================================ -->

<rt:block label="EndOfDay2">
    <rt:action>
        <instruction srcId="Entrance" function="setCIDist" classType="instance">
            <param name="dist" type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
            </param>
        </instruction>
    </rt:action>

    <rt:action>
        <instruction srcId="Entrance" function="setMRTDist" classType="instance">
            <param name="dist" type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
        </instruction>
    </rt:action>
</rt:block>

<!-- ============================================================================================ -->
<!-- AFTER SIMULATION OK-->
<!-- ============================================================================================ -->

<rt:block label="EndOfSimulationOk">
    <rt:action>
        <instruction srcId="gui.SimToolWin" function="runOrPause" classType="class">
        </instruction>
    </rt:action>

    <rt:action>
        <disableExceptions/>
    </rt:action>
```

```
    <rt:action>
        <showDialog
          type="info"
          text="Your settings are correct. Please refer to the statistic informations for further informations."/>
    </rt:action>
</rt:block>

<!-- ================================================================================== -->
<!-- AFTER SIMULATION WRONG-->
<!-- ================================================================================== -->

<rt:block label="EndOfSimulationWrong">
    <rt:action>
        <instruction srcId="gui.SimToolWin" function="runOrPause" classType="class">
        </instruction>
    </rt:action>

    <rt:action>
        <disableExceptions/>
    </rt:action>

    <rt:action>
        <showDialog type="info" text="Sorry your settings were not correct. Please reconfigure the simulation model."/>
    </rt:action>

    <rt:action>
        <showDialog type="info" text="Please reset settings before you are going to start the simulation again."/>
    </rt:action>

    <rt:action>
        <enableExceptions/>
    </rt:action>
</rt:block>

</rt:runtime>
```

F.2 Exception sets

F.2.1 Set 1

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                   http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set1">

<es:action name="WeekendBegin" maxCalls="1">
    <jump label="WeekendBegin"/>
    <conditions>
        <condition srcId="FillingStation1" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>1:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="WeekendEnd" maxCalls="1">
    <jump label="WeekendEnd"/>
    <conditions>
        <condition srcId="FillingStation1" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>3:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
```

```
    </conditions>
</es:action>

<!-- ================================================================================== -->
<!-- END OF ALL-->
<!-- ================================================================================== -->

<es:action name="EndofSimulationOk" maxCalls="1">
    <jump label="EndOfSimulationOk"/>
    <conditions connector="and">
        <condition srcId="Exit" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>4:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
        <conditions connector="and">
            <condition srcId="Exit" function="getThroughPut" classType="instance" condType="componentStatisticState">
                <returnValue compareOperator="&gt;=" type="stdType">
                    <value>640</value>
                    <type>java.lang.Double</type>
                </returnValue>
            </condition>
            <condition srcId="Exit" function="getThroughPut" classType="instance" condType="componentStatisticState">
                <returnValue compareOperator="&lt;=" type="stdType">
                    <value>675</value>
                    <type>java.lang.Double</type>
                </returnValue>
            </condition>
        </conditions>
    </conditions>
</es:action>

<es:action name="EndofSimulationWrong" maxCalls="1">
    <jump label="EndOfSimulationWrong"/>
    <conditions connector="and">
        <condition srcId="Exit" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;" type="stdType">
                <value>4:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
        <conditions connector="or">
            <condition srcId="Exit" function="getThroughPut" classType="instance" condType="componentStatisticState">
                <returnValue compareOperator="&lt;=" type="stdType">
                    <value>640</value>
                    <type>java.lang.Double</type>
                </returnValue>
            </condition>
            <condition srcId="Exit" function="getThroughPut" classType="instance" condType="componentStatisticState">
                <returnValue compareOperator="&gt;=" type="stdType">
                    <value>675</value>
                    <type>java.lang.Double</type>
                </returnValue>
            </condition>
        </conditions>
    </conditions>
</es:action>

<!-- ================================================================================== -->
<!-- THE ENTRANCE-->
<!-- ================================================================================== -->

<es:action name="BeginOfDay1" maxCalls="1">
    <jump label="BeginOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>0:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay1" maxCalls="1">
    <jump label="EndOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
```

```
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>0:08:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay1Reset" maxCalls="1">
    <jump label="EndOfDay2"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>0:09:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="BeginOfDay2" maxCalls="1">
    <jump label="BeginOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>1:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay2" maxCalls="1">
    <jump label="EndOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>1:08:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay2Reset" maxCalls="1">
    <jump label="EndOfDay2"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>1:09:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="BeginOfDay3" maxCalls="1">
    <jump label="BeginOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>2:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay3" maxCalls="1">
    <jump label="EndOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>2:08:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
```

```
    </conditions>
</es:action>

<es:action name="EndOfDay3Reset" maxCalls="1">
    <jump label="EndOfDay2"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>2:09:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="BeginOfDay4" maxCalls="1">
    <jump label="BeginOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>3:00:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay4" maxCalls="1">
    <jump label="EndOfDay"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>3:08:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="EndOfDay4Reset" maxCalls="1">
    <jump label="EndOfDay2"/>
    <conditions>
        <condition srcId="Entrance" function="getSimulationTime" classType="instance" condType="simulationRun"
                   triggerId="gui.SimToolWin" triggerClassType="class">
            <returnValue compareOperator="&gt;=" type="stdType">
                <value>3:09:00:00.000</value>
                <type>event.Time</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.2 Set 2

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set2">


<!-- ============================================================================================ -->
<!-- ENTRANCE-->
<!-- ============================================================================================ -->

<es:action name="checkEntranceSettings">
    <showDialog type="info" text="The source produces too few elements."/>
    <conditions connector="and">
        <condition srcId="Entrance" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;" type="stdType">
```

```
                <value>900</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&gt;" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<!-- ================================================================================== -->
<!-- LINKS-->
<!-- ================================================================================== -->

<es:action name="checkLinkSettings">
    <showDialog type="info" text="Setting the max elements attribute of the link to 0 will disable the link."/>
    <conditions connector="or">
        <condition srcId="l1" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l2" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l3" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l4" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l5" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l6" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l7" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l8" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l9" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
```

```
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="componentProperty">
            <returnValue compareOperator="&lt;=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.3 Set 3 Finish

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3Finish">

<!-- ================================================================================== -->
<!-- FINISH -->
<!-- ================================================================================== -->

<es:action name="Finish">
    <jump label="Finish" />
    <conditions connector="and">

        <!-- ================================================================================== -->
        <!-- ENTRANCE -->
        <!-- ================================================================================== -->

        <conditions connector="or">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="&gt;=" type="stdType">
              <value>1000</value>
              <type>java.lang.Long</type>
            </returnValue>
          </condition>
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
              <value>0</value>
              <type>java.lang.Long</type>
            </returnValue>
          </condition>
        </conditions>
        <condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compCIDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compRouting" classType="instance" condType="guiStatebutton">
```

```
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLING STATION 1 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>100.0,350.0,150.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation1" function="compRouting" classType="instance"  condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 2 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>100.0,350.0,150.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
```

```
<!-- FILLING STATION 3 -->
<!-- ============================================================================================ -->

<condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance"
   condType="guiStatebutton">
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>150.0,525.0,225.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation3" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- QUEUE ENTRANCE 1 -->
<!-- ============================================================================================ -->

<condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- QUEUE ENTRANCE 2 -->
<!-- ============================================================================================ -->

<condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
```

```
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 1 -->
<!-- ================================================================================== -->

<condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>10.0,300.0,20.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>30.0,90.0,60.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox1" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 2 -->
<!-- ================================================================================== -->

<condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>10.0,300.0,20.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>30.0,90.0,60.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
```

```
</condition>
<condition srcId="CashBox2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- LINKS -->
<!-- ================================================================================== -->

<condition srcId="l1" function="getMaxME" classType="instance"  condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l2" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l3" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l4" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l5" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
```

```
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0,8.0</value>
        <type>distribution.DistributionUniform</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l6" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l7" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l8" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0,8.0</value>
        <type>distribution.DistributionUniform</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l9" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="==" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l10" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="==" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
```

```
        <condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>1</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="==" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="&gt;" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<!-- ================================================================================== -->
<!-- NOT FINISH -->
<!-- ================================================================================== -->

<es:action name="NotFinish" minCalls="9" maxCalls="10">
    <jump label="SetAll" />
    <conditions connector="or">

        <!-- ================================================================================== -->
        <!-- ENTRANCE -->
        <!-- ================================================================================== -->

        <conditions connector="and">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
             <returnValue compareOperator="&lt;" type="stdType">
                <value>1000</value>
```

```
        <type>java.lang.Long</type>
    </returnValue>
  </condition>
  <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
     <returnValue compareOperator="!=" type="stdType">
        <value>0</value>
        <type>java.lang.Long</type>
    </returnValue>
  </condition>
</conditions>
<condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="Entrance" function="compCIDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>120.0</value>
        <type>distribution.DistributionExponential</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="Entrance" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 1 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>100.0,350.0,150.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation1" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 2 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
```

```
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>100.0,350.0,150.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
============================================================================================ -->
<!-- FILLINGSTATION 3 -->
<!-- ============================================================================================ -->
<condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>5.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>150.0,525.0,225.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="FillingStation3" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- QUEUE ENTRANCE 1 -->
<!-- ============================================================================================ -->

<condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
```

```
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 2 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>0.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 1 -->
<!-- ================================================================================== -->

<condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>10.0,300.0,20.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>30.0,90.0,60.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox1" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
```

```
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 2 -->
<!-- ================================================================================== -->

<condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>10.0,300.0,20.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>30.0,90.0,60.0</value>
        <type>distribution.DistributionTriangle</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="CashBox2" function="compRouting" classType="instance" condType="guiStatebutton">
    <param name="params" type="vector">
        <value>0</value>
        <type>java.lang.Integer</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- LINKS -->
<!-- ================================================================================== -->

<condition srcId="l1" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l2" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l3" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
```

```
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l4" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l5" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0,8.0</value>
        <type>distribution.DistributionUniform</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l6" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l7" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l8" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0,8.0</value>
        <type>distribution.DistributionUniform</type>
```

```
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l9" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>4.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l10" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l11" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>1</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>2.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l12" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>3.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
        <type>java.lang.Boolean</type>
    </returnValue>
</condition>
<condition srcId="l13" function="getMaxME" classType="instance" condType="guiStatebutton">
    <returnValue compareOperator="!=" type="stdType">
        <value>2</value>
        <type>java.lang.Long</type>
    </returnValue>
</condition>
<condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
    <param type="stdType">
        <value>3.0</value>
        <type>distribution.DistributionConst</type>
    </param>
    <returnValue compareOperator="!=" type="stdType">
        <value>true</value>
```

```
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.4 Set 3 Small Detail Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3SmallDetailedLevel">

<es:action name="SmallLevelNotOk" maxCalls="2">
    <jump label="SmallLevelNotOk" />
    <conditions connector="or">

        <!-- ================================================================================== -->
        <!-- ENTRANCE -->
        <!-- ================================================================================== -->
        <conditions connector="and">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
                <returnValue compareOperator="&lt;" type="stdType">
                    <value>1000</value>
                    <type>java.lang.Long</type>
                </returnValue>
          </condition>
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
                <returnValue compareOperator="!=" type="stdType">
                    <value>0</value>
                    <type>java.lang.Long</type>
                </returnValue>
          </condition>
        </conditions>
        <condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
                <param type="stdType">
                        <value>0.0</value>
                        <type>distribution.DistributionConst</type>
                </param>
                <returnValue compareOperator="!=" type="stdType">
                        <value>true</value>
                        <type>java.lang.Boolean</type>
                </returnValue>
        </condition>
        <condition srcId="Entrance" function="compCIDist" classType="instance" condType="guiStatebutton">
                <param type="stdType">
                        <value>120.0</value>
                        <type>distribution.DistributionExponential</type>
                </param>
                <returnValue compareOperator="!=" type="stdType">
                        <value>true</value>
                        <type>java.lang.Boolean</type>
                </returnValue>
        </condition>
        <condition srcId="Entrance" function="compRouting" classType="instance" condType="guiStatebutton">
                <param name="params" type="vector">
                    <value>0</value>
                    <type>java.lang.Integer</type>
                </param>
                <returnValue compareOperator="!=" type="stdType">
                        <value>true</value>
                        <type>java.lang.Boolean</type>
                </returnValue>
        </condition>

        <!-- ================================================================================== -->
```

```
<!-- FILLING STATION 1 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 2 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compProcTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 3 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
```

```
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>150.0,525.0,225.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 1 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 2 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
```

```
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 1 -->
<!-- ================================================================================== -->

<condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 2 -->
<!-- ================================================================================== -->

<condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
```

```
<!-- ============================================================================================ -->
<!-- LINKS -->
<!-- ============================================================================================ -->

<condition srcId="l1" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l2" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l3" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l4" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l5" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
        </returnValue>
</condition>
<condition srcId="l6" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l7" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l8" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l9" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l10" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l11" function="getMaxME" classType="instance" condType="guiStatebutton">
```

```
            <returnValue compareOperator="!=" type="stdType">
                <value>1</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                    <value>2.0</value>
                    <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                    <value>true</value>
                    <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                    <value>3.0</value>
                    <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                    <value>true</value>
                    <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                    <value>3.0</value>
                    <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                    <value>true</value>
                    <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.5 Set 3 Medium Detail Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3MediumDetailedLevel">

<es:action name="MediumLevelNotOk" minCalls="3" maxCalls="5">
    <jump label="MediumLevelNotOk" />
    <conditions connector="or">

        <!-- ================================================================================== -->
        <!-- ENTRANCE -->
        <!-- ================================================================================== -->
        <conditions connector="and">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
```

```
        <returnValue compareOperator="&lt;" type="stdType">
            <value>1000</value>
            <type>java.lang.Long</type>
        </returnValue>
  </condition>
  <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>0</value>
            <type>java.lang.Long</type>
        </returnValue>
  </condition>
</conditions>
<condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="Entrance" function="compCIDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="Entrance" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLING STATION 1 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 2 -->
<!-- ================================================================================== -->
```

```
<condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- FILLINGSTATION 3 -->
<!-- ================================================================================== -->

<condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>150.0,525.0,225.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 1 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
```

```
<condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 2 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 1 -->
<!-- ================================================================================== -->

<condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
```

```
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 2 -->
<!-- ================================================================================== -->

<condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- LINKS -->
<!-- ================================================================================== -->

<condition srcId="l1" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l2" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l3" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
```

```
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l4" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l5" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l6" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l7" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l8" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
```

```
<condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l9" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l10" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l11" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l12" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l13" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>3.0</value>
```

```
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.6 Set 3 High Detail Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3HighDetailedLevel">

<es:action name="HighLevelNotOk" minCalls="6" maxCalls="8">
    <jump label="HighLevelNotOk" />
    <conditions connector="or">

        <!-- ================================================================================== -->
        <!-- ENTRANCE -->
        <!-- ================================================================================== -->
        <conditions connector="and">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="&lt;" type="stdType">
                <value>1000</value>
                <type>java.lang.Long</type>
            </returnValue>
          </condition>
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
          </condition>
        </conditions>
        <condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compCIDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compRouting" classType="instance" condType="guiStatebutton">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
        </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- FILLING STATION 1 -->
<!-- ============================================================================================ -->

<condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- FILLINGSTATION 2 -->
<!-- ============================================================================================ -->

<condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ============================================================================================ -->
<!-- FILLINGSTATION 3 -->
<!-- ============================================================================================ -->

<condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>5.0</value>
```

```
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>150.0,525.0,225.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="FillingStation3" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 1 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance"
          condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- QUEUE ENTRANCE 2 -->
<!-- ================================================================================== -->

<condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance"
           condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
```

```
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 1 -->
<!-- ================================================================================== -->

<condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox1" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- CASH BOX 2 -->
<!-- ================================================================================== -->

<condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="CashBox2" function="compRouting" classType="instance" condType="guiStatebutton">
        <param name="params" type="vector">
            <value>0</value>
            <type>java.lang.Integer</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
```

```
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>

<!-- ================================================================================== -->
<!-- LINKS -->
<!-- ================================================================================== -->

<condition srcId="l1" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l2" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l3" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l4" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l5" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
```

```
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l6" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l7" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l8" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0,8.0</value>
                <type>distribution.DistributionUniform</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l9" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>2</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>4.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
        </returnValue>
</condition>
<condition srcId="l10" function="getMaxME" classType="instance" condType="guiStatebutton">
        <returnValue compareOperator="!=" type="stdType">
            <value>1</value>
            <type>java.lang.Long</type>
        </returnValue>
</condition>
<condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
        <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
        </param>
        <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
            </returnValue>
        </condition>
        <condition srcId="l11" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>1</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>2.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="guiStatebutton">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="l14" function="getMaxME" classType="instance" condType="guiStatebutton">
            <returnValue compareOperator="==" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.7 Set 3 Link Medium Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3LinkMediumLevel">

<es:action name="L1Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l1&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l1" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
```

```
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L2Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l2&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l2" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L3Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l3&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l3" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L4Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l4&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l4" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L5Wrong">
```

```
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l5&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l5" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L6Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l6&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l6" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l6" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L7Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l7&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l7" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L8Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l8&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l8" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
```

```
        </condition>
    </conditions>
</es:action>

<es:action name="L9Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l9&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l9" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L10Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l10&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l10" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L11Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l11&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l11" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L12Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l12&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l12" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
```

```
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L13Wrong">
    <setTextResultTab text="&lt;A HREF='tool/link'&gt;Link l13&lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l13" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.8 Set 3 Link High Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3LinkHighLevel">

<es:action name="L1RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l1 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l1" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l1" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L2RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l2 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l2" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l2" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
```

```
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L3RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l3 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l3" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l3" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L4RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l4 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l4" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l4" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L5RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l5 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l5" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l5" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L6RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l6 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l6" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
```

```
            </returnValue>
        </condition>
        <condition srcId="16" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L7RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l7 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l7" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l7" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L8RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l8 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l8" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l8" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L9RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l9 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l9" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l9" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L10RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l10 &lt;/A&gt;" mode="add"/>
```

```
    <conditions connector="or">
        <condition srcId="l10" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l10" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L11RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l11 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l11" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l11" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L12RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l12 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l12" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>4</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l12" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="L13RunWrong">
    <setTextResultTab text="&lt;A HREF='tool/link/runTab'&gt;Run tab of the link l13 &lt;/A&gt;" mode="add"/>
    <conditions connector="or">
        <condition srcId="l13" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>2</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="l13" function="compTransportTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>3.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
```

```
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.9 Set 3 Source Medium Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3SourceMediumLevel">

<es:action name="SourceWrong">
    <setTextResultTab text="&lt;A HREF='component/simulation.ComSource' &gt;Entrance&lt;/A&gt;" mode="add" />
    <conditions connector="or">
        <conditions connector="and">
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="&lt;" type="stdType">
                <value>1000</value>
                <type>java.lang.Long</type>
            </returnValue>
          </condition>
          <condition srcId="Entrance" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
          </condition>
        </conditions>
        <condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compCIDist" classType="instance" condType="script">
            <param type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.10 Set 3 Source High Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
```

```
                xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                 http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                id="Set3SourceHighLevel">

<es:action name="SourceStandardWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSource/standardTab' &gt;Standard tab of the source&lt;/A&gt;"
      mode="add" />
    <conditions connector="and">
        <condition srcId="Entrance" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="&lt;" type="stdType">
                <value>1000</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="getMaxME" classType="instance" condType="script">
            <returnValue compareOperator="!=" type="stdType">
                <value>0</value>
                <type>java.lang.Long</type>
            </returnValue>
        </condition>
    </conditions>
    </conditions>
</es:action>

<es:action name="SourceRunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSource/runTab'&gt;Run tab of the source&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="Entrance" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Entrance" function="compCIDist" classType="instance" condType="script">
            <param type="stdType">
                <value>120.0</value>
                <type>distribution.DistributionExponential</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="SourceRoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSource/routingTab'&gt;Routing tab of the source&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="Entrance" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.11 Set 3 Destination Medium Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3DestinationMediumLevel">

<es:action name="DestinationWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComDestination'&gt;Exit&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="Exit" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Exit" function="compDestroyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
\end{verabtim}
\end{tiny}

\subsection{Set 3 Destination High Level}
\label{sec:set3dhl}
\begin{tiny}
\begin{verbatim}
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3DestinationHighLevel">

<es:action name="DestinationRunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComDestination/runTab'&gt;Run tab of the destination &lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="Exit" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="Exit" function="compDestroyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.12 Set 3 Single Station Medium Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3SingleStationMediumLevel">

<es:action name="FillingStation1Wrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Filling station 1&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation2Wrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Filling station 2&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation3Wrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Filling station 3&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>150.0,525.0,225.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation3" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance1Wrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Queue entrance 1&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance2Wrong">
    <setTextResultTab
```

```
      text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Queue entrance 2&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox1Wrong">
    <setTextResultTab text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Cash box 1&lt;/A&gt;" mode="add" />
    <conditions connector="or">
        <condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="CashBox1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox2Wrong">
    <setTextResultTab text="&lt;A HREF='component/simulation.ComSingleStation'&gt;Cash box 2&lt;/A&gt;" mode="add" />
    <conditions connector="or">
        <condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
            </returnValue>
        </condition>
        <condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="CashBox2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.2.13 Set 3 Single Station High Level

```
<?xml version="1.0"?>
<es:exceptionSet xmlns:es="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet"
                 xmlns="http://www.ibr.cs.tu-bs.de/~tklie/simtool"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet
                  http://www.ibr.cs.tu-bs.de/~tklie/exceptionSet.xsd"
                 id="Set3SingleStationHighLevel">

<es:action name="FillingStation1RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the filling station 1&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation1RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/routingTab'&gt;Routing tab of the filling station 1&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="FillingStation1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
```

```
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation2RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the filling station 2&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>100.0,350.0,150.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation2RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/routingTab'&gt;Routing tab of the filling station 2&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="FillingStation2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation3RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the filling station 3&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="FillingStation3" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>5.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="FillingStation3" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>150.0,525.0,225.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="FillingStation3RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/rouitingTab'&gt;Routing tab of the filling station 3&lt;/A&gt;"
```

```
      mode="add" />
    <conditions>
        <condition srcId="FillingStation3" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance1RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the queue entrance 1&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="QueueEntrance1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance1RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/rouitingTab'&gt;Routing tab of the queue entrance 1&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="QueueEntrance1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance2RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the queue entrance 2&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="QueueEntrance2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="QueueEntrance2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>0.0</value>
                <type>distribution.DistributionConst</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
```

```
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="QueueEntrance2RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/rouitingTab'&gt;Routing tab of the queue entrance 2&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="QueueEntrance2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox1RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the cash box 1&lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="CashBox1" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
        <condition srcId="CashBox1" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox1RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/routingTab'&gt;Routing tab of the cash box 1&lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="CashBox1" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox2RunWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/runTab'&gt;Run tab of the cash box 2 &lt;/A&gt;"
      mode="add" />
    <conditions connector="or">
        <condition srcId="CashBox2" function="compMakeReadyTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>30.0,90.0,60.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
```

```
            </returnValue>
        </condition>
        <condition srcId="CashBox2" function="compProcTimeDist" classType="instance" condType="script">
            <param type="stdType">
                <value>10.0,300.0,20.0</value>
                <type>distribution.DistributionTriangle</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>

<es:action name="CashBox2RoutingWrong">
    <setTextResultTab
      text="&lt;A HREF='component/simulation.ComSingleStation/routingTab'&gt;Routing tab of the cash box 2 &lt;/A&gt;"
      mode="add" />
    <conditions>
        <condition srcId="CashBox2" function="compRouting" classType="instance" condType="script">
            <param name="params" type="vector">
                <value>0</value>
                <type>java.lang.Integer</type>
            </param>
            <returnValue compareOperator="!=" type="stdType">
                <value>true</value>
                <type>java.lang.Boolean</type>
            </returnValue>
        </condition>
    </conditions>
</es:action>
</es:exceptionSet>
```

F.3 Lesson file

```
# Verteilungen
distributions = Const,Normal,Uniform,Triangle,Exponential

# Nachfolgerwahl
routing = Fixed,WeightedRouting,RoundRobin,Best

# Bausteine
components = Source,SingleStation,Destination

# Bewegliche Elemente
movables = Car,BoxElement

# Werkzeuge im Baum
tools = Select

# Menues
menuFile = Save,SaveAs,Exit,New
menuHelp = About
menuDebug = EH,ObjectList
```

```
menuView = Comment,Toolbar,Zel,Grid,GridOptions

# Buttons auf der Toolbar
buttons = Save,Start,Init,Simspeed

# Locale
language = en
country = GB

################################
# Bausteinspezifische Optionen #
################################

# Quelle
Source.popup = Properties,Statistic
Source.props.general = id,createelements,maxelements
Source.props.run = mkreadytimes,createInt
Source.props.routing = routing

# Tankstelle
SingleStation.popup = Properties,Statistic
SingleStation.props.general = id
SingleStation.props.run = mkreadytimes,prctime
SingleStation.props.routing = routing

# Senke
Destination.popup = Properties,Statistic
Destination.props.general = id
Destination.props.run = mkreadytimes,destdel

# Kante
Link.popup = Properties,Statistic
Link.props.general = id
Link.props.run = trtime,maxelements
```

Appendix G

Default Settings

G.1 Default lesson file

The following lesson file is used as the default lesson which will be loaded if no lesson file is specified.

```
################## # Global options # ##################

# enabled distributions distributions=Const,Normal,Uniform,
Triangle,Poisson,Exponential

# enabled routing algorithms
routing=Fixed,WeightedRouting,RoundRobin,Best

# enabled simulation components
components=Source,SingleStation,AssemblyStation,Destination

# enabled movable elements
movables=BoxElement,PalletElement,AssemblyElement,Car

# enabled tool in the tool tree
tools=Select,Link,Erase

# enabled menu items
```

```
menuFile=New,Save,SaveAs,Exit
menuRun=Start,Init,Step
menuView=Comment,Tree,Toolbar,Zel,Grid,GridOptions
menuDebug=EH,ObjectList
menuHelp=About

# enabled buttons in the tool bar
buttons=Open,Save,Start,Init,Step,Simspeed,Zel

# Locale: British English
language = en
country = GB

########Component specific options ####################

# options for source component
Source.popup=Statistic,Delete,Move,Link,Properties
Source.props.general=name,id,createelements,maxelements
Source.props.run=mkreadytimes,createInt
Source.props.routing=routing
Source.routing=RoundRobin

# options for single station
SingleStation.popup=Statistic,Delete,Move,Link,Properties
SingleStation.props.general=name,id
SingleStation.props.run=mkreadytimes,prctime
SingleStation.props.routing=routing

# options for assembly station
AssemblyStation.popup=Statistic,Delete,Move,Link,Properties
AssemblyStation.props.general=name,elementcount,id
AssemblyStation.props.run=mkreadytimes,prctime
AssemblyStation.props.routing=routing
AssemblyStation.routing=Best,WeightedRouting

# options for destination
Destination.popup=Statistic,Delete,Move,Properties
Destination.props.general=name,id
Destination.props.run=mkreadytimes,destdel

# link options
```

```
Link.popup=Statistic,Delete,Properties
Link.props.general=name
Link.props.run=trtime,maxelements
```

G.2 Default values

The following table shows the pre settings of the properties of the components.

property	value	description
general tab		
id		will be set by the component
name		at startup this value is the same as the id
element chooser		the first element specified in the lesson file
capacity limit	0	0 means that a capacity limit is not used
quantities	1 of each	one element of each movable element specified in the lesson file
run tab		
create intervall	constant (1.0)	constant distribution with 1.0
make ready time	constant (0.0)	constant distribution which is not used
processing time	constant (1.0)	constant distribution with 1.0
delay of destruction	constant (0.0)	constant distribution which is not used
transport time	constant (1.0)	constant distribution with 1.0
capacity limit	3	3 movable elements are allowed on a link
routing tab		
best alternative		

The default values of the different distributions are shown in the table below.

distribution	values
constant	1.0
normal	expected 1.0, deviation 0.3
uniform	lower 0.5, upper 3.0
triangle	lower 0.5, upper 3.0, expected 1.0
Poisson	expected 1.0
exponential	expected 1.0

Appendix H

SimTool CD

H.1 Contents of the CD

The CD that accompany this diploma thesis has the following contents:

`bin/`			
	`Linux/`		
		`browser/`	SimTool-compatible web browser
		`jre/`	Java Runtime Environment
	`Windows/`		
		`browser/`	SimTool-compatible web browser
		`install/`	Nullsoft Installer
		`jre/`	Java Runtime Environment
`classes`			compiled Java classes of SimTool
`doc/`			
	`javadoc/`		
		`public/`	generated javadoc documentation
		`private/`	javadoc (incl. private methods)
	`papers/`		some references of the thesis
	`thesis/`		
		`TeX/`	LaTeX sources of this diploma thesis
		`thesisKS.pdf`	the thesis as a pdf document
`src/`			
	`installer/`		source files of the install programme
	`simtool/`		source packages of SimTool
	`lessons/`		files of the sample lesson
`readme.html`			readme file of the CD (HTML)
`readme.txt`			readme file of the CD (text)
`setup.exe`			installation programme for Windows

H.2 Installation

On Windows systems, SimTool can be installed by using an installer `setup.exe`. For information about the usage of this installer and the installation on other systems refer to `readme.txt, readme.html` located at the root directory of the SimTool CD.

Index

Bibliography

[AAS+93] Rita L. Atkinson, Richard C. Atkinson, Edward E. Smith, Daryl J. Bem, and Susan Nolen-Hoeksema. *Introduction to Psychology*. Hartcourt Brace Jovanovich College Publishers, Fort Worth, 11th edition, 1993.

[AI 01] AI Research Group, University of Koblenz. *Protein – A PROver with a Theory Extension INterface*, 2001. http://www.uni-koblenz.de/ag-ki/Implementierungen/Protein/index.html (12/08/2002).

[AMT02] AMT Learning Solutions, Inc. *CBIquick*, 2002. http://www.amtcorp.com/ (12/08/2002).

[Apa02] The Apache XML Project. *Xerces2 Java Parser Readme*, 2002. http://xml.apache.org/xerces2-j/index.html (12/01/2002).

[Arm02] Eric Armstrong. Java api for xml processing. *The Java Web Services Tutorial*, 2002. http://java.sun.com/webservices/docs/1.0/tutorial/doc/JAXPIntro/ (12/01/2002).

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, Reading, Massachusets, 1986.

[Aus02] Austrian Web Application Center, Institute for Information processing and Computer supported new Media. *GentleWBT*, 2002. http://wbt-3.iicm.edu/ (12/08/2002).

[BB02] Sandra Bangewitz and Manuela Bernhardt. *Erweiterung Simulationstool*. TU Braunschweig – Institut für Wirtschaftswis-

senschaften – Abteilung Allgemeine BWL, Informationsmanagenent und Wirtschaftsinformatik, June 2002. Seminar work.

[Beh01] Henning Behme. Angelpunkt – Wo die Extensible Markup Language derzeit steht. *iX – Magazin für professionelle Informationstechnik*, (6):52 – 54, Juni 2001.

[Bun02] Bundesministerium für Bildung und Forschung. *Federal Ministry of Education and Research*, 2002. http://www.bmbf.de/en/index.php3 (12/08/2002).

[DD95] Wolfgang Domschke and Andreas Drexl. *Einführung in Operations Research*. Springer, Berlin, 3rd edition, 1995.

[DHH+02] Oleg Dück, Marco Hellmann, Jörg Hevers, Torsten Klie, Florian Maul, Nils Oppermann, Björn Rösemeier, and Christoph Schalong. *Simulationsumgebung – Dokumentation*. TU Braunschweig – Institut für Wirtschaftswissenschaften – Abteilung Allgemeine BWL, Informationsmanagenent und Wirtschaftsinformatik, Februar 2002. Seminar work.

[Ede00] Walter Edelmann. *Lernpsychologie*. Beltz, Weinheim, 6th edition, 2000.

[Fla98] David Flanagan. *Java in a Nutshell*. O'Reilly, Cambridge, 2nd edition, 1998.

[fok02] fokno.org. *Parser*, 2002. http://www.fokno.org/parser/download.html (12/01/2002).

[Gon02] Li Gong. *Java^{TM}2 Platform Security Architecture*. Sun Microsystems,Inc., 1.2 edition, 2002. http://java.sun.com/j2se/1.4.1/docs/guide/security/spec/security-spec.doc.html (12/01/2002).

[Gre98] Dale Green. *The Java Tutorial Continued: The Rest of the JDK*, chapter Trail: The reflection API. Addison Wesley Publishing Company, 1st edition, 1998.

[Hal02] Half-Baked Software Inc. *Hot Potatoes*, 2002. http://www.halfbakedsoftware.com (12/08/2002).

[Hud99] Scott E. Hudson. *CUP – LALR Parser for Java^{TM}*. GVU Center, Georgia Tech, September 1999.

http://www.cs.princeton.edu/~appel/modern/java/CUP/ (12/01/2002).

[Hyp02] Hyperwave AG. *HYPERWAVE – The Power of Wisdom*, 2002. http://www.hyperwave.de/d/ (12/08/2002).

[IBM02] IBM. *Jikes*, October 2002. http://ibm.com/developerworks/oss/jikes/ (12/01/2002).

[Ins02] Institute for Computing Systems Architecture, University of Edinburgh. *simjava*, 2002. http://www.dcs.ed.ac.uk/home/hase/simjava/ (12/08/2002).

[JAMX00] Andrew F. Seila John A. Miller and Xuewei Xiang. The jsim web-based simulation environment. In *Future Generation Computer Systems (FGCS), Special Issue on Web-Based Modeling and Simulation*, volume 17, pages 119–133. Elsevier North-Holland, October 2000. http://chief.cs.uga.edu/ jam/jsim/papers/mailfiles/simjournal.ps.

[Ker99] Michael Kerres. Didaktische konzeption multimedialer und telemedialer lernumgebungen. In *HMD - Praxis der Wirtschaftsinformatik*, volume 205, pages 9–21. dpunkt.verlag, Heidelberg, Februar 1999. http://ddi.cs.uni-potsdam.de/HyFISCH/Multimedia/Learning/DidaktischeKonzeptionKerres.pdf.

[KG95] Ján Košturiak and Milan Gregor. *Simulation von Produktionssystemen*. Springer, Wien, 1995.

[Kle01] Nik Klever. Elementarteilchen – XML Schema: Objektorientierte Dokumenttypdefinitionen. *iX – Magazin für professionelle Informationstechnik*, (6):62–66, Juni 2001.

[Kre02] Marc Kretschmer. *E-Learning in der Hochschulausbildung*. Books on Demand, Leutenbach, 2002.

[Lef94] Guy R. Lefrancois. *Psychologie des Lernens*. Springer, Berlin, 3rd edition, 1994.

[LK91] Averill M. Law and W. David Kelton. *Simulation modeling & analysis*. McGraw-Hill, Inc., New York, 2nd edition, 1991.

[MDB01] Rajiv Mordani, James Duncan Davidson, and Scott Boag. *Java API for XML Processing*. Sun Microsystems, Inc., version 1.1 final realease edition, February 2001. http://java.sun.com/xml/jaxp/dist/1.1/jaxp-1_1-spec.pdf (12/01/2002).

[Mil01] John A. Miller. *JSIM: A Java-Based Simulation and Animation Environment*. Computer Science Department, University of Georgia, 2001. http://chief.cs.uga.edu/ jam/jsim/ (12/08/2002).

[mon02] monist-Gruppe. *monist - Modellsimulation neuronaler und kognitiver Informationsverarbeitung*, 2002. http://www.monist.de/ (12/08/2002).

[PC00] Michael Pidd and Ricardo A. Cassel. Using java to develop discrete event simulations. *Journal of the Operational Research Society*, 51:405–412, 2000.

[Pid92] Michael Pidd. *Computer Simulation in Managemet Science*. John Wiley & Sons, Chichester, 3rd edition, 1992.

[RWT02] RWTH Aachen, Ludwig-Maximilians-Universität München, Cisco Systems GmbH. *Virtuelles Informatik Praktikum*, 2002. http://www.vip.rwth-aachen.de/index.html (12/08/2002).

[Sci02] SciFace Software GmbH & Co. KG. *MuPAD Pro*, 2002. http://www.sciface.com/MuPAD/Pro.shtml (12/08/2002).

[Sun01] Sun Microsystems, Inc. *XML Processing*, 2001. http://java.sun.com/j2se/1.4/docs/guide/xml/jaxp/index.html (12/01/2002).

[Sun02] Sun Microsystems,Inc. *Default Policy Implementation and Policy File Syntax*, April 2002. http://java.sun.com/j2se/1.4.1/docs/guide/security/ PolicyFiles.html (12/01/2002).

[SW02] Michael Seeborger-Weichselbaum. *Java/XML*. verlag moderne industrie Buch AG & Co. KG, Landsberg, Landsberg, 1st edition, 2002.

[Tec02] Tecnomatix Technologies Ltd. *eM-Plant*, October 2002. http://www.emplant.com/Frames/eMPlant/default.html (12/01/2002).

[Uni02a] Universität Bamberg. *Clear Campus*, 2002. http://www.iaws.wiai.uni-bamberg.de/forschung/projekte/lernsoft/clear-campus/index.html (12/08/2002).

[Uni02b] Universität Essen, Otto-Friedrich-Universität Bamberg, Friedrich-Alexander-Universität Erlangen-Nürnberg, DIN Deutsches Institut für Normung e.V. *VAWI - Virtuelle Aus- und Weiterbildung Wirtschaftsinformatik*, 2002. http://www.vawi.de/ (12/08/2002).

[Uni02c] University of Koblenz. *PL2TME online*, 2002. http://www.uni-koblenz.de/~hasan/pl2tme/ (12/08/2002).

[Uni02d] University of Koblenz-Landau, University of Paderborn, Saarland University of applied Science, Saarland University, Humboldt-University Berlin. *In^{2}Math*, 2002. http://www.in2math.de/ (12/08/2002).

[Ven99] Bill Venners. *Inside the JAVA2 Virtual Machine*. McGraw-Hill, New York, 2nd edition, 1999.

[VG01] Stefan Voß and Kai Gutenschwager. *Informationsmanagement*. Springer, Berlin, 2001.

[Vir02] Virtual Global University, School of Business Informatics. *Virtual Global University, School of Business Informatics*, 2002. http://www.vg-u.de (12/08/2002).

[Web01] WebCT. *WebCT*, 2001. http://www.webct.com/ (12/08/2002).

[Zim95] Philip G. Zimbardo. *Psycholgie*. Springer, Berlin, 6th edition, 1995.