

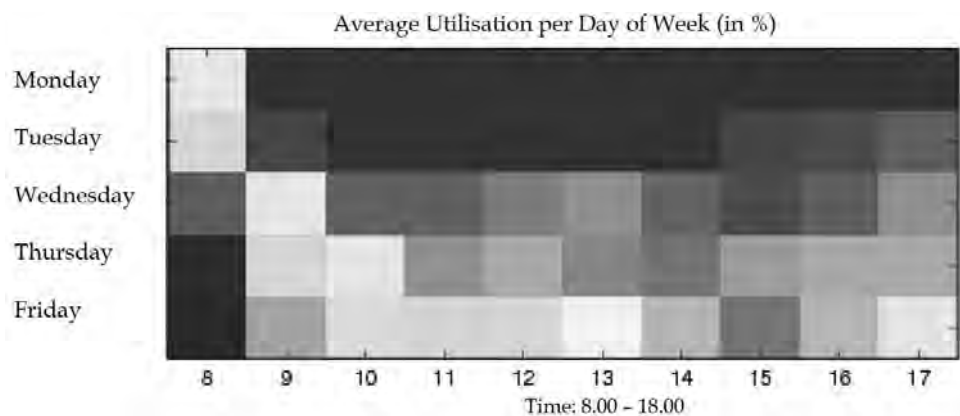
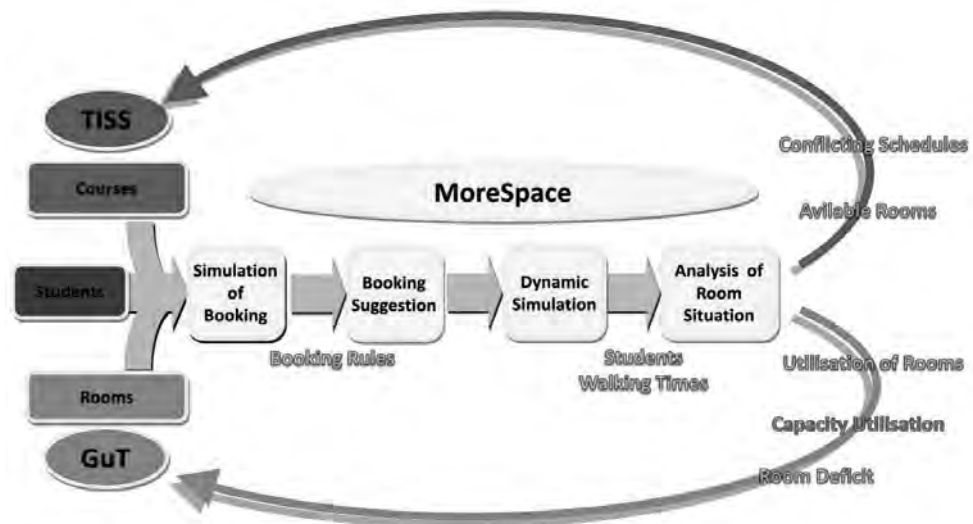


FBS
19

Fortschrittsberichte Simulation
Advances in Simulation

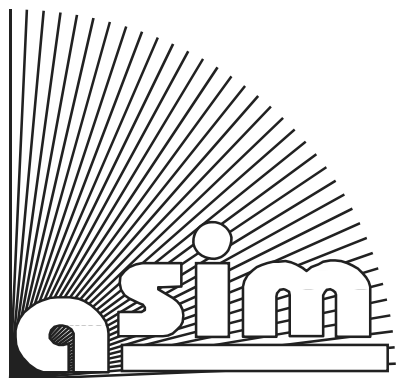
Integration of Agent Based Modelling in DEVS for Utilisation Analysis: The MoreSpace Project at TU Vienna

Shabnam Tauböck



ISBN print 978-3-903024-85-4 ISBN ebook 978-3-903347-19-9 DOI 10.11128/fbs.19





FBS Fortschrittsberichte Simulation

Advances in Simulation

Herausgegeben von **ASIM** - Arbeitsgemeinschaft **Simulation**, Fachausschuss der **GI** – Gesellschaft für Informatik - im Fachbereich **ILW** – Informatik in den Lebenswissenschaften

Published by **ASIM** – German Simulation Society, Section of **GI** – German Society for Informatics - in Division **ILW** – Informatics in Life Sciences

ASIM FBS 19

Shabnam Tauböck

**Integration of Agent Based Modelling
in DEVS for Utilisation Analysis:
The MoreSpace Project at TU Vienna**

FBS - Fortschrittsberichte Simulation / Advances in Simulation

Published on behalf of **ASIM** – German Simulation Society, → www.asim-gi.org

ASIM is a section of of **GI** – German Society for Informatics, → www.gi.de, in division **ILW** – Informatics in the Life Sciences, → fb-ilw.gi.de

Herausgegeben von **ASIM** - Arbeitsgemeinschaft Simulation, → www.asim-gi.org

ASIM ist ein Fachausschuss der **GI** - Gesellschaft für Informatik, → www.gi.de, im Fachbereich **ILW** – Informatik in den Lebenswissenschaften, → fb-ilw.gi.de

Series Editors

Prof. Dr.-Ing. Th. Pawletta (ASIM), HS Wismar, Thorsten.pawletta@hs-wismar.de

Prof. Dr. D. Murray-Smith (EUROSIM / ASIM), Univ. Glasgow,

David.Murray-Smith@glasgow.ac.uk

Prof. Dr. F. Breitenecker (ARGESIM / ASIM), TU Wien, Felix.Breitenecker@tuwien.ac.at

Title: Integration of Agent Based Modelling in DEVS for Utilisation Analysis:

The MoreSpace Project at TU Vienna

Author: Shabnam Tauböck, Shabnam.tauböck@gmx.net

FBS Volume: 19

Typ: PhD Thesis

ISBN print: 978-3-903024-85-4, 2019, TU-Verlag, Vienna, 2019;

Print-on-Demand, www.tuverlag.at

ISBN ebook: 978-3-903347-19-9, 2019, ARGESIM Publisher Vienna, 2016;

www.argesim.org

DOI: 10.11128/fbs.19

Pages: 123



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Diese Dissertation haben begutachtet:

.....
Felix Breitenecker

.....
Gašper Mušič

DISSERTATION

Integration of Agent Based Modelling in DEVS for Utilisation Analysis: The MoreSpace Project at TU Vienna

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaft unter der Leitung von

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker
Institut 101
Institut für Analysis und Scientific Computing

eingereicht an der Technischen Universität Wien
bei der Fakultät für Mathematik und Geoinformationen
von

Dipl.-Ing. Shabnam Michèle Tauböck
Matrikelnummer: 9411564
Hirschengasse 15, 1060 Wien

Wien, am

Table of Content

Abstract.....	8
Zusammenfassung.....	12
Acknowledgments	13
Chapter 1: Introduction to Modelling and Simulation.....	14
1.1 Reasons for Simulation.....	15
1.2 Data	16
1.3 Validation and Verification.....	16
1.3.1 Comparison to Other Models	16
1.3.2 Event Validity:	16
1.3.3 Extreme Condition Tests	17
1.3.4 Face Validity	17
1.3.5 Fixed Values.....	17
1.3.6 Historical Data Validation	17
1.3.7 Data Validity	17
1.3.8 Conceptual Model Validation	17
1.3.9 Model Verification	17
1.4 Continuous Simulation.....	18
1.5 Discrete Simulation.....	19
1.6 Discrete Event Simulation.....	20
1.6.1 Eventlist	21
1.7 Cellular Automata.....	21
1.8 Agent Based Simulation.....	24
Chapter 2: Database Controlled Assembling of a Simulation Model	28
2.1 From Model to Simulation.....	28
2.2 Basic Idea and Concept	30
2.2.1 Interface to External Data Source.....	31
2.2.2 Automatic Model Generation.....	32
2.3 Simulation Database	33
2.3.1 System Elements.....	33
2.3.2 System Structure Parameters.....	33

2.3.3	System Parameters	33
2.3.4	Input Parameters	33
2.4	Simulation Environment	33
2.4.1	The Template	33
2.4.2	The Simulation Library	34
2.4.3	Functions	34
2.4.4	The Simulation Model	35
2.5	Result Aggregation	35
2.6	Effects on Verification and Validation	35
Chapter 3:	MoreSpace	37
3.1	History of the Vienna University of Technology	37
3.2	Univercity 2015	38
3.3	TU Campus	39
3.4	Problems with Room Management at TU Vienna	40
3.5	Definition of Terms	41
3.5.1	Utilisation of Rooms	41
3.5.2	Capacity Utilisation	41
3.5.3	Travelling Time	41
3.5.4	Clearance Time	41
3.5.5	Not Successfully Booked Lectures	42
3.5.6	Compulsory Lecture	42
3.5.7	Elective lecture	42
3.5.8	Additional Lectures	42
3.6	Intended Purpose of MoreSpace	42
3.6.1	Aide for Booking Courses	44
3.7	Benefits	45
Chapter 4:	MoreSpace: Model Description	46
4.1	The MoreSpace Model from DEVS Point of View	48
4.1.1	Queue In and Queue Out:	48
4.1.2	Door	49
4.1.3	Room	49
4.1.4	The Student: A Double Agent?	50
4.2	Room Management	54

4.3	Booking Management.....	54
4.3.1	Sorting by Date	54
4.3.2	Sorting by Duration	54
4.3.3	Sorting by Capacity Demanded	54
4.3.4	Sorting by Category of Lecture	54
4.3.5	Sorting by Type of Lecture.....	55
4.3.6	Sorting by Modus of Lecture	55
4.3.7	Additional Options	55
4.3.8	Behaviour of Rooms.....	55
4.4	Student Numbers - a General Problem	55
4.5	Step I: Simulation Model Assembly	59
4.6	Step II: Simulation of Booking Procedure.....	59
4.6.1	Sorting.....	61
4.6.2	Booking Criteria	61
4.7	Step III: Dynamic Simulation.....	62
Chapter 5:	The Simulation Approach.....	64
5.1	Simplify, Simplify! (Henry David Thoreau).....	64
5.2	Simulation Peoples Movement in ED	65
5.3	Hybrid Model: Connection to the CA Model.....	69
5.3.1	Interface between JAVA and ED.....	71
5.3.2	Problems.....	71
5.3.3	Impact on the MoreSpace Model	73
5.4	The MoreSpace Simulation Environment: Enterprise Dynamics.....	74
5.4.1	Everything is an Atom.....	75
5.4.2	Of Mothers and Daughters	76
5.4.3	The Atom Editor.....	76
5.4.4	Events.....	77
5.4.5	Eventlist	78
5.4.6	Attributes.....	78
5.4.7	Tables	79
5.4.8	The 4d Script	79
5.4.9	Enterprise Dynamics Simulation Environment	79
5.5	MoreSpace Application and Model file	80

5.6	MoreSpace GUI	81
5.7	MoreSpace Library	81
5.7.1	Building	81
5.7.2	Room	81
5.7.3	Courses	83
5.7.4	Student.....	84
Chapter 6:	Functional Description of MoreSpace	87
6.1	Booking Procedure.....	87
6.1.1	Best Fit Loop	88
6.1.2	Optimal Environment Loop	88
6.1.3	Room Setup Loop.....	88
6.2	Student Creation Procedure	89
6.3	Simulation Run.....	99
6.4	Functions regarding Data Exchange	100
6.4.1	Interface between ED and the Simulation Database	100
6.4.2	Interface to TUWIS++ / TISS.....	100
6.5	Interface from ED to the CA Model.....	100
6.6	Functions regarding Alternative Scenarios	103
6.6.1	Reduction of Not Booked Lectures.....	103
6.6.2	Increase of Utilisation – Keeping of ‚Free‘ Rooms.....	104
Chapter 7:	Experimenting with MoreSpace.....	105
7.1	MS GUI	105
7.2	Changes to Design	107
7.2.1	Room Structure.....	107
7.2.2	Courses	107
7.2.3	Students	107
7.3	Tables for Input Data.....	107
7.3.1	Studies.....	107
7.3.2	Courses	107
7.3.3	Building	108
7.3.4	Room Structure.....	108
7.3.5	Room Reservation.....	108
7.4	Changes to Behaviour.....	109

7.4.1	Behaviour of Rooms.....	109
7.4.2	Behaviour of Students	109
7.4.3	Booking Behaviour.....	109
7.5	Alternative Scenarios.....	110
7.5.1	Comparison of Booking Rules.....	110
7.5.2	Reduction of Not Booked Lectures.....	110
7.5.3	Increase of Utilisation - Keeping of ‚Free‘ Rooms.....	110
7.6	Tables for Simulation Results.....	111
7.6.1	Documentation	111
7.6.2	Not Successfully Booked Lectures.....	111
7.6.3	Room Utilisation.....	111
7.6.4	NichtExact	111
7.6.5	NichtGebäude.....	112
7.6.6	List of Booked Lectures	112
7.7	Results: Database Reports.....	112
7.7.1	Utilisation of Lecture Rooms	112
7.7.2	Capacity Utilisation of Lecture Rooms	114
7.7.3	Not Successful Booking.....	114
7.8	Accessibility of Lectures.....	116
Chapter 8:	Conclusion.....	117
	Table of Definitions.....	119
	Table of Figures	120
	References.....	122

*For Georg:
you are the Best*

*Get your facts first,
and then you can distort them as much as you please.*

- *Mark Twain*
(1835 - 1910)

Abstract

In this dissertation an approach to integrating agent based modelling into a discrete event simulation system is developed and tested in the course of the MoreSpace project done at the Vienna University of Technology. Considering the basic formalism of DEVS a Discrete Event System Specification (DEVS) is a structure $M = (X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$, where X is the set of input values, S the set of states, Y the set of output values, δ_{int} the internal transition function, δ_{ext} the external transition function, δ_{con} the confluent transition function, λ the output function, ta the time advance function. Several of this so called atomic models can be put together to form a coupled model.

An agent based model is defined as tuple $\langle A, E \rangle$ where A is a set of agents with $A = \cup a^k$ and $1 \leq k \leq N_{agents}$ and E is the Environment. The agent k itself is defined as a function $a^k: R_S^k \rightarrow \Lambda^k$; an Environment E is defined as tuple $\langle \Sigma, \tau \rangle$ where Σ is the system state, $\tau: R_\Lambda \rightarrow \Sigma$ is a state transformer function that changes the system state based on $r_i^k \in R_\Lambda$ with

- s_j^k is the j^{th} set of state variables that is seen by agent k where $1 \leq k \leq N_{agents}$
- α_j^k is the j^{th} action done by agent k in response to a set of state variables s_j^k
- $\Lambda^k = \cup \alpha_j^k$ all actions done by agent k
- r_i^k , the i^{th} run of agent a^k , is the i^{th} sequence of interleaved $s_0^k, \alpha_0^k, s_1^k, \alpha_1^k, \dots$
- $R^k = \cup r_i^k$, the set of runs of agent k , where $1 \leq i \leq N_{runs}^k$
- $R_S^k = R^k$ that ends with an s_j^k

During the last years the feasibility of finding an equivalent discrete event model for any agent based model that conforms to the specification given above has been discussed and shown in several publications. This implies that every agent that is part of such an agent based system can be integrated in a discrete event system. It is obvious that both modelling techniques have their advantages and drawbacks. Discrete Event Simulation is known to be efficient and fast as long as the concept of event driven time steps is able to use its advantage of jumping over time intervals where no changes to the system state occur and only update the system elements of the time points where events are schedules or triggered. People's behaviour and movement is hard to model in such a system. Movement i.e. is usually not along a certain foretold line - people do tend to take the shortest route from A to B but the easiest way that seems to be free of obstacles and decisions between different ways

are due to individual personal preferences – some people prefer to take the stairs, some rather wait for the elevator, some take on the longer walk to the escalator to save themselves from having to take the stairs.

Therefore it should be possible to develop a model, where both systems - discrete event and agent based - coexist.

This work focuses on the discrete event simulation environment Enterprise Dynamics (ED) that is designed to develop discrete event simulation systems. The formalism shown by DEVS is very much reflected in the basic make up of ED. Basic elements called atoms would refer to the atomic models described by DEVS. They are all identical in their basic structure, their behaviour defined by the transition functions. A simulation model is build by using atoms again, again a concept conform to the coupled models in DEVS. The main idea now is to create an atomic model that behaves like an agent to integrate it in the discrete event simulation.

The main characteristic of an agent is its ability to make its own decisions based on its own state and its environment. To model the human decision making in this work the Utility Theory is used: it assumes that the decision process has two elements: the options and the evaluation function, called utility function that maps each option in the choice set to a numerical value. The function $u: \mathcal{X} \mapsto \mathbb{R}$ is a utility function if \mathcal{X} is the set of choices. Preferences of the modelled individuals can be: no preferences (\sim), prefer the first over the second option ($>$) or the second over the first ($<$). If the preferences observed in the individuals modelled correspond to the relations given by $u(\circ)$, $u(\circ)$ is called a valid utility function for the given decision problem.

As application of this theory the MoreSpace project done for the TU Vienna is introduced: The simulation of the booking management for all courses held at the TU Vienna and the student flow to determine the utilisation and accessibility of lectures and the allocated space. Dealing with the simulation of the whole TU Campus in Vienna as well as the student behaviour it needs the best of both worlds, DEVS and ABM, to cover all characteristics of this system. Entering and leaving rooms can be best modelled using a queuing system whereas students are best represented by agents to model their individual behaviour regarding the attendance of lectures as well as the travelling between lecture halls. The main idea is to use an ED atom to create an agent that interacts with and moves through the discrete event simulation, regarding it as its environment.

According to the aforementioned definition an agent goes through a sequence of states s_j^k following certain actions α_j^k ; from the DEVS point of view an action α_j^k of an agent is a time consuming activity; therefore it is possible to replace it with an event triggered at the beginning of this activity using δ_{ext} the external transition function; the time consumed by the activity needs to be represented by ta the time advance function. So basically every action α_j^k needs to be replaced by an event $x \in X$ and $\delta_{ext}(x)$ to ensure the correct update of the state variables of the atom itself as well as its environment. The state transformer function τ that updates the environment of the agent needs to be ingrained in the interaction of the agent with the DEVS based system.

The main problem is to coordinate the behaviour of the DEVS system with the AB elements. In the case of the project MoreSpace the solution to this problem lays partly in the definition of the problem itself and partly in the simulation package used. Enterprise Dynamics is a discrete event simulation environment that represents the DEVS formalism in its basic makeup. All elements, called atoms, share the same layout and basic functionality. Several atoms can be used to build a more complex atom; several of them can be combined again, reflecting the principle of coupled models.

A given set of events may cause an update of the atoms state, the transition functions can be defined using the ED internal programming language. The time advance function ta is handled using an eventlist that controls the adjustment of time steps.

But additional to everything that forms an atomic or coupled model, ED offers spatial attributes as well as the functionality to move objects through space. And the set X of incoming values to trigger an event includes two elements that assist the ABM as well: the timed event x_t and the message event x_m . Both are not caused by a change of state of a model but in the first case by reaching a certain point in time and in the second by a message from outside of the system. Especially the latter option makes it possible to control elements in the DEVS system from external.

In case of the MoreSpace project this option was used to transfer the agents out of the ED model into a JAVA model for the calculation of travelling times at the movement between two spatial locations.

The project *MoreSpace* was launched to develop a software tool to support the planning phase of "University2015". "University2015" is a project of the Vienna University of Technology (TU Vienna) to renovate all university buildings and to improve the existing infrastructure and the inherent processes. This shall also be done by determining and evaluating the (spatial) resources required.

The project was launched to assist the department of Gebäude und Technik (GuT) at the TU Vienna during the planning phase of University 2015. The team responsible for this project used static methods to calculate an assessment of the number of square meters each faculty of the TU Vienna needs for lectures and other student related activities. This calculation was based on the number of students that took exams and the hours of teaching for each faculty. The resulting numbers did show the need of square meters required to accommodate the number of student during the lectures but did not take into consideration how these square meters are used over time. One thing that complicates things considerably is the fact that lectures at the TU Vienna do not take place one after the other in a strict pattern as it is done at schools or colleges but are set at times favoured by the lecturer. This results in a weekly timetable for the student that contains times where several lectures may be very close to each other, even overlapping, and times where no lectures at all are taking place, leaving a big time gap. Over the whole of the TU Vienna one can say that there are certain times during the week that are more or less preferred by many lecturers, resulting in a high demand of rooms during certain time periods, where

other times – i.e. Friday afternoons – are not that popular. The fact that the required amount of square meters is available, does not mean it is used in a way that ensures that it will be indeed possible to acquire a room for a lecture at a certain time.

The idea to develop a simulation was born, a tool to reproduce the situation as it currently is concerning the lectures and their demand on room and experiment with the room structure. The basic idea was to keep the lectures exactly as they were and test if the new room situation was able to accommodate all of them.

In 2008 the working group consisting of the Research Group for Mathematical Modelling and Simulation and the Research Group for Real Estate Development and Management at the Vienna University of Technology were asked by GUT TU, co-responsible for Project Management of “Univercity 2015”, to introduce a model for the room management that can simulate the usage of resources to optimize the planning of the rooms and the future usage.

The agents represent students that attend lectures at a given time; therefore it is possible to use timer events for controlling the activity of the agent atoms. Triggered by a timed event the activity of the agent takes place: they either change their location by moving towards a certain lecture hall, take a decision whether they should attend a lecture and chose one in case of several overlapping, they enter a queue, or leave the ED model for the more exact simulation of peoples movement in a JAVA model.

But it is the agents themselves that control the setting of these events: each atomic agent has its own personal settings and takes its own decisions.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Kombination von Discrete Event Simulation und Agent Based Modelling. Der Schwerpunkt liegt hierbei in der Integration des Agentenkonzepts in ein Discrete Event Simulationssystem.

Geht man von der formalen Definition der Discrete Event System Specification aus, so ist ein DEVS eine Struktur $M = (X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$, wobei X die Menge an Eingangswerten ist, S die Menge an Zuständen, Y die Menge der Ausgangswerte, δ_{int} die interne transition function, δ_{ext} die externe transition function, δ_{con} die konfluente transition function, λ die output function, ta die time advance function.

Mehrere dieser atomic models können zu einem sogenannten Coupled Model zusammengefaßt werden.

Ein Agent Based Model ist definiert als Tupel $\langle A, E \rangle$ wobei A die Menge der Agenten darstellt, mit $A = \cup a^k$ und $1 \leq k \leq N_{agents}$ und E ist die Umgebung. Der Agent k selbst ist definiert als Funktion $a^k: R_S^k \rightarrow \Lambda^k$;

Eine Umgebung E ist definiert als Tuple $\langle \Sigma, \tau \rangle$ wobei Σ den System Zustand darstellt, $\tau: R_\Lambda \rightarrow \Sigma$ ist eine Zustandsübergangsfunktion das den Zustand des Systems ändert basierend auf $r_i^k \in R_\Lambda$ wobei

s_j^k ist die j^{te} Menge von Zustandsvariablen die vom Agent k beobachtet wird,

wobei $1 \leq k \leq N_{agents}$

α_j^k ist die j^{te} Aktion die der Agent k durchführt in Reaktion auf die Zustandsvariablen s_j^k

$\Lambda^k = \cup \alpha_j^k$ aller Aktionen durchgeführt von Agent k

$r_i^k =$ der i^{te} Durchlauf des Agenten a^k , ist die i^{te} Folge $s_0^k, \alpha_0^k, s_1^k, \alpha_1^k, \dots$

$R^k = \cup r_i^k$, die Menge aller Durchläufe des Agenten k , wobei $1 \leq i \leq N_{runs}^k$

$R_S^k = R^k$ endend mit s_j^k

Im Laufe der letzten Jahre wurde die Möglichkeit agentenbasierte Modelle in DEVS Systemen abzubilden in diversen Veröffentlichungen diskutiert. Die Übersetzbarkeit von ABM in DEVS wurde dabei zu Genüge demonstriert. Beide Methoden verfügen über ihre distinkten Vor- und Nachteile: Das legt nun die Überlegung nahe ein System zu entwickeln, in dem agentenbasierte Element mit DEVS Elementen coexistieren um damit die Vorteile beider Welten je nach Bedarf ausnutzen zu können.

Acknowledgments

I would like to take this opportunity to express my gratitude to all those people who contributed, directly or indirectly, to this work.

My very sincere thank you goes to Prof. Dr. Felix Breitenecker who never stopped supporting me and did his best to help me balancing work and my ever growing family. Without him I would not be where I am now.

To my dear husband Georg I can only say: Thank you for your endless patience and calm.

And last but not least I want to thank my friends and family for their constant support, be it in taking over babysitting duty whenever needed or listening to my nervous ramblings when work became too much or simply dragging me out to have some fun.

So thank you to all of you, who supported me and believed in me – it was all of you together who made this possible for me.

Chapter 1: Introduction to Modelling and Simulation

A model is a representation (especially in miniature), or description of a system of entities, phenomena, or processes. Basically a model is a simplified abstract view of the complex reality. It may focus on particular views, enforcing the "divide and conquer" principle for a compound problem.

A modelling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. One example for modelling languages is the Unified Modelling Language (UML) for software systems.

By developing a model a certain extent of abstraction is required. It is normally not possible to model any system in a way that it is a complete and exact representation of the real thing. The complexity of such a model would become too much and the advantage of using a simulation model would quickly be lost.

It is necessary to clearly define the information one wants to gain from this model. Then it is possible to see which aspects of the real system can be simplified and which need to be represented as exactly as possible.

This is reflected in the following quote from Eugene Miya:

'Simulation is really only an extension of human intellect, not the way things behave in nature'

To more clearly define the phrase simulation we can take a look into the Oxford English Dictionary that describes simulation as:

'The technique of imitating the behaviour of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel.'

The Encyclopaedia of Computer Science, Nov. 1999:

'Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model in order to understand the behaviour of the system or evaluate strategies for the operation of the system.'

Or put another way, simulation is the technique of building a model of a real or proposed system so that the behaviour of the system under specific conditions may be studied. One of the key powers of simulation is the ability to model the behaviour of a system as time progresses.

Generally Simulation can be considered as conducting experiments using a model. This can be a crash test for cars where the situation during an accident is imitated to test the impact on the car. The situation that leads to the accident is of no interest for the results to be gained – this is where the abstraction of the system comes in.

What we are going to call simulation in this work is more precisely computer simulation which means the implementation of a model on a computer using some kind of program. It offers the possibility to study the behaviour of a system over time and usually it is able to deliver results quickly and in a way that makes further processing easily possible.

1.1 Reasons for Simulation

Quite often it is of great interest to study a certain system, especially under variable circumstances. Watching the effect of certain changes over a period of time can give interesting results concerning future developments.

For example in the field of Supply Chain Management it is of great interest to analyze the effect of changes in the customer behaviour. These effects may not show immediately but will move through the supply chain with a certain time delay.

Equally of interest would be the effect of different queuing strategies in a production plant to avoid bottle necks. The bottle necks themselves can be discovered by watching the original system, and then different strategies can be tried out to find a solution.

But it is often difficult to experiment with a real system or even not possible at all. Reasons for that can be:

- * High costs
- * Too fast processes - e.g. electrical processes
- * Too slow processes - e.g. evolutionary processes
- * Too huge - e.g. planets
- * Too small - e.g. molecules
- * Too dangerous - e.g. crash tests
- * Too time consuming
- * Too elaborate in setup

Especially if a high number of repetitions of the experiments are needed to cover all possibilities the usage of a computer model seems an interesting and reasonable solution.

If a system is to be modelled the decision has to be made which modelling approach should be used. There are different classifications of modelling:

- * Static - Dynamic
- * Continuous - Discrete
- * Stochastic - Deterministic

1.2 Data

Creating a simulation model usually requires a lot of data and that more often than not proves to be the Achilles Heel of any simulation project.

1.3 Validation and Verification

Before the results of a simulation study can be used for decision making it has to be verified that they are correct. This means the model has to be checked for its validity. This process is often underestimated in both, effort and importance. Validating a simulation model is not an easy task as quite often the amount of data needed is not available.

There are different approaches but usually it is a very individual process that very much depends on the structure of the model as well as the implementation of the simulation model.

There are several methods that may be used to check the model validity. Which one is the most applicable will depend on the individual model and its implementation.

1.3.1 Comparison to Other Models

If other - valid - models already exist a direct comparison can give a first insight about the correctness of the results achieved. If identical data sets are used in the comparison, the results should correspond.

1.3.2 Event Validity:

Events that take place in reality also have to occur in the simulation.

1.3.3 Extreme Condition Tests

The results for extreme conditions should still be reasonable. E.g. if in a supply chain no orders are placed, no deliveries should be made.

1.3.4 Face Validity

Expert knowledge can be used to assess the results of the simulation.

1.3.5 Fixed Values

If it is possible to use a set of fixed data where the results can be calculated without using the simulation for comparison a so called test scenario can be created.

1.3.6 Historical Data Validation

If historical data exist it can be used to determine if the simulation works the way the real system does.

1.3.7 Data Validity

The validity of the data the model is based upon is crucial for the development of an accurate model. It affects the conceptual design of the model as well as the behaviour of the model elements.

Checking this validity is an enormous and often impossible task. The best that can be done is to refine the routines of data collection and data processing. Each step has to be recorded and must be completely transparent to ensure the accuracy of the used data. Additional internal consistency checks can be done.

1.3.8 Conceptual Model Validation

Conceptual model validity is determining that (1) the theories and assumptions underlying the conceptual model are correct, and (2) the model representation of the problem entity and the model's structure, logic, and mathematical and causal relationships are "reasonable" for the intended purpose of the model.

1.3.9 Model Verification

Computerized model verification ensures that the computer programming and implementation of the conceptual model are correct. To help ensure that a correct computer program is obtained, program design and development procedures found in the field of software engineering should be used in developing and implementing the computer program.

1.4 Continuous Simulation

Continuous simulation is usually used for systems whose state changes continuously over time. These systems are most often described by using differential equations. Such continuous systems can be found in the field of natural sciences as physics, chemistry, medicine, electrical engineering, and so on.

Usually these models are deterministic, with only very few stochastic elements in them.

One of the most popular examples is the predator-prey relationship:

The *Lotka-Volterra equations*, also known as the predator-prey equations, are a pair of first order, non-linear, differential equations. They describe the interaction of two species, one a predator and one its prey.

They were proposed independently by Alfred J. Lotka in 1925 and Vito Volterra in 1926.

$$\frac{dx}{dt} = x(\alpha - \beta y)$$

$$\frac{dy}{dt} = -y(\gamma - \delta x)$$

Where

y is the population size of predators – for example wolves

x is the population size of its prey – for example rabbits

t represents the time

$\alpha, \beta, \gamma, \delta$ are parameters representing the interaction of the two species

$\frac{dx}{dt}$ and $\frac{dy}{dt}$ represent the growth of the two populations against time

The equations have periodic solutions which do not have a simple expression in terms of the usual trigonometric functions. However, an approximate linearised solution yields a simple harmonic motion with the population of predators following that of prey by 90°.

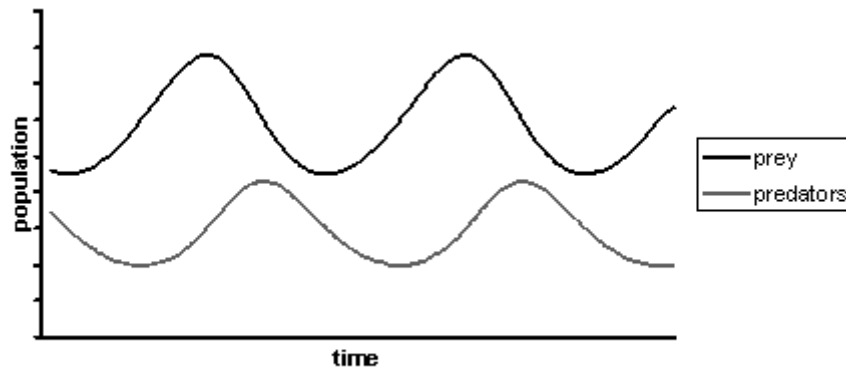


Figure 1-1 Predator – Prey Relationship

This model is used in studying the interaction between species and the effect that changes to one population will have on the other.

1.5 Discrete Simulation

In discrete processes the state of the system or its elements does change at certain time points. Typical examples are classic server – queue models as a supermarket or a post office.

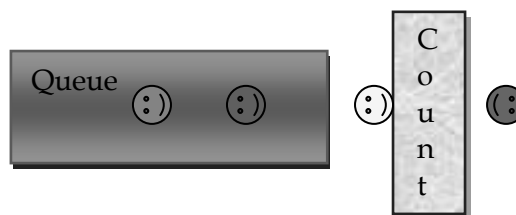


Figure 1-2 Simple Server – Queue Model

People wait in a queue in front of a counter as shown in Figure 2-2. Time moves in steps, e.g. as soon as the customer in front of the counter is finished several actions take place: the customer moves away, so the state of the place in front of the counter switches to idle. Therefore the first in the queue can step forward and leave the queue – the content of the queue decreases by one, the place in front of the counter becomes busy again. The next time step will be either if a new customer enters the queue and changes its content or the customer in front is finished with their business. The time in between is not regarded as no changes to the states of any model elements take place.

1.6 Discrete Event Simulation

Discrete event simulation is one way of building up models to observe the time based (or dynamic) behaviour of a system. There are formal methods for building simulation models and ensuring that they are credible. During the experimental phase the models are executed (run over time) in order to generate results. The results can then be used to provide insight into a system and a basis to make decisions on.

The main characteristic of Discrete Event Simulation is that the system state does only change at certain time points when events occur. Time moves from one of these events to the next, the time in between is of no relevance.

A typical event would be the entering of a queue in front of a server. This could be a customer in a supermarket, waiting at the cash.

Each event has a time of occurrence. If an event takes place it may cause changes to the state of individual objects as well as the system itself. These changes occur right at the time of the event or after a certain time delay, but not slowly over time as it may happen in continuous simulation. Any changes happen within a certain time point.

The occurrences of events and the time they take place create the timeline of the simulation run.

Discrete Event Simulation Specification (DEVS) is a widely used approach for modelling and simulation of dynamic discrete systems. The modern object-oriented DEVS world view regards active objects (entities) passing passive objects (stations) along given path.

Definition 1: *Discrete Event System Specification (DEVS)*

A DEVS is a structure

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

- * X is the set of input values
- * S is the set of states
- * Y is the set of output values
- * $\delta_{int} : S \rightarrow S$ is the internal transition function
- * $\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function, where
- * $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state set
- * e is the time elapsed since the last transition
- * X^b denotes the collection of bags over X (= sets in which some elements may occur more than once)
- * $\delta_{con} : Q \times X^b \rightarrow S$ is the confluent transition function
- * $\lambda : S \rightarrow Y^b$ is the output function
- * $ta : S \rightarrow \mathbb{R}_{0, \infty}^+$ is the time advance function

Definition 2: Coupled Model

A Coupled Model is a structure build of several atomic models with

$N = (X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, \text{Select})$, where

- * X_N, Y_N are the sets of input and output values of the coupled model
- * D is the set of component references, so that for each $d \in D, M_d$ is a DEVS model
- * For each $d \in D \cup \{N\}, I_d \subset (D \cup \{N\}) \setminus \{d\}$ is the set of influencer models on subsystem d
- * For each $i \in I_d, Z_{i,d}$ is the translation function, where
- * $Z_{i,d}: \begin{cases} X_N \rightarrow X_d & \text{if } i = N \\ Y_i \rightarrow Y_N & \text{if } d = N \\ Y_i \rightarrow X_d & \text{otherwise} \end{cases}$
- * $\text{Select}: 2^D \rightarrow D$ is a tie breaking function for simultaneous events; it must verify $\text{Select}(E) \in E$, with $E \subset 2^D$, the set of components producing the simultaneity of events

1.6.1 Eventlist

The occurrence of events and their order need to be controlled somehow. In discrete simulation software programs this usually is done by a so called event list or event chain. This list contains all future events as well as some additional information as the time of occurrence and optionally a priority. During the simulation run new events are added to this list. Only if all events are executed and the list is empty the simulation run has reached its logical end. Of course additional conditions for ending the simulation run can be set, and then the simulation run will be stopped even if the event list does still contain future events.

One big concern in Discrete Event Simulation is the handling of events that take place at the same time. Usually the event list contains all events in order of their occurrence. Events that will take place at the same time are simply listed in the order they have been added to this list.

To ensure the correct order of event priorities have to be assigned to give a ranking for the execution. This corresponds to the Select function in Definition 2.

1.7 Cellular Automata

A cellular automaton is a collection of cells placed on a grid that can assume certain states according to given rules. These rules are applied iteratively in equidistant time

steps to evaluate the current state of each cell. The state of each cell is defined by the set of rules and the state of the neighbouring cells.

Definition 3: Cellular Automaton (CA)

A cellular automaton is a 4-tuple $\langle L, \Sigma, \mathcal{N}, \phi \rangle$, where

- * $L = \mathbb{Z}^d$ is a dimensional lattice of cells indexed by integers
- * Σ is a finite set of cell states
- * \mathcal{N} is a neighborhood scheme, which is a finite list of lattice vectors from \mathbb{Z}^d
- * and ϕ is a local transition function $\phi: \Sigma^{\mathcal{N}} \rightarrow \Sigma$

A configuration is simply an assignment of states from Σ to each cell in L . Given a configuration, the transition function ϕ determines a new configuration after a single discrete time step. For each cell x , the new state is given by applying ϕ to the current states of the cells in the list $x + \mathcal{N}$.

So, each cellular automaton is defined by

- * the grid the cells are placed upon
- * the definition of the neighbourhood
- * the initial state of the cells
- * the set of rules defining the changes to the state of the cells
- * the number of states a cell can assume

Which cells are considered to be neighbouring cells is conditioned by the type of Neighbourhood the cellular automata works with.

The von Neumann neighbourhood is a diamond-shaped neighbourhood that can be used to define a set of cells surrounding a given cell (x_0, y_0) that may affect the evolution of a two-dimensional cellular automaton on a square grid.

The von Neumann neighbourhood of range r is defined by

$$N_{x_0, y_0}^v = \{(x, y) : |x - x_0| + |y - y_0| \leq r\}$$

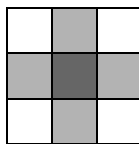


Figure 1-3: von Neumann Neighbourhood

I.e. the von Neumann neighbourhood for $x_0 = 0, y_0 = 0, r = 1$ is the set of neighbours:

$$N = \{\{0, -1\}, \{-1, 0\}, \{0, 0\}, \{+1, 0\}, \{0, +1\}\}$$

The number of cells in the von Neumann neighbourhood of range r is the centred square number $2r(r + 1) + 1$.

The Moore neighbourhood is a square-shaped neighbourhood that can be used to define a set of cells surrounding a given cell (x_0, y_0) that may affect the evolution of a two-dimensional cellular automaton on a square grid. The Moore neighbourhood of range r is defined by

$$N_{x_0, y_0}^M = \{(x, y) : |x - x_0| \leq r, |y - y_0| \leq r\}$$

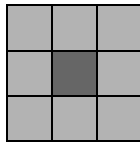


Figure 1-4: Moore Neighbourhood

I.e. the Moore neighbourhood for $x_0 = 0, y_0 = 0, r = 1$ is the set of neighbours

$$N = \{\{-1, -1\}, \{0, -1\}, \{1, -1\}, \{-1, 0\}, \{0, 0\}, \{+1, 0\}, \{-1, +1\}, \{0, +1\}, \{+1, +1\}\}$$

The number of cells in the Moore neighbourhood of range r is the odd squares: $(2r + 1)^2$.

Usually cellular automata are considered to be set upon an infinite grid. In reality this is not always the case so certain rules have to be applied for the calls on the edge of the grid. One way is to define a fixed state for the 'missing' neighbouring cells; another is to connect the left cells with right, and the upper cells with lower. In the case of a rectangular grid this would result in a Torus.



Figure 1-5: Torus

1.8 Agent Based Simulation

Agent Based Modelling is a relative newly rediscovered and powerful simulation technique for discrete systems. An Agent Based Model consists of autonomous elements that are able to make their individual decisions based on the state of their situation and a given set of rules.

The concept of agent based modelling was developed in the late 1940s but since a agent based simulation model very quickly requires a huge amount of computational resources its real breakthrough came when the according hardware became first available in the 1990s.

Its history can be tracked to the Von Neumann machine, one of the first cellular automata and John Conway's Game of Life.

The definition of Jennings und Wooldridge, the so called "Weak Notion of Agency" lists the following attributes of an agent:

- * **Autonomy:** each agent acts on its own and decides its own behaviour
- * **Social ability:** agents are able to communicate with each other
- * **Reactivity:** agents react to their environment and changes therein
- * **Pro-activeness:** Agents do not only react to their environment but act on their own as well

Stan Franklin and Art Graesser tried to find a mathematical definition for an agent, but call it 'weak around the edges'.

'An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.'

According to this the attributes of an agent can be listed as:

Reactive (sensing and acting): responds in a timely fashion to changes in the environment

Autonomous: exercises control over its own actions

Goal-oriented(pro-active, purposeful): does not simply act in response to the environment

Temporally Continuous: is a continuously running process

Communicative (socially able): communicates with other agents, perhaps including people

Learning (adaptive): changes its behaviour based on its previous experience

Mobile: able to transport itself from one machine to another

Flexible: actions are not scripted

Character: believable "personality" and emotional state.

Agents may be usefully classified according to the subset of these properties that they enjoy. Every agent, by this definition, satisfies the first four properties. Adding other properties produces potentially useful classes of agents, for example, mobile, learning agents. Thus a hierarchical classification based on set inclusion occurs naturally. Mobile, learning agents are then a subclass of mobile agents.

Agent-based models describe the behaviour and interactions of a system's parts from the bottom up; There are several situations, where agent based modelling is a more practical and realizable approach then i.e. describing the system using differential equations:

Complex interactions: it can be particularly useful using agent based modelling if describing the discontinuity of individual behaviour is difficult, for example, using differential equations.

Heterogeneous populations: Using an agent-based approach offers the possibility to design a heterogeneous population of individuals where agents can represent any type of unit, from which intuitive groups of units can be formed, creating the population design from the bottom up.

Topological complexity: The topology of agent interactions is heterogeneous and complex. Aggregate flow equations usually assume global homogeneous mixing, but the topology of an interaction network can lead to significant deviations from predicted aggregate behaviour. This is particularly significant concerning social processes where physical or social networks have great influence; it can also result from complex behaviour of the agents, including the ability of learning and adaptation.

Appropriate model framework: Quite often using the agent-based approach for describing and simulating a system composed of real-world entities seems the most natural and intuitive way as it is more akin to reality than other modelling approaches; this makes ABM the natural choice for simulating people and objects in very realistic ways.

In particular, the agent-based approach is useful when the units of a system are easier to describe than the system itself; this is the case if one of the following situations applies:

- * The individual behaviour cannot clearly be defined through aggregate transition rates (e.g. panic within a fleeing crowd).
- * The individual behaviour is complex.
- * The individual behaviour is stochastic

Flexibility: The agent based approach is very flexible. An agent-based model can be defined within any given system environment. Agents have the ability to move within their environment, in different directions and at different velocities, making the model very flexible in terms of potential variables and parameters that can be

specified. The implementation of agent interactions is by far easier to implement than using mathematics, for example. Agent-based models also provide a robust and flexible framework for tuning the behaviour of agents, their degree of rationality, ability to learn and evolve, and the rules of interaction.

Definition 4: Agent Based Model (ABM)

An Agent Based Model is defined as tuple $\langle A, E \rangle$ where A is a set of agents with $A = \cup a^k$ and $1 \leq k \leq N_{agents}$ and E is the Environment with:

- * s_j^k is the j^{th} set of state variables that is seen by agent k where $1 \leq k \leq N_{agents}$
- * $S^k = \cup s_j^k$ all sets of state variables seen by agent k where $1 \leq k \leq N_{agents}$
- * $S = \cup S^k$ all sets of state variables seen by all agents
- * α_j^k is the j^{th} action done by agent k in response to a set of state variables s_j^k
- * $\Lambda^k = \cup \alpha_j^k$ all actions done by agent k
- * $\Lambda = \cup \Lambda^k$ all actions done by all agents
- * r_i^k , the i^{th} run of agent a^k , is the i^{th} sequence of interleaved $s_0^k, \alpha_0^k, s_1^k, \alpha_1^k, \dots$
- * $R^k = \cup r_i^k$, the set of runs of agent k , where $1 \leq i \leq N_{runs}^k$
- * $R_S^k = R^k$ that ends with an s_j^k
- * $R_\Lambda^k = R^k$ that ends with an α_j^k
- * $R_S = \cup R_S^k$
- * $R_\Lambda = \cup R_\Lambda^k$

Definition 5: Agent

An Agent k is defined as a function $a^k: R_S^k \rightarrow \Lambda^k$.

Definition 6: Environment

An Environment E is defined as tuple $\langle \Sigma, \tau \rangle$ where

- * Σ is the system state
- * $\tau: R_\Lambda \rightarrow \Sigma$ is a state transformer function that changes the system state based on $r_i^k \in R_\Lambda$

It has been shown in several publications that it is possible to find an equivalent discrete event model for any agent based model that conforms to the specification given in the following formal definition of an agent. This implies that every agent that is part of such a system should be able to be integrated in a discrete event

system. Therefore it should be possible to develop a model, where both systems - discrete event and agent based - coexist.

This is of great interest, because the system developed in course of the project done for the TU Vienna contains elements of both worlds: the main layout of the TU Vienna can be modelled using discrete event simulation, especially regarding the modelling of rooms. Students enter a room through doors, queuing occurs as soon as the number of people trying to enter or leave the room becomes too big. This resembles the classic server queue problem represented by the traditional post-office-problem. But students are not simple entities, waiting in a queue to be processed and routed through the system. Their individual behaviour is much more complex than that and the overall behaviour of all students on the TU Vienna can hardly be modelled by using the classic approach of discrete event simulation alone. Combining the individuality of the student with the classic discrete event driven behaviour of the environment seems the logical solution.

Chapter 2: Database Controlled Assembling of a Simulation Model

2.1 From Model to Simulation

To implement a Model in a computer program there are several options:

General Programming Languages as C or C# can be used to develop a computer model. Some of these languages offer certain libraries that contain basic functions as random variables or distribution functions.

Simulation Languages usually provide a number of functions that are typical for developing a discrete simulation model i.e. probability functions.

Discrete Simulators usually offer a graphical oriented environment for creating models from predefined elements like sources and queues that are contained in libraries; the simulation engine includes some kind of time handling like an event list or event chains.

Specialized Discrete Simulators are already designed to be applied within a certain range of applications.

Special Simulators are already highly specialized for the implementation of certain applications. The software package that will be later on discussed in more detail for example, Enterprises Dynamics offers different suites for the simulation of logistic or airports. These suites already contain the basic elements that are typical for this kind of application. This keeps development time down to a minimum.

Problem Specific Solutions are Simulators, developed to solve a certain Problem.

It is obvious that the higher a simulation program is situated in the pyramid shown in

Figure 2-1, the higher its specialization in a certain area becomes; and concordantly the higher becomes the user friendliness in regard to the needed time and effort in developing a model. But as this increases another aspect gets lost: flexibility. The more a simulator is designed to solve a very specific kind of problem, the more difficult it becomes to use it for anything that does not fit into the same scheme.

The lower a simulation program remains in the pyramid the higher the flexibility in developing a simulation model is. But as flexibility increases so does the time that needs to be invested in the development of a model.

So, choosing the right balance of flexibility and user friendliness is a task not to be underestimated.

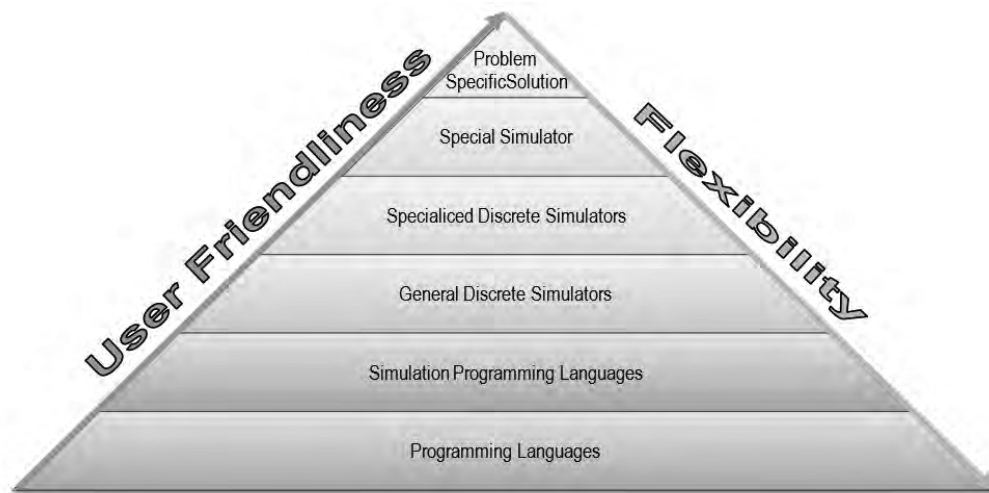


Figure 2-1 Flexibility contra User Friendliness

There are two aspects that make the application of simulation on an everyday basis more difficult than it might appear:

- * The gap between need and knowledge: Expert knowledge is required to develop an accurate and valid simulation model and it is required to modify it as well. But more often than not this is not existent in the average company as they are not familiar with the concept of modelling and simulation and have no prior knowledge of using a simulation tool. A simulation model is usually developed by external specialists; the main interest for the company is not the simulation itself but the results calculated.
- * The time and effort needed to redesign a model: small changes to a simulation model may not seem too complicated to incorporate; but the more complex a model is, the more time consuming even small variations of parameters may become. If changes to the structure are necessary it can quickly become a difficult task to modify the model, especially if several different approaches to such changes are to be compared.

During the last years the ability of a simulation system to be quickly and easily modified has become more and more apparent due to different reasons: The comparison and analysis of different approaches and systems has been one of them, the reusability of a simulation model another.

2.2 Basic Idea and Concept

A first step to offering a higher flexibility has been to allow the changing of parameters within certain boundaries to observe the effect on the whole system. The next step would be offering a possibility to change parts of the simulation without actually needing to go down to the programming level. And finally the last development would be a simulation that is based upon certain structural parameters that define the whole system. Using these parameters the complete simulation build-up process can be automated according to the current parameters. This approach aims to keep the effort for the step between Model and Computer Simulation as shown in Figure 7-1 to a minimum by automating it.

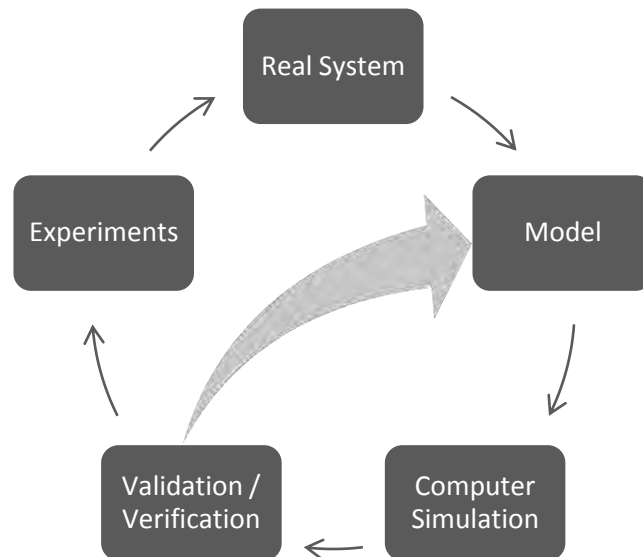


Figure 2-2 : Life Cycle of a Simulation Model

The basic idea is to separate the basic model from the scenario specific data. Keeping the data outside of the simulation environment e. g. in a database offers several advantages:

- * The data is much easier to manage
- * The data is much easier to modify
- * The simulation model is less burdened with storing the data during the simulation run; this is especially noticeable with result data

The data needs to be structured to represent a model of the simulated system that contains all possible combinations of parameters.

The interaction with the database can be easily done via a GUI, hiding the simulation environment as well. Additionally this GUI can be used to start a simulation run and to analyze the results.

This approach implies the split between a simulation environment and a data model: The simulation environment contains the basic model elements as well as the functions to create the simulation model according to the data in the external database.

The data model contains all additional information needed.

To ensure a high flexibility the simulation implemented in Enterprise Dynamics is completely controlled by the data stored in an ACCESS database. Depending on this data the model structure is formed, the single components are connected and parameterized.

The database contains all needed specifications. A list containing all of the scenarios stored in the database enables the user to choose which ones to simulate. A function implemented in Enterprise Dynamics allows exporting the result data back to the database before starting the next simulation run.

The outcome is no longer a simulation, but a highly specialized simulator, that is able to generate any scenario consisting of the specified components; it only has to be defined in the database. Using it to simulate places where one could not be sure if they would be able to deal with the additional customers allowed a better assessment of the future situation, as well as an easy way to evaluate the effect of changes.

2.2.1 Interface to External Data Source

To improve the speed of simulation runs it is often recommendable to keep as much data as possible out of the simulation environment and in external databases. This also enhances the user friendliness as these databases are much more convenient to handle.

Several projects done during the last years have shown the tendency to hide the simulation environment itself from the user behind an easy to manage General User Interface (GUI) to offer a more comfortable way to parameterize the simulation model.

Enterprise Dynamics offers the possibility to use ODBC connections to an external data source or ADO to communicate with a database as ACCESS or ORACLE.

The more complex a simulation model becomes and the more complex therefore the parameterization becomes the more complex the user interface becomes and entering information is a task that is most of the time more time demanding than the simulation run itself.

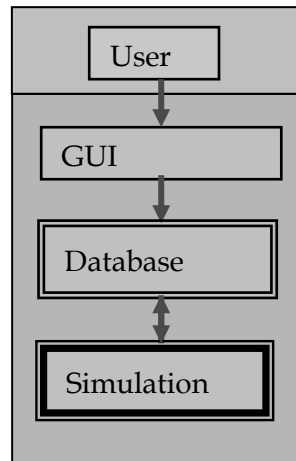


Figure 2-3 User Interaction via GUI

2.2.2 Automatic Model Generation

To offer a possibility of quickly creating and modifying simulation models the approach of database driven model generation was developed. Basically this is done by splitting the simulation model specific data and the basic model to enhance the flexibility of the simulation model.

The input data needs to be structured to represent a model of the simulated system that contains all possible combinations of parameters.

The interaction with the database can be easily done via a GUI, hiding the simulation environment as well. Additionally this GUI can be used to start a simulation run and to analyze the results.

This approach implies the split between a simulation environment and a data model: The simulation environment contains the basic model elements as well as the functions to create the simulation model according to the data in the external database.

The data model contains all additional information needed.

It might not even be known what the maximum number of each component might be so it is not possible to create a pool activated on need as was done for RHI. We no longer have a model with a certain number of components; all we know is the basic structure. Here we need to take full advantage of the object oriented approach of ED.

2.3 Simulation Database

The model has to be well defined in order to be created accurately. So not only each component of the model is defined via its parameters but also the simulation model itself.

The simulation database contains the information needed to create all possible scenarios. The according data model implemented in the external data source, like a database or a spreadsheet reflects the basic structure of the atomic models used to build the simulation model architecture. This data source is then used to specify the scenarios to be simulated.

2.3.1 System Elements

The Data Model contains all elements that may exist in the simulated system. If they exist in a certain scenario depends on the according system structure parameters.

2.3.2 System Structure Parameters

These parameters define the structure of the simulation model in a certain scenario: of which elements it will exist, how many of them and how they are linked.

2.3.3 System Parameters

These Parameters define the state of the system: Parameters for each component are stored in a database, allowing easy and quick editing. With this data the behaviour of each single component is well defined.

The model has to be well defined in order to be created accurately. So not only each component of the model is defined via its parameters but also the simulation itself.

2.3.4 Input Parameters

These parameters are relevant for the simulation run; they contain all information that is not directly linked to a certain element.

e.g.: the length of the simulation run

2.4 Simulation Environment

2.4.1 The Template

Before the simulation model is created several actions have to be taken. These are executed by functions embedded in the simulation model template.

Such actions are the import of the data from the data model; the creation of the simulation model, the start of a simulation run, the processing of the result data. All these actions can be automatically triggered if a simulation run is started.

2.4.2 The Simulation Library

The simulation Library contains all elements that are defined in the data model. Each has its own behaviour that can be influenced by the parameters given in the data model. The functions creating the simulation model will use these elements and set their individual parameters after the model build-up.

2.4.2.1 Classes and Library

All components of the model are divided into classes with certain parameters: and a library is created containing these classes.

Each class already contains the full functionality and behaviour, for example the class Intermediate Storage has its order strategy as well as its delivery behaviour fully implemented. Each object created from this class inherits this functionality; parameters like storage size, minimum storage, delivery interval are individual for each object. This data needs to be defined.

We usually have a complex web of product flow and message flow as well as a huge amount of parameters. It seems only logical to once more use an external source for editing the data in a more manageable way.

The library of classes allows an easy creating of objects but building a simulation consisting of several hundred objects, connecting them without making any errors quickly becomes a tiring as well as extremely complex task.

So the idea to expand the functions we used before for parameterising even further quickly comes to mind.

Each element already contains the full functionality and behaviour, every model element created inherits this functionality. For example each kind of cash has its parameters like cycle time, opening times; they are individual for each object. This data needs to be defined.

2.4.3 Functions

To generate the final simulation objects are built from the classes in the library, their individual behaviour being influenced by their parameters.

The basic simulation only consists of functions that:

- * Import the data from an external source
- * Create the needed components from the library
- * Parameterize them
- * Connect them
- * Reset and initialize the system

The data has to be imported into the simulation, objects have to be created and parameterized according to this.

As soon as all components of the simulation are created we have the same situation as in the prior case: These objects need to be connected, parameterized and initialized. This is done as before using functions.

Now we have a simulation, automatically created basing on the data provided. Changing this data causes a change to the model; therefore the simulation is newly generated.

This offers a great flexibility. Once the simulation is implemented scenarios can be tested with hardly any limits to the range of parameters.

Of course, creating the basic simulation, programming the functions that in turn will create the final simulation is more intricate for everything has to be done completely generic to ensure the adaptability to any scenario. To generate the final simulation objects are built from the elements in the library, their individual behaviour being influenced by their parameters.

2.4.4 The Simulation Model

The Simulation Model is the end result of executing the functions for the model build-up and represents the structure defined in the chosen scenario that is based on the data model. All elements have their predefined behaviour in dependency from their parameters as specified in the data model.

The System State is defined by the System Parameters; the simulation run is executed according to the input data received from the data model.

2.5 Result Aggregation

Most Simulation Tools as Enterprise Dynamics do offer some basic routines for analyzing result data. But as soon as results are used to compare different scenarios or model approaches data needs to be stored outside of the simulation tool.

The results generated in the simulation run are automatically stored for further processing. It is possible to add the functionality to automatically create and simulate a new scenario if certain results are not within given boundaries. The parameters of the new scenario are set according to predefined rules – i.e. if the average waiting time in a queue is to long an additional production line is added to the system.

2.6 Effects on Verification and Validation

The automation of the process of building the simulation model has one additional effect: the process of verification and validation should become easier to handle at least after the basic model has been developed. Using the concept of automatic

generation of simulation models in simulation projects recommends using the basic concepts of software development, especially in regard of testing.

The basic functions for creating the model elements and setup of the model structure can be tested for working correctly – the result of executing them should be the correct setup of the model, according to the model structure and parameters given in the data model. Once these functions are verified there is no further need to test them again as long as no changes are done.

The model elements as they are listed in the project specific library can and must be validated by using the basic model, e.g. by using historical data. So the main work for validation lies in the data model.

Chapter 3: MoreSpace

3.1 History of the Vienna University of Technology

The Vienna University of Technology (TU Vienna) was founded 1815 as Imperial and Royal Polytechnical Institute. It was the first University of Technology within German speaking Europe. The first director, Johann Josef Ritter von Prechtel, took the Ecole Polytechnique in Paris as an example and developed an Organisational Charter with non military orientation for the Viennese Institute. The concept of Prechtel's project who was a liberal advance thinker, a humanist and pedagogue was quite revolutionary: It offered freedom of teaching and learning for professors and students far beyond the usual scope of education for these times.

On November 6th 1815 with the ceremonial inauguration of the k. k. Polytechnische Institut opened its doors and commenced its academic activities.

1815	Foundation as the Imperial and Royal Polytechnical Institute (k. k. Polytechnisches Institut)
1865	Restructure: 5 specialised colleges (1928: faculties) Rectorate
1872	Renamed to College of Technology
1902	First Doctorates awarded
1919	Admission of Women
1975	Renamed to University of Technology
2004	Autonomy
2006	Start of 'Univercity 2015'

Nowadays the Vienna University of Technology is the largest research and educational institution in Austria in the fields of technology and natural sciences. The TU Vienna offers a wide range of studies at its eight faculties:

- * Faculty of Mathematics and Geo-information
- * Faculty of Physics
- * Faculty of Chemistry
- * Faculty of Informatics
- * Faculty of Electrical Engineering and Information Technology
- * Faculty of Mechanical Engineering
- * Faculty of Civil Engineering
- * Faculty of Architecture and Regional Planning

2006 the idea was born to relocate the campus of the Technical University of Vienna to an area outside of the city that offered more space for a university campus. This launched a long discussion and evaluation of the pros and contras until finally around ninety percent of those affiliated to the TU Vienna voted to remain in the current inner-city location; the decision was made to keep the historical location at the centre of Vienna. But almost two hundred years of research and teaching have left their marks on much of the buildings of the TU Vienna. To keep the current location but meet the standards of a 21st century university establishment measures had to be taken. Instead of moving the Technical University started to renovate and restructure the existing buildings to make better use of the existing space. The project Univercity 2015 took on this task with the aim to finish until the 200 years celebration in 2015.

3.2 Univercity 2015

Based on its autonomous decision to remain at its current city location, the Vienna University of Technology, the largest research and education institution in the fields of technology and natural sciences in Austria, has initiated the project "TU Univercity 2015." The main features of the project are the building projects that will create new research and teachings competencies, and the impetus to further develop a self-conscious, future-oriented university culture.

"TU Univercity 2015" is much more than a restoration project. Totally new and comprehensive features must be established by the 200-year anniversary in 2015. The top project goal is the creation of optimal basic conditions for those who study and work at the TU Vienna. Therefore, the main points of the project have been set up on topics such as accessibility, sustainability, art/culture, and corporate identity.

Firstly, an entire existing university is redesigned. The following measures are envisioned: an innovative information and signage system, communication areas, more attractive spaces (also for sports or recreational activities), appealing and inviting entrances and passages, as well as an increased uniformity of the area (keyword: campus).

Quality standards were developed for all space types - offices, various labs, workshops, auditoriums, seminar rooms, library, and so forth. All spaces must be

optimally equipped and should offer flexible functionality as much as possible. The standards are effective for the entire TU and should also be of high quality. The equipment updating periods were defined, and an “intelligent” internal technology is deployed

TU spends 30 million Euro annually for its space. Therefore, each space that is wrongly used costs cold cash. In other words, the resources must be used efficiently! The space allocation must be based on current and future needs. This includes a new assessment of the “grown” structures. International benchmark figures, current usage, and development plans are all involved in this process. The unified key figures model for spaces has field-specific space profiles and includes an evaluation of employees and students. This leads to a new equitable space allocation. The university equally provides flexible, multi-functional space structures, as well as manages auditoriums and seminary spaces and a transparent space allocation.

3.3 TU Campus

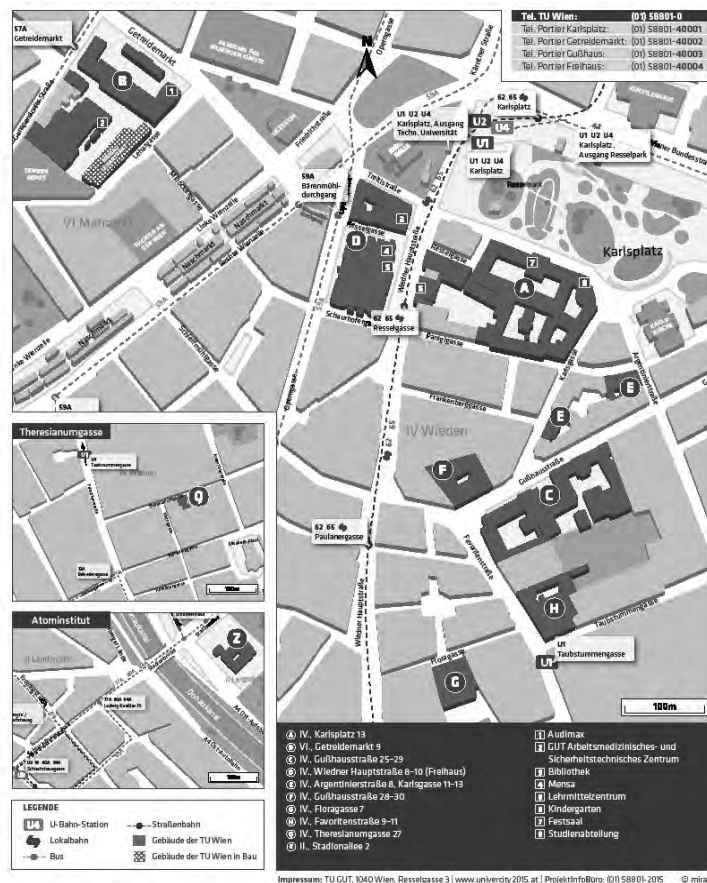


Figure 3-1: Map of the TU Campus

Due to its inner-city location the space of the TU Vienna is largely restricted. The old buildings are part of the UNESCO international cultural heritage which enforces their preservation and puts a lot of pressure upon any renovation and redecoration plans. The newer buildings as Freihaus or the new part of the Electrotechnical Institute (EI) are not so easily rebuild either.

3.4 Problems with Room Management at TU Vienna

The general feeling on the TU Vienna can be summarised as: there is never a room available if one is needed, especially on short notice. This leads to the firm belief, that there simply is not enough room available. But if you walk around and look inside, quite often rooms that are marked as booked at this time in the TUWIS System are unoccupied.

This situation has been analysed by the working group of Gebäude und Technik (GuT) of the TU Vienna who developed a calculation model based on the courses and the number of student attending to derive the real requirement of lecture halls for each of the eight faculties at the TU Vienna. And this calculation showed that theoretically there is more than enough space available. So obviously the real problem lies somewhere else.

For a deeper analysis a dynamic calculation was considered and a cooperation with the institute for Städtebau-, Landschaftsarchitektur und Entwerfen, represented by Univ. Prof. DI Dietmar Wiegand, and the Institute for Analysis and Scientific Computing, represented by a.O.Univ.Prof. DI Dr.techn. Felix Breitenacker was initiated to develop a tool for the dynamic simulation of the booking and utilization of lecture rooms.

The first step was a feasibility study done in 2007, based on the faculties of Architecture and Maschinenbau. The data needed was provided by the Zentralen Informatik Dienst der TU Vienna (ZID). It contained the information about courses, their dates and the number of students that passed the according exams. The study was a success, confirming the GuT results and leading to the development of MoreSpace, a tool designed to simulate the whole TU campus.

This dissertation will focus on the simulation tool that was developed and give a detailed description of the simulation approach used as well as the different algorithms that were implemented to offer a broad spectrum of possible scenarios to experiment with.



Figure 3-2 Original Building on Karlsplatz

3.5 Definition of Terms

To avoid any misconception several terms used in describing the MoreSpace project are shortly explained here.

3.5.1 Utilisation of Rooms

This term indicates the percentage of the time the room is booked over the semester.

3.5.2 Capacity Utilisation

This term indicates how well is the capacity of the room used, respectively how many people are in the room when it is occupied.

3.5.3 Travelling Time

This means the amount of time it takes a student to walk from one room to the other.

3.5.4 Clearance Time

This term indicates the time it takes to leave a room – of course this depends on the number of people currently in the room as well as the size and location of the exit,

3.5.5 Not Successfully Booked Lectures

The booking procedure tries to assign a room to every lecture that is planned. If it proves to be impossible to assign any room to a certain lecture this lecture is marked as not successfully booked.

3.5.6 Compulsory Lecture

Lectures that are mandatory for a certain field of study are called compulsory lectures.

3.5.7 Elective lecture

During the master study the student is able to choose between several lectures that are offered at the TU to focus their interest on a special field within their field of study. These lectures are called elective lectures.

3.5.8 Additional Lectures

Additional to compulsory and elective lectures the student has to choose a certain number of lectures. These he can choose freely

3.6 Intended Purpose of MoreSpace

The project *MoreSpace* was launched to develop a software tool to support the planning phase of “University2015”. “University2015” is a project of the Vienna University of Technology (TU Vienna) to renovate all university buildings and to improve the existing infrastructure and the inherent processes. This shall also be done by determining and evaluating the (spatial) resources required.

The project was launched to assist the department of Gebäude und Technik at the TU Vienna during the planning phase of University 2015. The team responsible for this project used static methods to calculate an assessment of the number of square meters each faculty of the TU Vienna needs for lectures and other student related activities. This calculation was based on the number of students that took exams and the hours of teaching for each faculty. The resulting numbers did show the need of square meters required to accommodate the number of student during the lectures but did not take into consideration how these square meters are used over time. One thing that complicates things considerably is the fact that lectures at the TU Vienna do not take place one after the other in a strict pattern as it is done at schools or colleges but are set at times favoured by the lecturer. This results in a weekly timetable for the student that contains times where several lectures may be very close to each other, even overlapping, and times where no lectures at all are taking place, leaving a big time gap. Over the whole of the TU Vienna one can say that

there are certain times during the week that are more or less preferred by many lecturers, resulting in a high demand of rooms during certain time periods, where other times – i.e. Friday afternoons – are not that popular. The fact that the required amount of square meters is available, does not mean it is used in a way that ensures that it will be indeed possible to acquire a room for a lecture at a certain time.

The idea to develop a simulation was born, a tool to reproduce the situation as it currently is concerning the lectures and their demand on room and experiment with the room structure. The basic idea was to keep the lectures exactly as they were and test if the new room situation was able to accommodate all of them.

In 2008 the working group consisting of the Research Group for Mathematical Modelling and Simulation and the Research Group for Real Estate Development and Management at the Vienna University of Technology were asked by GUT TU, co-responsible for Project Management of “Univercity 2015”, to introduce a model for the room management that can simulate the usage of resources to optimize the planning of the rooms and the future usage. Figure 3-3 shows the effect simple changes to the space management can have: If the strategy is used to increase the utilisation of fewer rooms it automatically offers larger continuous timeslots in the remaining rooms.

	Room 1	Room 2	Room 3
10:00			
11:00			
12:00			
13:00			
14:00			
15:00			
16:00			
17:00			
18:00			

	Room 1	Room 2	Room 3
10:00			
11:00			
12:00			
13:00			
14:00			
15:00			
16:00			
17:00			
18:00			

The diagram shows a second table identical to the first, but with curved arrows indicating a shift in room usage. A large arrow points from Room 1 (12:00-16:00) to Room 2 (12:00-16:00), and another large arrow points from Room 2 (12:00-16:00) to Room 3 (12:00-16:00). This indicates that the utilization of Room 1 is reduced, and the freed-up space is used by Room 2 and Room 3, resulting in larger continuous timeslots for those rooms.

Figure 3-3 Keeping larger Timeslots

The first step to developing MoreSpace was to use the data of the faculty of Architecture and Raumplanung and the faculty of Bauingenieure and test if the main building on Karlsplatz as it was planned would be able to hold all lectures of these two faculties.

To be able to experiment with the room structure was quickly advanced by the possibility to use different strategies for the space management. It does make a big difference in what order the list of demands for a room is processed.

This shows very nicely the analogy to the classical application of DEVS, the simulation of logistic processes in production lines or manufacturing facilities where the order of processing may become crucial for production times and the feasibility of manufacturing certain quotas in time. Here simulation is one of the most evident ways of analysing and enhancing the existing system.

The classical discrete event simulator Enterprise Dynamics was chosen to develop the dynamic simulation including features like different possibilities of room selection, different management of the resources, variability of classes, class structures and number of students - only to mention a few of the features. This discrete dynamic simulator was combined with - and is guided by - methods and procedures of the real estate management like business process models.

In the course of developing the first studies the working group identified two main problems that are different to tasks which are normally solved by classical discrete event models. On one hand the interfaces to data collection and booking system had to be improved and standardized. Booking features and also the representation of data was not sufficiently optimized neither to the needs of the simulation tool that has to be implemented nor to the needs of the future booking system that has to be simulated. On the other hand the buildings of the Vienna University of Technology - as the university will not move out of the city of Vienna - remain very scattered over a few districts of the city. Therefore a simulation of room booking and facility management has to be aware the problem of differently structured buildings within the university and therefore consider travelling times, sometimes even between classrooms within the same building.

3.6.1 Aide for Booking Courses

During its development MoreSpace did attract the attention of several people working with the problem of assigning rooms to lectures. Especially the challenge of finding a room on short notice quite often causes discontentment and the fact that one might not find an available room via TUWIS++ but taking a good look around shows several empty rooms that should be occupied according to TUWIS++.

This situation makes it annoyingly difficult to book rooms that are not only available but also best equipped and comfortably close.

MoreSpace has been developed not only to show that technically more than enough room is available but also to show a way to improve the current situation. The department that is responsible for creating the plan of lectures for each semester and

for assigning rooms to these lectures has of course taken quite some interest in the development of MoreSpace and has contributed to the integration of different strategies in several meetings and discussions. The future plan here is to use MoreSpace as an aide for their booking organisation to compare different strategies.

3.7 Benefits

The benefit of using a simulation tool in general is that ways of enhancement can be tested without risk to the real system. The same can be said about MoreSpace – strategies for booking can be experimented with, in scenarios rooms can be closed, added, joined or redecorated with no risk or costs.

Results can be achieved very quickly; the comparison of different scenarios can be easily done.

Due to the GUI used for parameterising and the connection to databases for the data exchange the usage of the simulation tool should be as comfortable as possible.

MoreSpace has been designed to enable the calculation of different scenarios to evaluate and compare different booking strategies but also to test future developments in a safe environment.

This may be changes to

- * Student numbers – increase of student numbers => more demand of room
- * Room structure – rooms may be out of use for longer time periods due to renovation or technical problems
- * Courses – the different fields of studies may be reduced or new ones may be added
- * Handling of courses – blocking of courses becomes more and more popular

Chapter 4: MoreSpace: Model Description

The MoreSpace model actually simulates two different but interlaced things: the booking/assignment of rooms for lectures and the dynamic behaviour of students attending these lectures and the resulting utilisation of space. Figure 4-1 shows the steps MoreSpace executes: first is the collection of data from several different sources:

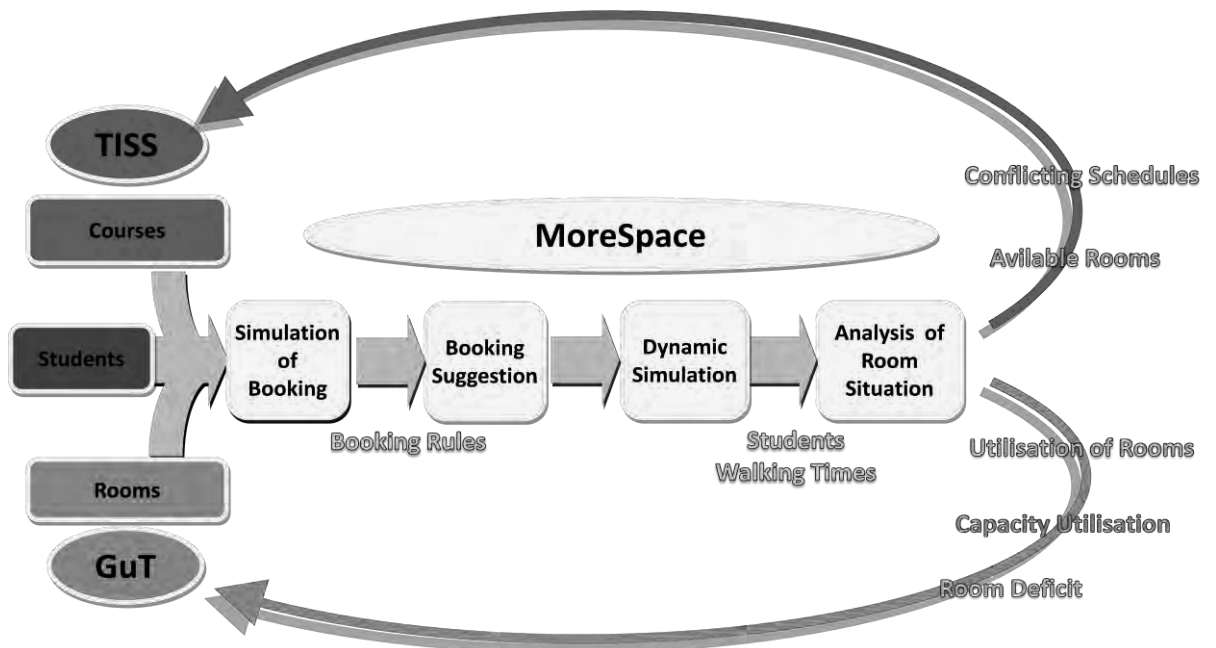


Figure 4-1 MoreSpace

The MoreSpace simulation model uses three different sources for the data basis of a simulation run: the information about courses is acquired from TUWIS++ respectively its successor TISS; all information regarding the lecture halls and their

status is obtained via the database of GuT; information about the number of students attending a course are estimated based on historical data as well as actual trends.

The MoreSpace simulation uses this data for calculating a suggestion for the assignment of rooms according to the selected booking management rules. This leads to successful and not successful booking attempts, both which are part of the simulation result. Other results are i.e. the expected utilisation of rooms.

This suggestion is used as a basis for the dynamic simulation of the semester that includes the movement of students through the system. Students are simulated as single agents with their own 'intelligence'; they are able to make decisions i.e. which lecture to attend if two lectures overlap. The total number of students attending a lecture is not really known, it is estimated based on the historical number of students that have visited this course during the years before as well as an additional factor that can be manipulated by the MoreSpace user. During the course of the semester they attend their lectures and move through the university campus, causing the dynamic behaviour of the system. This delivers additional results like the capacity utilisation of the rooms - the number of students attending the lectures held in a room opposed to the capacity of the room. This gives a greater understanding of the real demand on room compared to the available space. Also a result of the dynamic simulation based on the student - agents is the accessibility of lectures - spatial as well as temporal and the utilisation of rooms. Accessibility is of great interest for it takes the size and layout of the TU Campus into consideration. This is one of the major advantages of MoreSpace: it does not stop at the best possible assignment of rooms but also considers the individual problems that may occur for students to be able to attend a lecture.

The model is designed to allow three different angles of experimenting:

- * The room structure containing the type of rooms and their capacity and location,
- * The list of courses held with their time and expected number of students and the required setup of the room
- * The booking management used for the assignment of rooms to the single lectures of each course

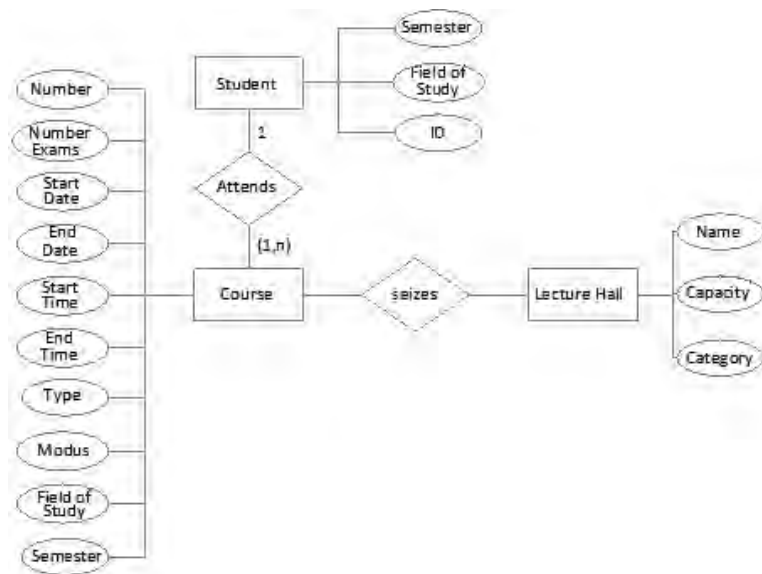


Figure 4-2 Entity Relationship Model

4.1 The MoreSpace Model from DEVS Point of View

As shown in Figure 4-2 the basic elements of the MoreSpace model are room, course and student. The course is a passive object that is assigned to a student but has no behaviour at all.

This leaves the room and the student. The room can be split into several small models, representing a coupled model consisting of four atomic models as shown in Figure 4-3:

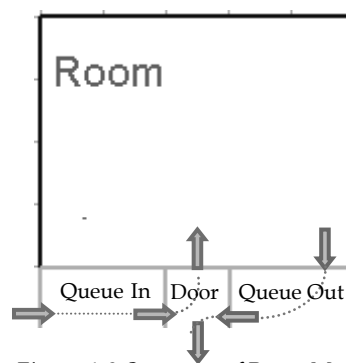


Figure 4-3 Structure of Room Model

4.1.1 Queue In and Queue Out:

The queues Q_{in} , Q_{out} in front of the door are simple FIFO buffers. Using the DEVS formalism the definition for the atomic model representing both of them is:

$Q_{in} = Q_{out} = (X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$, with

- * $\Sigma = \{0, 1, 2, \dots\}$
- * $X = \{1, -1\}$
- * $Y = \{0, 1\}$
- * $ta(x) = \begin{cases} \infty & \text{if } x = 1 \\ 0 & \text{if } x = -1 \end{cases}$
- * $\delta_{int}(s) = s - 1$
- * $\delta_{ext}(s, x) = \max(0, s + x)$
- * $\lambda(s) = \begin{cases} 0 & \text{if } s = 0 \\ 1 & \text{if } s > 0 \end{cases}$

4.1.2 Door

The door leading to a room is represented by a server atom in ED; the DEVS formalism describes it as follows:

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

- * $\Sigma = \{0, 1\}$
- * $X = \{1\}$
- * $Y = \{1\}$
- * $ta(x) = \text{cycletime}$
- * $\delta_{int}(s) = 0$
- * $\delta_{ext}(s, x) = 1$
- * $\lambda(s) = 1$

4.1.3 Room

The room has no other functionality than taking count of the number of people entering and leaving over time.

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

- * $\Sigma = \{0, 1, \dots\}$
- * $X = \{1, -1\}$
- * $Y = \{\}$
- * $ta(x) = \infty$
- * $\delta_{int}(s) = 0$
- * $\delta_{ext}(s, x) = s + x$
- * $\lambda(s) = 0$

4.1.4 The Student: A Double Agent?

The student is considered an atomic model with the behaviour of an agent. It is a member of both worlds, DEVS as well as ABM because it still interacts with the DEVS environment as a regular input and output of the DEVS system represented by Queues and Server.

Considering the definition of a run r_j^k an action α_j^k takes place in response for every state s_j^k the student takes in.

S^k is the set of all state variables an agent a^k can see. The actions as well as the change to the state variables have to be defined by using the structure of the atomic model.

For the interaction with the discrete event simulation system the agent has to act as an input to the DEVS elements. I.e. in ED an external event of a Queue is triggered by an incoming atom.

In case of the student the actions can be listed as:

- * *Decision* if the student will attend the next lecture: depending on their individual time management and their own preferences the students decide which lecture they will attend next.
- * *Start Walking*: the students start to move towards their destination. This can either mean they will 'frozen' in the ED model and transferred to the JAVA model or it starts moving according to the function integrated in ED.
- * *Walk*: the students move from their current location to the assigned lecture room.
- * *Arrive*: The students reach their destination. This means either they are returned from the JAVA model and 'unfrozen' in the ED model, or they have reached their destination according to the function integrated in the ED model.
- * *Enter*: the agents enter the room. That means they enter the Queue_In atom and let the DEVS system take over.
- * *Leave*: either the lecture is over or the students have another lecture with higher priority: that means they enter the Queue_Out atom and let the DEVS system take over.
- * *Attend*: the students have entered the room and attend the lecture

The student atom has several states that result from their activities:

- * *Waiting*: after a decision: if at the current time there is no lecture to attend the student waits until it is time to get ready for the next lecture. Their location is set to somewhere outside the campus and their speed is set to 0.
- * *Attending*: the students attend a lecture and remain in this state until they leave. Position and speed do not change during this state.

- * *Start Moving*: students enter this state if they start walking and remain unfrozen in the ED model. Their speed is changed, their position remains the same. This state is immediately changed to *walking* by the action *walk*.
- * *Walking*: the students move from one point in space to another with a given speed. They remain in this state after they start walking until they arrive. Their position changes, the speed remain the same.
- * *Frozen*: the students remain in this state until they are 'unfrozen'; no events or activities can happen during this state
- * *Arrived*: after the students stop walking their speed is set to 0. If they are 'unfrozen' their position is updated.
- * *Queuing*: as long as the student is moving through the queuing process he remains in this state. Due to the ED atom design the entering and exiting process automatically causes an input to the atom and therefore to the agent atom as well. This causes an update in the agents state

$$\Lambda^k = \{decide, wait, start walking, walk, arrive, enter, attend, leave\}$$

$$S^k = \{position, speed, frozen, walking, queuing, waiting\}$$

The student agent entering the queue atom is treated as the positive input value $x \in X$ and therefore triggers the time advance function and the external translation function, thus the autonomous behaviour of the atomic model 'Queue'. The 'Queue' treats the agent atom just like any other input, resulting in the output value $\lambda(s)$ that causes the student atom to be moved into the next atom, the 'Door'. Here again the agent atom is treated as any regular input, generating the output value $\lambda(s)$ after $ta(x)$ has passed from the 'Door', that causes the agent atom to be positioned in the 'Lecture Room'. Here the control is returned to the agent atom: the lecture room has no further functionality but to count the number of student atoms it contains, it does not generate an output value that influences the agent atom. The next action of the agent atom is completely independent from the activity of the discrete event system elements: the student will leave again at the time either the lecture ends or it has another activity planned that is of higher interest than its current activity. Leaving is the identical procedure: the student agent enters the queue leaving the room, passes through the door and then moves on to its next activity.

Using the concept of the timed events in ED an activity of an agent can be represented by two events set at the beginning of an activity and at their end. The events are triggered at the corresponding time and are used to update the state variables.

The main characteristic of students is their ability to make their own decisions based on their own state and their environment. To model the human decision making in this work the Utility Theory is used: it assumes that the decision process has two elements: the options and the evaluation function, called utility function that maps each option in the choice set to a numerical value.

Definition 7: Utility Function

The function $u: \mathcal{X} \mapsto \mathbb{R}$ is a utility function if \mathcal{X} is the set of choices.
 Preferences of the modelled individuals can be:

- * No preferences (\sim)
- * Prefer the first over the second option ($>$)
- * The second over the first ($<$)

If the preferences observed in the individuals modelled correspond to the relations given by $u(\circ)$, $u(\circ)$ is called a valid utility function for the given decision problem.

In case of the MoreSpace model one decision the student has to take is the case of overlapping lectures. If a lecture x and a lecture y take place at the same time the student has to decide which one to attend.

Basically the utility function for the decision making of the student for overlapping lectures can be derived from the type of lectures the student has to choose between:

$$X = \{\text{compulsory lecture, elective lecture, additional lecture, exam, anything else}\}$$

$$u(x) = \begin{cases} 5, & x = \text{exam} \\ 4, & x = \text{compulsory lecture} \\ 3, & x = \text{elective lecture} \\ 2, & x = \text{additional lecture} \\ 1, & x = \text{anything else} \end{cases}$$

And the preference is assumed to be $>$.

Of course the individual preferences need to be added as well: the option not to go anywhere can be added to model the possibility that a student might not attend any lecture at all. The decision between lectures may be done stochastically.

But a tendency towards one lecture has to be remembered – the decision the next time may depend on the decisions felled in the past.

$$X = \{\text{no lecture, lecture with mandatory attendance, exam,}\}$$

lecture without mandatory attendance}

$$u(x) = \begin{cases} 6, & x = \text{exam} \\ \text{uniform}(5,4), & x = \text{lecture with mandatory attendance that} \\ & \text{has been attended in the past} \\ \text{uniform}(4,3), & x = \text{lecture without mandatory attendance} \\ \text{uniform}(3,2), & x = \text{lecture without mandatory attendance that} \\ & \text{has been attended in the past} \\ \text{uniform}(2,1), & x = \text{lecture without mandatory attendance} \\ \text{uniform}(5,1), & x = \text{no lecture} \end{cases}$$

And the preference is assumed to be $>$.

The decision also depends on the quality of the lecture itself: Are there enough seats? Does the course have a good scriptum? Is it an interesting topic?

$$u(x) = \begin{cases} 10, & x = \text{exam} \\ \text{uniform}(9,8), & x = \text{lecture with mandatory attendance that} \\ & \text{has been attended in the past} \\ \text{uniform}(8,7), & x = \text{lecture with mandatory attendance} \\ \text{uniform}(7,6), & x = \text{lecture with mandatory attendance and} \\ & \text{interesting topic} \\ \text{uniform}(6,5), & x = \text{lecture with mandatory attendance and} \\ & \text{good scriptum} \\ \text{uniform}(5,4), & x = \text{lecture without mandatory attendance that} \\ & \text{has been attended in the past} \\ \text{uniform}(4,3), & x = \text{lecture without mandatory attendance and} \\ & \text{interesting topic} \\ \text{uniform}(3,2), & x = \text{lecture without mandatory attendance} \\ \text{uniform}(2,1), & x = \text{lecture without mandatory attendance and} \\ & \text{good scriptum} \\ \text{uniform}(9,1), & x = \text{no lecture} \end{cases}$$

And the preference is assumed to be $>$.

The depth of the utility function can be easily altered. Right now the behaviour of the students is kept fairly simple to ensure the model works correctly. After validation of the MoreSpace booking simulation the behaviour of the students can be tuned to allow additional analysis of results.

4.2 Room Management

The existing courses can be mapped to different room structures. This enables the user to draw conclusions regarding the utilization and capacity utilization of rooms. Alternative situations can be easily given a try i.e. the effect of additional available space or the blocking or releasing of rooms. It is also possible to cause external events during the simulation that causes a change to the room structure at any given time during the simulated semester.

4.3 Booking Management

The rules that define the way rooms are assigned to lectures are summarized with the term booking management. It includes the order in which the courses are dealt with.

Aim of the simulation is it to study the consequences of different strategies and different conditions regarding the room situation at the TU Vienna.

MoreSpace offers several strategies that can be used to establish the order in which the lectures are booked into the rooms.

4.3.1 Sorting by Date

The list of lectures is sorted by begin date and time.

4.3.2 Sorting by Duration

The list of lectures is sorted by the duration of the lectures beginning with the longest descending.

4.3.3 Sorting by Capacity Demanded

The list of lectures is sorted by the number of students that are expected to attend beginning with the highest value and descending. This should lessen the crowding out effect as lectures demanding big lecture halls are automatically booked first.

4.3.4 Sorting by Category of Lecture

Especially for Master Studies the lectures are divided into compulsory lectures that are mandatory for a student of this field of study and additional lectures that can be chosen from a wide range of offered lectures.

OF course compulsory lectures for bachelor as well as master studies usually should be regarded as of higher priority.

4.3.5 Sorting by Type of Lecture

There are several kinds of lectures: Basic lectures, Laboratory courses, Exercise courses, Practical courses, Seminar courses, etc.

Usually different types also have different demands on the amount of space they require as well as the setup. Therefore it seems reasonable to add some kind of sorting regarding the type of lecture as well.

4.3.6 Sorting by Modus of Lecture

Lectures may be held in different kind of modus: Weekly lectures used to be the most common type: the lecture takes place every week at the same time from the beginning of the semester until the end. This may be once or several times a week, depending on the amount of hours this lecture takes up.

Blocked lectures take place during a few consecutive days, covering the whole semester in a few blocks of several hours. This used to be most common for laboratory exercises but has become more and more employed during the last years.

Single lectures: every week a new time and day is assigned.

4.3.7 Additional Options

Additional options that are considered by the booking routine are: fixed rooms for weekly courses, a classroom concept for bachelor studies where the change of rooms is kept to a minimum and the lectures are consecutively timed, focusing on the best fit of room size where the search is narrowed down to the optimal room regarding its capacity versus the expected number of attending students..

4.3.8 Behaviour of Rooms

It is possible to cause external events during the simulation that causes a change to the room structure at any given time during the simulated semester. This includes the joining or dividing of rooms or the sudden need to close rooms due to technical problems for a certain time period. Of course this would result in the immediate need of reassigning the lectures booked in these rooms during this time period.

4.4 Student Numbers - a General Problem

A general problem in formulating the model was the lack of distinct information regarding the number of students attending each lecture. Due to the lack of mandatory attendance at most lectures the numbers differ quite strongly from the number of passed exams or even the number of enrolled students.

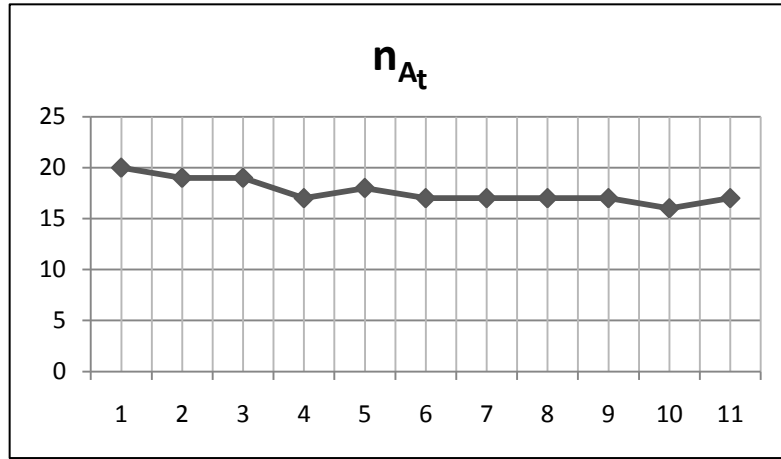


Figure 4-4: Course with Mandatory Attendance

It is a common occurrence that the numbers of students that attend a lecture tend to drop after the initial weeks of a new semester. This effect is especially distinctive observable in lectures without mandatory attendance. The real number of students following the course is mostly not matching the number of students really attending each lecture; the best approximation for this number is n_F , the number of students taking the exam for passing the course. But the number n_{A_t} of students attending a lecture at time t depends on several factors:

- * The perspicuity of the lecturer – the quality of the lecture influences the attendance of students considerably. A lecturer that is hard to understand or whose explications are hard to follow will have fewer students than one who presents his lectures in a comprehensible and interesting way.
- * Of course the topic itself is always a factor as well
- * The quality of the accompanying material: a lecture that offers high quality manuscripts that contain anything the student needs to pass the exam usually lead to a drop in attendance. The strongest factor for attending a lecture after it being mandatory is the need for lecture notes. If they are provided one main factor simply falls away.
- * The importance of the lecture: compulsory lectures are usually highest in priority for they must be done. Some courses are precondition for getting a place in a tutorial
- * Assessment of student's performance: if during the semester several interim tests are done for the assessment of the students this usually leads to a different behaviour: at the date of the tests most students attend the lecture to take part in the test. Right before and after these dates the number raises and drops respectively.

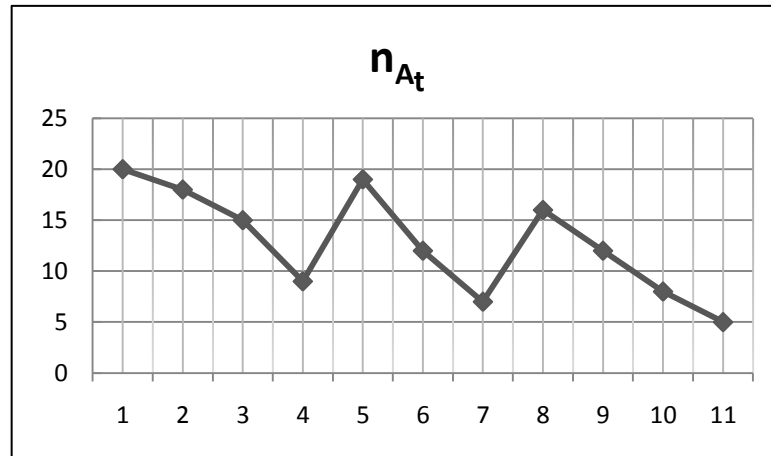


Figure 4-5: Course with Interim Tests

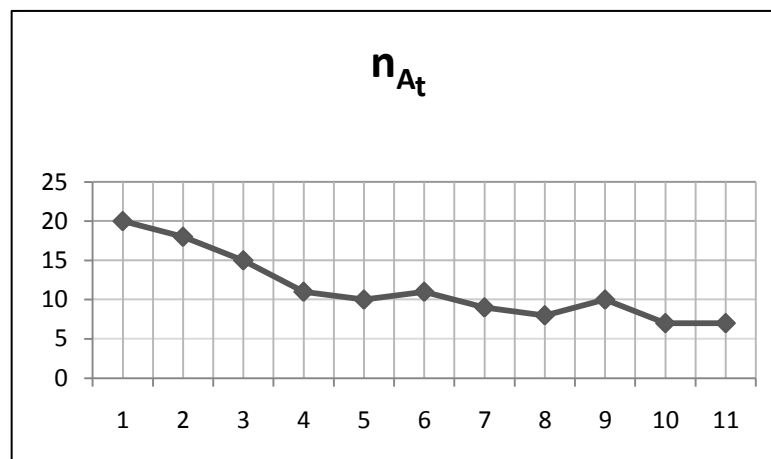


Figure 4-6: Course without Interim Tests

Using the agent based approach did allow to use the known data as basis for the number of students. Students have their own behaviour in attending lectures that recreates the attendance behaviour of the real system.

Known are:

n_E ... Number of enrolled students

n_F ... Number of students finishing the course by passing the exam

Not known are

n_{A_t} ... Number of attending students at time t

n_{D_t} ... Number of drop out students at time t
 n_{N_t} ... Number of not attending students at time t

One may say:

$$n_E \geq n_{A_t} \geq n_F$$

Considering the current situation on the TU Vienna one may even be sure to say:

$$n_E > n_{A_t} \geq n_F$$

$$n_{A_t} = n_F * F_S$$

$$n_F = n_{A_t} + n_{N_t} - n_{D_t}$$

For courses with a mandatory attendance one might say:

$$n_E > n_{A_t} = n_F + (n_{D_{t_E}} - n_{D_t})$$

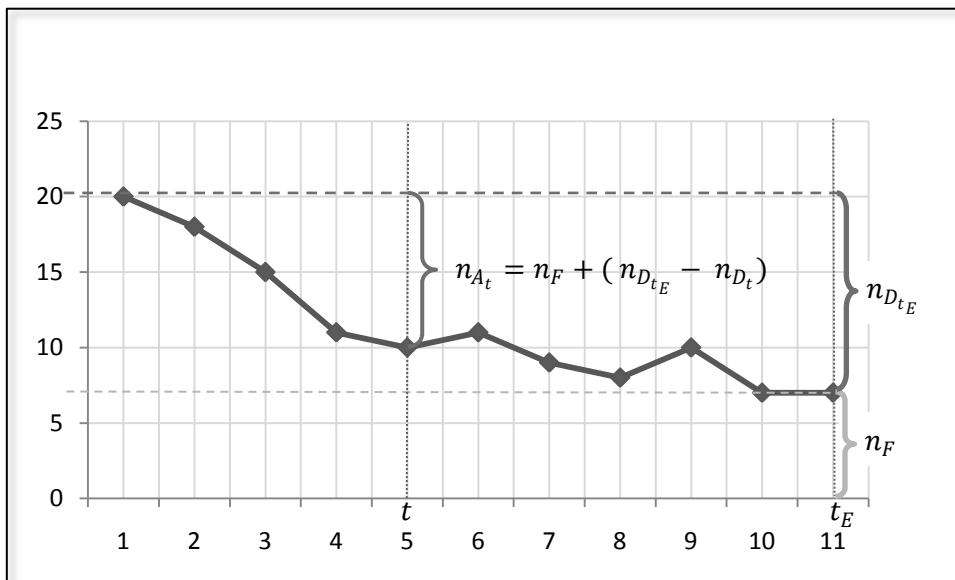


Figure 4-7: Number of Attending Students

4.5 Step I: Simulation Model Assembly

The first step for each simulation run is assembling the simulation model according to the basic input data transferred from the database. For this the Data Import Functions are used as well as some internal tables for storing the data that is used not only during the model assembly but during the simulation run as well. The MoreSpace Library contains all atoms the model may contain; the Model Template offers the functions for the data import and the model build-up.

First all buildings are created, based on the BUILDING atom from the library. For each building its parameters are set to the values defined in the database.

Then the lecture halls are built from the ROOM atom, set into their assigned building and tagged with their individual parameters.

The last point is the compiling of the list of courses, preparing it for the next step, the booking procedure.

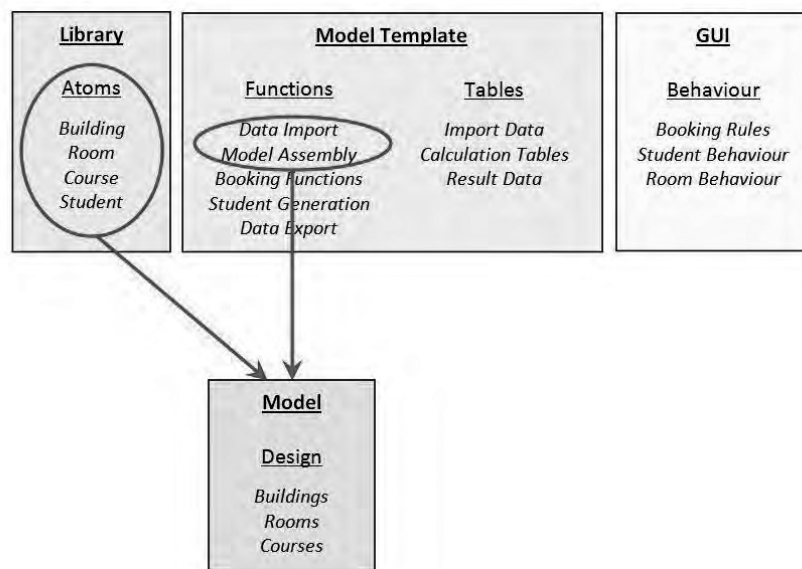


Figure 4-8: Model Assembly

4.6 Step II: Simulation of Booking Procedure

For the simulation of the booking procedure the Model Template, the GUI and the Simulation Model work together: the GUI delivers the input from the user regarding the model behaviour relevant for the booking function. Based on the simulation model design e.g. the number of rooms and the list of courses the booking of the rooms is simulated. The simulation is dominated by the booking rules and the behaviour of the rooms over time. The result is the list of all booked events as well as the list of unsuccessful booking attempts.

From this the utilisation of the rooms can be derived.
 A more detailed description of the booking procedure is given in Chapter 8
 Functional Description.

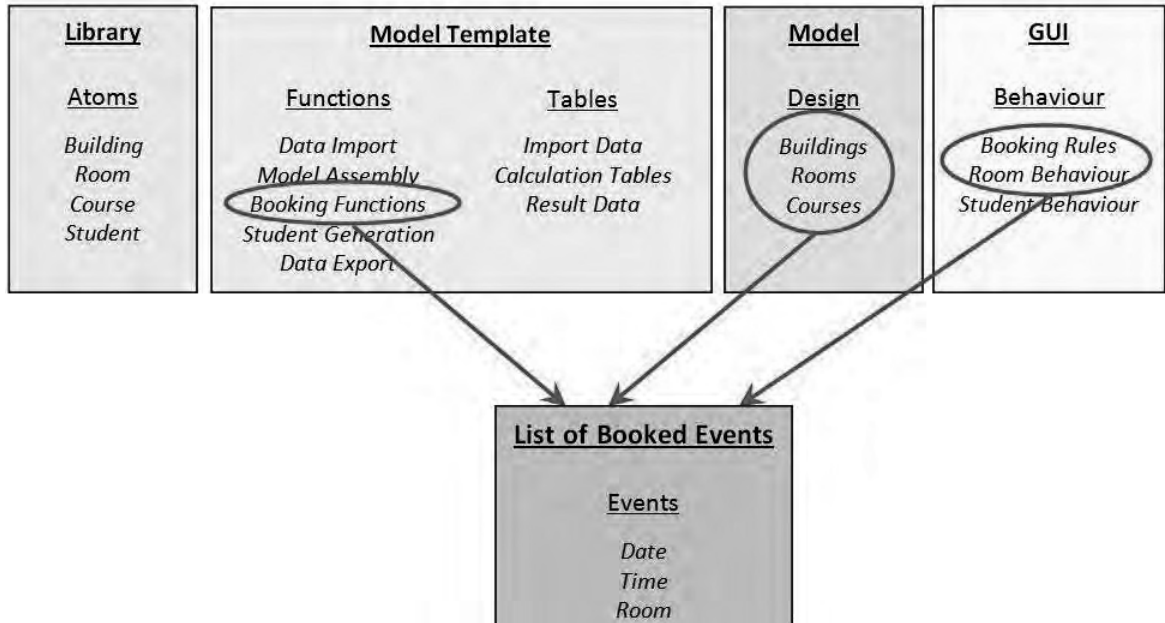


Figure 4-9: Simulation of Booking Procedure

Every Room has the following Parameters:

Room Capacity	C_R
Room Setup	S_R
Room Environment	E_R

Every Course has the following Parameters:

Number of expected Students	N_C
Required Room Setup	S_C
Course Type	T_C
Course Modus	M_C
Required Environment	E_C
Begin Time of the course	T_{BC}
End Time of the course	T_{EC}
Required Environment	E_C
Length of the course	$L_C = T_{EC} - T_{BC}$

4.6.1 Sorting

Sorting of courses may happen according to the type of the course T_C , the modulus of the course M_C , the number of the expected students N_C or the length of the course L_C . The sorting defines the order in which the courses are booked.

4.6.2 Booking Criteria

The booking procedure tries to find a room to each lecture. The attributes of the lecture determine exactly what kind of room can be assigned. Certain criteria have to be fulfilled to enable the assignment of a designated room to the lecture.

Definition 8: Strong Best Fit Criterion

The capacity of the room has to equal the expected number of students:

$$C_R = N_C$$

Definition 9: Weak Best Fit Criterion

The capacity of the room has to be equal or greater than the expected number of students:

$$C_R \geq N_C$$

Definition 10: Optimal Environment Criterion

The environment of the room has to equal the environment of the course:

$$E_R = E_C$$

In other words: the appropriation of lecture halls and rooms to the institute that holds the course has to be taken into consideration during the allocation of a room. All courses of an institute have to take place as far as possible within the lecture halls and rooms assigned to this institute.

Definition 11: Room Setup Criterion

The setup of the room has to accord to the requirement of the lecture:

$$S_R = S_C$$

4.7 Step III: Dynamic Simulation

The dynamic simulation involves the student going through the semester and attending their courses. They allocate space in the assigned room and add to the capacity utilisation of the room.

External Events:

The system state changes during the simulation due to events caused by the elements of the simulation model. There are two additional events that influence the system state but are not triggered by the occurrences within the system:

- * Assignment of rooms to lectures
- * Blocking of rooms during certain time period due to reservation or reconstruction

The booking of rooms for lectures will take place before the actual semester begins, but there will always be a demand of space during the semester as well, resulting in additional room assignment as well as an adjustment in the student behaviour. Additional lectures or other events will cause student to attend them and therefore make them take new or different decisions than they would have before this change happened.

The blocking of a room for a certain time period due to reconstruction or other reasons will lead to a reassignment of all lectures booked in this room. This will also affect the students as they have to be informed about the change in location.

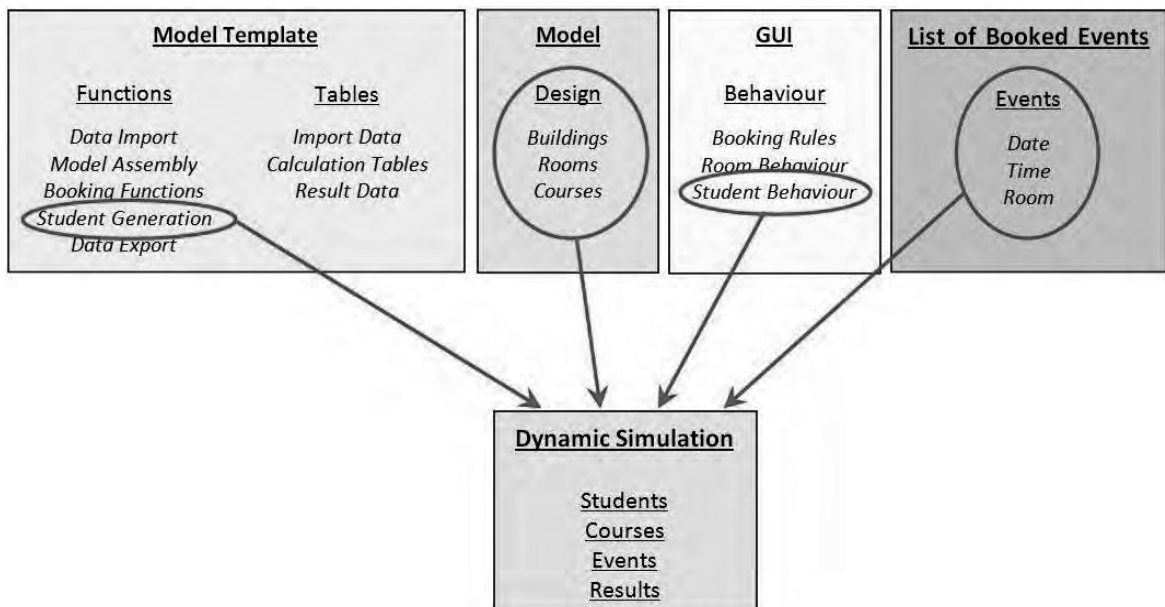


Figure 4-10: Dynamic Simulation

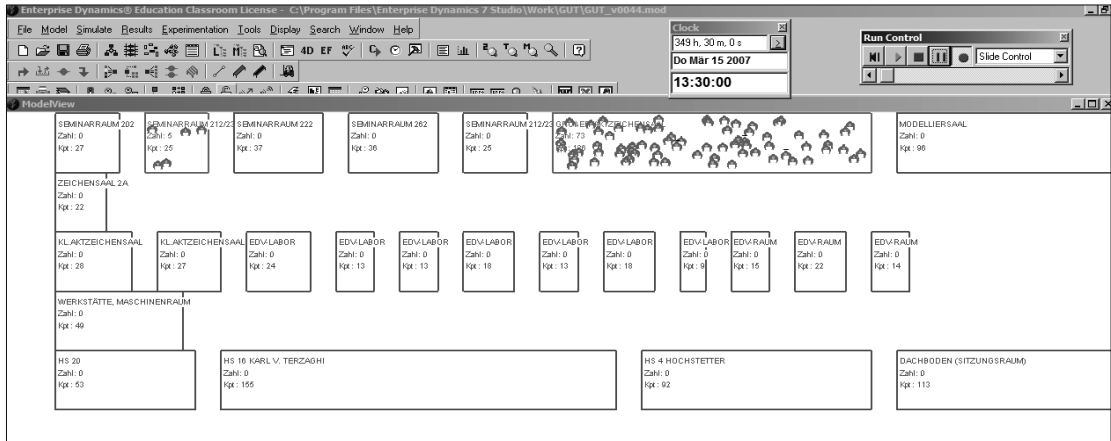


Figure 4-11 Simulation Model in ED

Chapter 5: The Simulation Approach

5.1 Simplify, Simplify! (Henry David Thoreau)

As this simulation solely focuses on the lecture halls of the Technical University of Vienna all other rooms and their usage are not regarded here. All events that take place outside of the lecture halls are not considered and this fact has to be kept in mind at the evaluation of the simulation results.

A main task of creating a simulation of a real system is simplifying the system as it is observed in the real world as much as to reach a model that is only as complex as it is needed to achieve the desired results. In the case of MoreSpace the first focus was on the different rooms available for the lectures and the long list of events that had to be booked into them. Only after working with the input data and analysing it the realisation dawned that one aspect had not yet been taken into account: the students. Just as important as it is to find the best equipped room with the right size for each event is it to ensure that students are able to attend this perfectly planned event. The collision of lectures is a not uncommon problem that forces many a student to decide with lecture he will not attend. This may be due to overlapping times of lectures but even more often it is the time it takes to get from one lecture room to the next. This movement time is often underestimated and causes lectures to interfere with each other although they are not really taking place at the same time.

But enabling students to attend their lectures as unopposed as possible is one important factor for keeping the quality of education on the Technical University of Vienna at its best.

So the focus in the model was also set on the students. In a classic DEVS system the students would have been regarded as entities that are routed through the system, in this case through the rooms and the lectures occurring there. In the case of two overlapping lectures the entity would have always followed its designated path: attend the first lecture until it is finished and then proceed to the next. But students are no mindless entities; they have their own priorities and preferences and usually every student is a unique and to some extent unpredictable person.

So the classical DEVS approach seemed not satisfying and considering a group of people with certain behaviour quickly leads to the most common approach for problems of that kind: Agent Based Modelling.

As already described in section ... Agent based Modelling works with individual 'agents' that have a predefined behaviour that results from an evaluation of their current situation and a set of rules.

According to this students can be considered as agents, their current situation results from their list of lectures they have to attend and the decision of where they will go is derived from the given set of rules. If certain possibilities are entered for certain decisions the individuality of the single persons simulated can be achieved.

The model that has been developed for MoreSpace is a dynamic model that enables the user to have all the available data to their disposal as well as all the knowledge collected during the project time in discussions and tutorials with several people related to the booking system. The main focus is on the comparison the effect of different behaviour in regard of the booking of rooms for lectures may have on the availability and the utilisation of the lecture halls.

The model is designed to ensure the maximum flexibility. Different scenarios can be calculated and compared quite quickly. This includes changes to the room structure, the lecture plans or the rules for the booking management.

Basically it is necessary to strictly differentiate between two aspects of MoreSpace: the model design and the model behaviour.

5.2 Simulation Peoples Movement in ED

Discrete Event Simulation (DEVS) is a widely used approach for modelling and simulation of dynamic discrete systems. The modern object-oriented DEVS world view regards active objects (entities) passing passive objects (stations) along given path. An event mechanism updates movement of entities, being capable also of collisions and other state-dependent phenomena. Consequently; DEVS can be used for simulation of pedestrian and evacuation dynamics. Classical straightforward modelling approaches, based on abstract classical queuing systems, do not take into account the spatial distributions primarily – only for animation a spatial component is used. But modern object-oriented tools allow modelling of a spatial distribution primarily by means of topological attributes as well in active and passive objects at the modelling level. This contribution discusses the "spatial" features of DEVS simulation systems The chosen simulation tool, Enterprise Dynamics, offers a high flexibility due to its programming language 4d script that allows adding further functionality to the provided elements as well as creating completely new ones.

At present there exist a lot of software tools for modelling and simulation of discrete dynamic systems. Almost all tools are based on a modern object-oriented DEVS world view regarding active objects (entities) passing passive objects (stations) along given path. A time event mechanism updates movement of entities, being capable

also of collisions and other state-dependent phenomena. Some of these tools can look back at a long tradition, now offering also any kind of object-oriented features, etc. Some of the tools are results of new research projects, starting with object-oriented approach from the first release.

All these tools are more or less similar at the basic modelling level where general entities are passing general stations. But big differences are met in application libraries: some tools have specialised for a certain application, like network modelling, offering preconfigured stations like network routers. Other tools offer different modelling levels, from a library with basic elements to high specialised modelling libraries.

The basic Elements of a DEVS are Servers and Queues. In this first approach of modelling the movement of people through a given area toward an exit a model consisting of servers and queues is build.

Classic DEVS as Enterprise Dynamics do offer only basic routines and functions to describe the movement of elements. For usually the way of such an element is given by its passing through other elements as servers and queues.

ED offers the *travelto* and the *movingto* function to let an element (atom) move from a starting position (x_0, y_0, z_0) to a destination (x_d, y_d, z_d) .

What is not taken into account when using this functions are two important things:

- * Due to the event based handling an agent cannot easily change its movement towards another destination
- * Several elements may take up the same position in space without causing a conflict in the system.

So when using a DEVS based simulation system especially the possibility of collisions has to be given special consideration.

The basic idea is to cover the area of interest by a "field" of servers (and queues), adding up to form the topology of the given area. The area is cut into single places and each of them represented by a server. Each place has eight adjoining places, building a structure similar to a honeycomb (Figure 5-1: Places represented by servers).

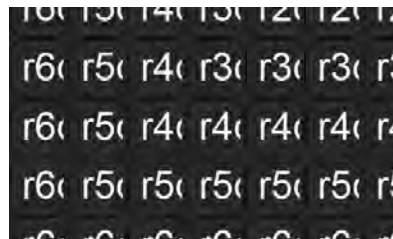


Figure 5-1: Places represented by servers

The cycle time of each server is given by the time a person needs to pass through and depends on their individual attributes as well as the position of the cell in regard to the exits. People walking through the area pass through these places, choosing the next place to go to according to certain rules.

In principle, this modelling approach for the spatial topology is similar to modelling approaches with cellular automata, whereby a server mirrors a cell. Here, the entities are active objects, changing the state of a server or cell, respectively, on the basis of a time event mechanism. A classical state update of cellular automata works on an equally spaced time pattern, here the update happens at the time of entering and exiting the cell.

Time in discrete event simulation systems does not pass continuously but in irregular time steps that correspond to the changes in the system state. So the movements of a person can be reduced to be a sequence of events. We can identify two different events: entering a place and exiting a place

Between entering and exiting will always pass a certain time, depending on the persons walking speed as each place does represent a certain walking distance, but exiting one place and entering another happens instantaneous.

These two events will be consecutively repeated until the person has reached its destination.

Parts of the area where people are not supposed to pass through due to obstacles or other reasons can be easily marked. It is only necessary to set the capacity of the corresponding places to zero to ensure they cannot be entered.

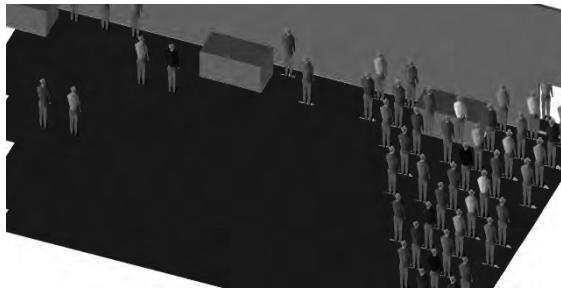


Figure 5-2: Pseudo-3D Visualisation of an area with obstacles

Modern simulation tools like Enterprise Dynamics allow generating this and other topologies automatically from databases.

Each person is modelled as a single element (entity) within the simulation, carrying their own attributes as size, speed, and the current spatial position.

They choose the exit they wish to use depending on the distance between their own position and the position of the exits. This decision is repeatedly done, so a change of direction might occur if the way to the chosen exit is blocked.

On this way, they are passing through the (fixed) places, each step depending on their decisions and the state of the surrounding places.

As the capacity of each passable place is set to 1 and each place represents exactly one position in space it is not possible for two elements to share the same position. If the next place a person would need to pass through to continue its way towards the exit is already occupied it will enter the next free place instead, rerouting itself. If none of the adjoining places is free the person is unable to move until one is vacated.

Using servers to represent each single position in the two dimensional space has one major drawback: to represent an area of even small size a huge amount of servers is needed. That slows down simulation time. In principle, the spatial distribution of a server equals the area a person needs to stand.

The first approach was based on using the classic features of Enterprise Dynamic: it has its own programming language developed to offer a high flexibility in using and expanding its functionality. This language allows creating and moving spatial objects, extending the capabilities of classical entities dramatically.

Atom Visualization Events Editor Attributes Table File Spatial Ch			
	X	Y	Z
Location	-8	-10	0
Size	5	2	1
Speed	0	0	0
Translation	0	0	0
	Around Self		Around Container
Current Rotation	0		0
Rotation Speed	0		0

Figure 5-3: Spatial Attributes of an Atom in Enterprise Dynamics

The basic modelling idea is now, to model the pedestrian by such spatial objects, which are moving and allocating places. The topology is similar to the one used in the first approach but instead of using servers to build up the walking area only the area itself is regarded now. It is once more cut into small places; once more each place has eight adjoining ones.

But now the information whether these places are free or occupied is stored in a table where each cell corresponds to one of these places.

Instead of setting capacities to zero to ensure a certain place may not be entered now it only needs to be marked as unavailable.

Movements of people are now reduced to only one reoccurring event: the next step. Time between these events once more depends on the walking speed. The continuous movement of a walking person is now broken down in jerky leaps from one position to the next.

The person no longer moves from one server into another. Instead it only changes its location, moving from its current position to the next. This is done with the

setloc(e1,e2,e3,e4) command that sets the x, y, z coordinates of the object (atom) referenced with $e4$ to $e1, e2, e3$ within the current spatial space.

The decision in what direction the next step is going to be taken is now based on the calculation of a direction vector that is calculated each time another step is taken.

Additionally it has to be checked whether the destination is already occupied by another person. In the first approach this was not necessary due to the usage of the servers – their basic functionality already covered this problem.

In the first approach, using servers to represent each single position in the two dimensional space had one major drawback: to represent an area of even small size a huge amount of servers is needed. That slows down simulation time. In this approach, simulation time could be reduced significantly.

The MoreSpace Model is designed to simulate the behaviour of about 20 000 students. Even though the second approach discussed here is a lot less demanding in regard of computational time the high number of agents to be simulated proved to be too much to remain within reasonable computing time of a simulation run. Therefore the simulation of the exact movements of each agent based on the layout of rooms and corridors was not implemented in ED but in an external simulation model developed in JAVA based on cellular automata. The ED model is able to use average travelling times or to interact with the JAVA model by sending the student agents to the external model of corridors and have them re-entering the system as soon as they have reached their destination. The agent based approach is the key for the successful interaction of both models as the agent simply crosses the borders to the other system and returns to the ED model at a later time. The development of the JAVA based model is not focus of this work, therefore only a short overview is given in the following remaining chapter.

5.3 Hybrid Model: Connection to the CA Model

The simulation of the exact movement of students through and between the TU buildings is technically possible in Enterprise Dynamics. A system of coordinates is ingrained into the ED structure; every element in ED has its spatial position in x, y, z coordinates as well as its speed, acceleration and translation. The atoms build to represent the agent student are handed over to the external model in JAVA for processing and calculation of the travelling time between buildings in different areas but also vacation times within the buildings itself. This system is furthermore backed up with a dynamic computation of vacation and room clearance times in ED, as for static values (i.e. areas where the vacation times do not differ influenced by dynamic parameters) values can be computed in the discrete simulator itself.

Implementing the hybrid system different problems occur. For example questions of solving interface problems between ED and the CA implemented in JAVA had to be solved. Hybrid Model

The main elements of the hybrid model are the "Students", that are moving through the existing structure of the CA i.e. the "Building structure". The building structure is processed on basis of CAD plans of the buildings.

The whole structure of the systems is very simple. This simplification was a main goal in planning and structuring the whole simulation set.

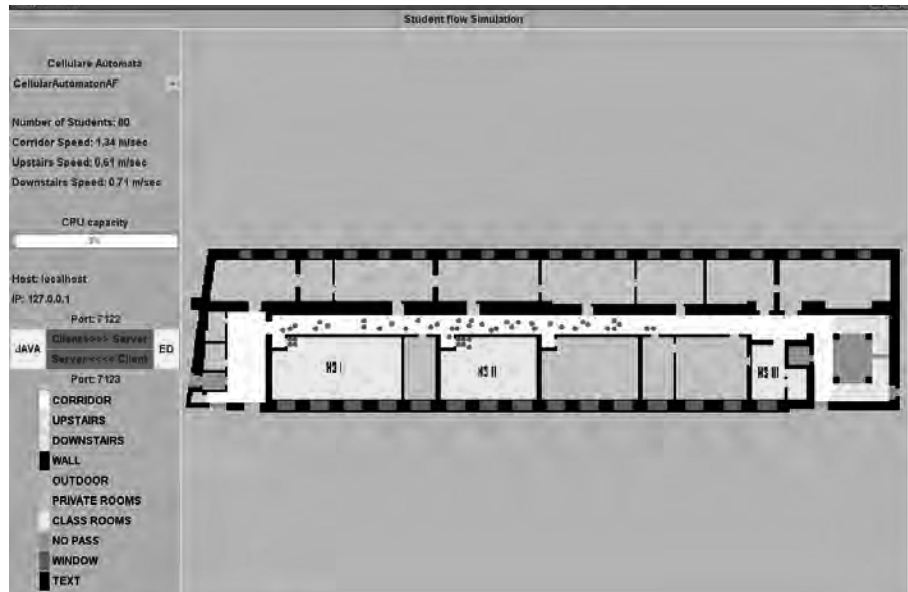


Figure 5-4 GUI in JAVA with CA of the currently simulated level

On one hand the data interface for getting information is complicated as plans change very often. For this reason the interface should be very easy to use not only for members of the research group but also for architects or other persons working with the plans.

On the other hand the simulation has to handle a huge amount of data. Vienna University of Technology has to coordinate over 9.000 rooms in about 26 objects spread over the whole inner city of Vienna. A total floor area of about 276.000 square meters would result in a total matrix size for the CA of about 17.700.000 cells. To handle those sizes two strategies were implemented. First of all the size of the matrices are not constant but variable to reduce the amount of cells. Second for this reason and for future parallelisation reasons the plans were cut into a lot of different, small sub-planes to process them in an easy way.

For speed values of individuals the speed results of the different grounds particles can walk over. While on stairs the upstairs velocity is approximately 0.61 m/sec an individual can move downstairs with about 0.71 m/sec. On corridors particles can cross with 1.34 m/sec. But the ground is not the only influence on speed of individuals. Another influence is the density of particles in an area. This is well known from simulation tools for escape routes in emergency situations and was also taken and integrated in the model.

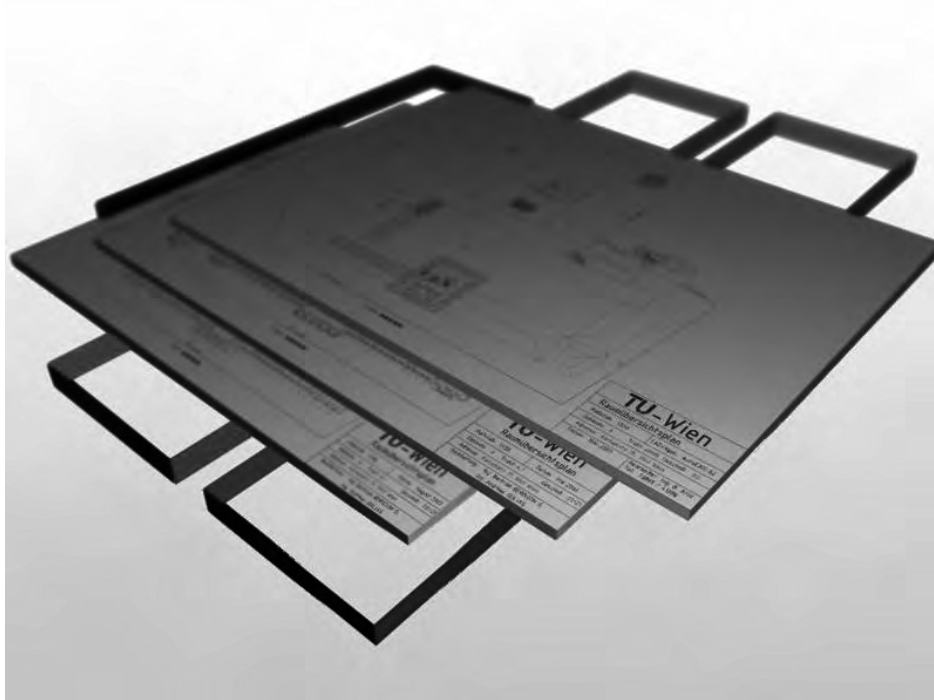


Figure 5-5: About 276.000 Square Meters have to be integrated into the CA

5.3.1 Interface between JAVA and ED

Because the JAVA program for Cellular Automata is of course only a part of the simulation system, it is necessary for data exchange to establish a connection between the two simulations. This interface was implemented with the aid of the Transmission Control Protocol and the Internet Protocol, or short TCP/IP. For a more detailed description see Section 6.5.

5.3.2 Problems

The approach of sending a TCP/IP message for each student from ED to the CA model proved to have two major disadvantages:

First the amount of messages for larger models with a few hundred students quickly became too much to handle in a reasonable time. For each student at least four messages are sent between the two models: the message from ED to the CA for the movement request, the according response and then the message to transmit the result of the CA calculation for the walking time of the student plus the according acknowledgment message from ED. This adds up to a very heavy traffic of messages between the two models and results into quite a time delay for processing them.

The second problem that arises is synchronizing the two models. To achieve a time synchronization of the two independent models that may even run on different computers it is necessary to use a fixed time interval for updating the system. During

times of student movement ED has to wait for the CA Model where the exact simulation of the movement takes some calculation time and causes it to run significantly slower than the ED model. The time interval used is 5 minutes assuming that walking distances of 5 minutes and less have no significant impact on the system. Lectures tend to start and end at times that are on a 5 minute grid. During times of no movement the CA model is empty and no synchronization has to be done.

Breaking down the simulation time of one semester that consists of at least 16 weeks into 5 minute intervals slows down the simulation run considerably. Assuming a lecturing time of five days a week, ten hours a day where at least half of the time student movements take place results in 4800 synchronization events. Considering the whole TU campus it even has to be assumed that at any given time there will be some kind of student movement. This would result in 9600 events.

The first problem can be solved by no longer using single messages for each student but one message for each time point where any new student movement is started. All students that want to change their position at the same time point are collected and the according data is transferred to the CA model using a simple text file. The message sent to the CA model will now contain the reference to the text file. This solution has a great impact especially for lectures with a high number of students as no longer four messages per student is needed - for a lecture with 150 students attending this would mean 600 messages - but four messages for the whole group. This proves to be a far more efficient way of communication between the two models.

The problem of time synchronization remains. The main idea of reducing the amount of events is to use a more flexible time interval between the synchronization points. This is based on the fact that in a discrete system nothing changes between one event and the next. So the update within the ED model is only necessary at the time of the next event in the event list. Due to the fact that the arrival of a student in the lecture room triggers no other event but the collecting of result data, this arrival can be delayed until the end of this lecture or the next occurring event. So it would be possible to set the next time of synchronization either to the time until the next event or 5 minutes, depending on the higher value.

This idea has not been fully realized yet so the actual decrease of synchronization events cannot yet be determined.

Overall it has to be said, that using this hybrid model for the simulation of the whole TU campus will require a high amount of resources and time. It is not planned to use this approach for every future simulation run but only for the special analysis of certain critical time periods within the semester.

The CA model is able to calculate realistic walking times to be used in the ED model for regular simulation runs. Only if the results of these runs show a problem the hybrid model can be used for further, much more detailed runs.

5.3.3 Impact on the MoreSpace Model

One thing the CA model was able to show was the dependency of the time needed to reach a certain lecture room from the 'student traffic'. It seems logical that the time will increase if a lot of people move through the same corridors, obstructing each other. The extent of the delay due to a high density of people is shown in Figure 7. This result could not have been calculated in the ED model.

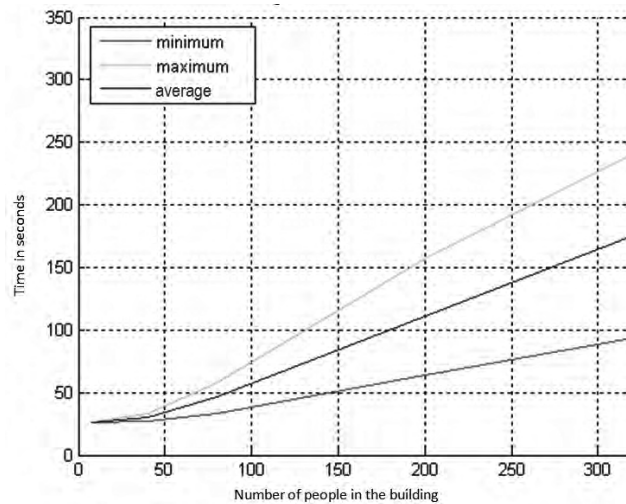


Figure 5-6: Walking time from room HS1 to room HS2

5.4 The MoreSpace Simulation Environment: **Enterprise Dynamics**

At present there exist a lot of software tools for modelling and simulation of discrete dynamic systems. Almost all tools are based on a modern object-oriented DEVS world view regarding active objects (entities) passing passive objects (stations) along given path. A time event mechanism updates movement of entities, being capable also of collisions and other state-dependent phenomena. Some of these tools can look back at a long tradition, now offering also any kind of object-oriented features, etc. Some of the tools are results of new research projects, starting with object-oriented approach from the first release.

All these tools are more or less similar at the basic modelling level where general entities are passing general stations. But big differences are met in application libraries: some tools have specialized for a certain application, like network modelling, offering preconfigured stations like network routers. Other tools offer different modelling levels, from a library with basic elements to highly specialized modelling libraries.

In this investigations the software tool Enterprise Dynamics is used, an object-oriented dynamic analysis and control system. The system consists of an Enterprise Dynamics (e.D.)-engine® and many building blocks grouped into e.D.-Suites®. An e.D.-Suite is configured for a specific field of expertise, to assist the modelling of a specific problem, branch or area.

Animations can range from 2D flowcharts to true 3D Virtual reality models that empower imagination and creativity. Building blocks can be easily created, customized and added.

For developing the later discussed simulation model Enterprise Dynamics 7.0 has been used. The model can be run in the currently newest version 8.0 without any compatibility problems.

Enterprise Dynamics can be regarded as a discrete event simulation program with a graphical user interface. In regard to the pyramid of simulators it can be attributed to two different levels: by using the very specialized E.D.-Suites Enterprise Dynamics becomes a simulation tool for a specific area of interest.

Using the 4d script Enterprise Dynamics can be used to develop new applications, thus putting it on the level of discrete simulation programming languages as it becomes a development environment as well as a simulator.

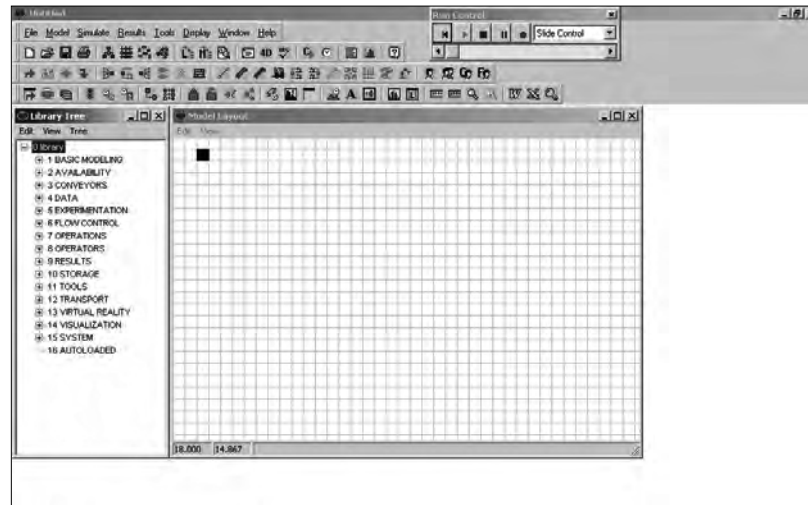


Figure 5-7 Simulation Environment of ED

5.4.1 Everything is an Atom

The Atom is the most important element in Enterprise Dynamics. The atom's main characteristics are:

An atom is something with 4 dimensions: its location and speed in 3D space (x,y,z) plus its dynamic behaviour (time).

- * Everything is an atom: an application, a model, a library, a resource, a product, etc.
- * Every atom can contain other atoms
- * Atoms can inherit properties from other atoms
- * Atoms can be created and destroyed
- * Atoms can be moved from one atom into another.
- * The atoms are hierarchically structured. At the top of the tree is the main atom. This is the only atom that is always present and it is automatically created when the program starts. The Main Atom is the only atom that is not contained by another atom.
- * Normally, the main atom will contain at least three atoms: the application atom, the library atom and the model atom.
- * The library atom generally contains a number of atoms with predefined functionality to be used when building a model.

All Atoms have exactly the same structure. With the atom editor you can access all existing functionality of an atom, add and delete functionality and even build completely new atoms from scratch.

The most important characteristics of an atom are:

- * Description and identification (Name, ID, Colour, Icon, Info, Mother)

- * Display settings (defining how the atom is animated)
- * Channels (each atom can have input and output channels)
- * Behaviour (each atom can have a description of its behaviour)
- * Attributes (each atom can have one or more attributes)
- * Table (each atom can have its own table with data)
- * Spatial (each atom has a location, size, speed, etc)
- * Label fields (also referred to as Dynamic Database or ddb)

5.4.2 Of Mothers and Daughters

When an atom is created it is always a daughter of another atom. This other atom will mostly be an existing atom from the library, but can also be a baseclass atom. A baseclass atom is an empty atom that has no functionality and only has default settings like: no attributes, no behaviour, no tables and its colour is black.

The atom that is created, copies all the characteristics and parameter values of the mother as they are at the time of creation. From that point on, these parameter values are independent of the mother. The parameters of the mother can be changed without affecting the parameters of the copy (whether it's a daughter or a duplicate) and the copy's parameters can be changed without affecting the mother. Only in the event handlers, the description of the atom's behaviour is not copied but dynamically inherited. This means that if you change the behaviour of the mother, the existing daughters will automatically also change their behaviour, except when you have overruled the mothers behaviour on the daughter explicitly.

Atoms can be displayed in tree structures or in animation windows. In these views you can generally double-click the atoms: most atoms will have an action associated with it.

The easiest way to create an atom is to drag an atom from the tree view into an animation window, but of course one can also use 4DScript commands.

5.4.3 The Atom Editor

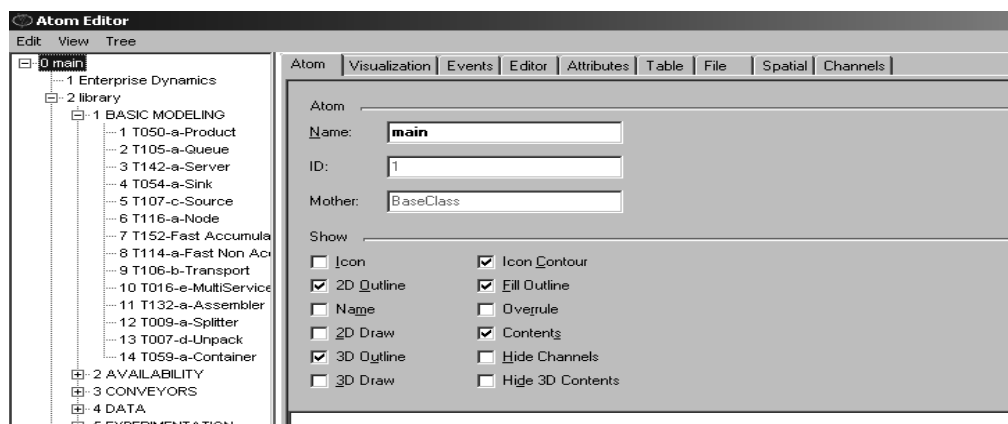


Figure 5-8: The Atom Editor

The atom editor allows defining the complete functionality and appearance of an atom. It can be activated from the tree windows by selecting it from the menu or by pressing the function key F9. The atom editor consists of a number of sheets:

Atom: general properties of the atom and its appearance

Visualization: colours, icons and VR appearance

Events: definition of the atom functionality in 15 event handlers

Editor: a text editor for editing 4DScript statements

Attributes: number and content of the atom's user-defined attributes

Table: an alpha-numerical table that is local to the atom

File: file attachments to atoms

Spatial: location and spatial parameters of the atom

Channels: style and name of channels

5.4.4 Events

The 15 event handlers on this sheet define the complete behaviour logic of an atom. The first column indicates the event type. The second column indicates if an atom inherits logic from its mother, otherwise it is blank. The third column displays the 4DScript logic that has been defined. Use the 4DScript edit field below the event handler fields, or the editor sheet, to actually edit the 4DScript code. The functionality of the 4DScript editor applies to both the edit field and the editor sheet. The inherited behaviour of an event handler can be displayed by double-clicking on the third column of the row.

The event handlers are:

- * Event
- * Entering
- * Entered
- * Exiting
- * Exited
- * Creation
- * Destruction
- * Reset
- * User
- * OcReady
- * IcReady
- * 2dDraw
- * 3dDraw
- * Message
- * Init

5.4.5 Eventlist

Enterprise Dynamics works event-oriented: The eventlist as shown in Figure 5-9 contains all future events sorted by their time of occurrence. Each event contains the atom reference for the atom that triggered it as well as an involved atom that may be affected by it. It is also possible to set a priority value to ensure the correct order of events that are scheduled at the same time.

ED Tracer						
1	Event:	t:	10.00	c:	1	p: 0 a: (149) Source1 i:0
2	Event:	t:	10.00	c:	1	p: 0 a: (150) Source2 i:0
3	Event:	t:	10.00	c:	1	p: 0 a: (158) Source12 i:0
4	Event:	t:	10.00	c:	1	p: 0 a: (162) Source16 i:0

Figure 5-9: ED Eventlist

5.4.6 Attributes

The number of attributes per atom can be defined in the attributes field. If more than 0 attributes have been defined a table will appear. The first column shows the attribute number.

Attribute values can be strings, numbers or 4DScript statements.

Attributes can be called by their name or number using the 4DScript command att.

The attributes of the current atom (c) can also be called directly by their name.

No.	Name	Attribute
1	cylinder	-negate()
2	radius	=1
3	batchsize	=1
4	batchsize	1
5	cum	0
6	curved	0
7	stepsize	0
8	entrytype	0
9	entrytype	0
10	radius	0
11	radius	0
12	totalbuy	0
13	cylinder	-(time()-epoch)/length()
14	dicycle	0
15	redbuy	0
16	redbuy	0
17	curvature	0
18	entrytype	-(epoch()-epoch)/epoch()
19	3dcon	3
20	buyflag	0
21	mch	0
22	redhead	0
23	mch2	0
24	count2	0
25	laststat1	0
26	laststat2	0
27	cylindercount	0
28	lastdown	0

Figure 5-10 Attributes

5.4.7 Tables

The table of an atom can be used to store data that is either used by the simulation (input data) or produced by the simulation (output data).

The table can be dimensioned using the Rows and Columns edit fields or with the 4DScript command `settable`. Table cells can contain values or numbers. As text that is typed as a valid 4DScript expression can be executed using the `execstring` command, table cells can in this way also contain 4DScript.

5.4.8 The 4d Script

4DScript is the programming language of Enterprise Dynamics. Everything that is executed in the Enterprise Dynamics software is or can be done via 4DScript. This openness allows the user to manipulate (parts of) the atoms as well as the complete user interface. 4DScript is a simple structured language that is interpreted by the engine when it is executed.

5.4.9 Enterprise Dynamics Simulation Environment

The Enterprise Dynamics Simulation Environment is subdivided into several parts; the ED.exe file is the lowest level, as shown in Figure 5-11 Basic ED Organisation. It contains the ED engine, the central core of the simulation environment. This file is the only one that must not be changed by the user. Using the ED.exe the *.app file can be opened. This file contains the information about which further files need to be loaded.

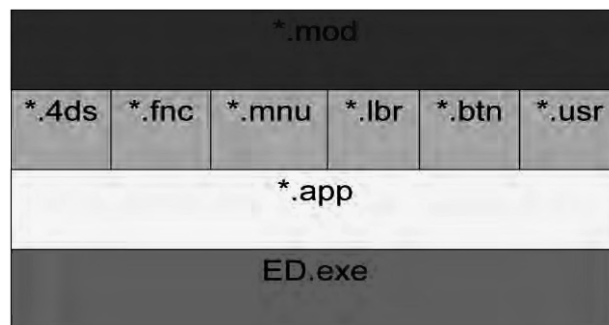


Figure 5-11 Basic ED Organisation

5.4.9.1 Application File

The ED Application file contains calls to *.usr, *.btn, *.mnu, *.fnc, *.4ds at the start-up of the ED Environment and therefore defines the layout and the included functionality of the ED application.

5.4.9.2 User File

The user file contains predefinition of certain functions and variables that are already needed at the start-up of the model file.

5.4.9.3 Buttons File

The Buttons file specifies the layout of the shortcut bar as well as the functionality behind it. For each button a function can be defined to execute a section of 4d script.

5.4.9.4 Library File

The library file contains the 4d script to load a list of atoms into the application library at the start-up of the model. Only atoms contained in the library can be used for the building of the simulation model. Of course it is possible to add additional atoms after start-up. The possibility to create atoms, save them and ass them to libraries makes the concept of reusable models executable. Atoms can be single atoms as well as more complex constructs, even complete submodels.

5.4.9.5 4ds File

The 4ds file contains 4d script commands that can be executed in the simulation environment.

5.4.9.6 Enc File

The 4ds file contains functions that can be executed in the simulation environment.

5.5 MoreSpace Application and Model file

The MoreSpace application file morespace.app loads the MoreSpace model file morespace.mod as well as the MoreSpace library file morespace.lib. The MoreSpace model file contains data tables to contain the simulation relevant data as well as a list of functions.

These functions can be grouped into several areas:

- * Functions for model build-up and initialisation
- * Functions for booking
- * Functions for data import from database
- * Functions for data export to database

A more detailed description is given in Chapter 6:.

5.6 MoreSpace GUI

The implementation of the simulation tool MoreSpace was designed in a way to ensure the maximum flexibility, not only in choosing the parameters for the booking procedure but in leaving the design of the simulation model itself as modifiable as possible. To achieve this strict diversion between the simulation model relevant data and the simulation environment is necessary. Input data does contain the list of all buildings on the TU Campus, all rooms and their individual parameters and the list of courses. All simulation input data is pre-processed in an external database, only at the time of the model assembly it is transferred to the simulation environment where it defines the design of the model build-up as well as the basic design of the scenario. Only the data needed during the simulation run is stored in ED tables to keep the amount of memory needed for the data storage to a minimum.

The assembly of the simulation model is part of the simulation run itself.

The simulation environment consists of four separate parts that interact with each other; at start-up three of these components are immediately available:

The Library that contains all atoms to build the simulation model, the Model Template that contains functions and tables and the Graphical User Interface (GUI) for choosing the parameters for the model behaviour.

Out of these three components two are needed to generate the MoreSpace Simulation Model.

The MoreSpace GUI has been developed to offer a clear overview over the parameters that can be set for a simulation run. All parameters entered are saved to the database table that is used for the simulation run to document the parameter set used and enable the user to repeat the run at a later time.

5.7 MoreSpace Library

5.7.1 Building

The building atom does not contain any functionality. Its main purpose is to group the room atoms to achieve a hierarchical structure.

5.7.2 Room

5.7.2.1 Room Atom Structure

The Room Atom is build from four atoms to model the clearance time. The room atom itself with the given capacity and the category that defines the kind of equipment the room offers. This is essential for the type of courses that can be held. To model the act of entering and leaving the room a server - queue combination is used consisting of two queues - one for the people leaving the room, another for those who want to enter - and a multi-server that represents the doorway. A multi-

server in Enterprise Dynamics is a server with capacity greater than one. A regular server in ED is also able to process several entities at once but only in a so called batch-run. This means all entities are processed simultaneously. The multi-server is able to handle several entities even with deferred entry times. The capacity of the multi-server representing the door corresponds to the size of the doorway. A small door where only one person at a time can pass through will have capacity 1; larger doors an accordingly higher capacity.

Entering and leaving has no fixed priority. Both queues are connected to the Server. A student entering the *Queue_In* will pass through the *Door* atom and enter the *Room*. A student leaving the room will enter the *Queue_Out*, pass through the *Door* atom and leave for their next destination.

Each room is represented as a single object within the simulation with its own attributes capacity, category and divisibility as well as its list of events with their date and time.

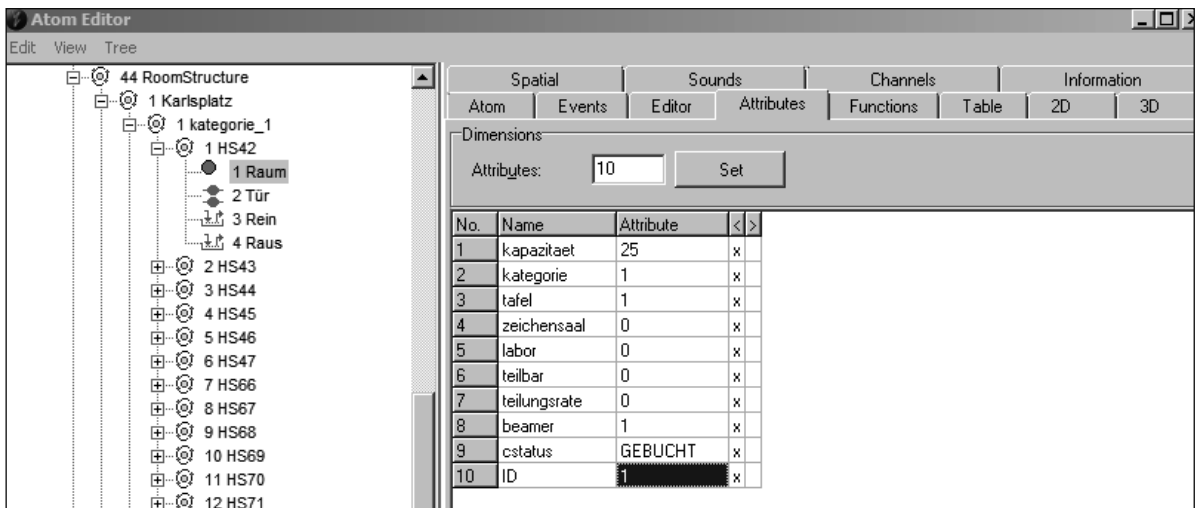


Figure 5-12 Lecture Hall with Attributes

5.7.2.2 Capacity

The capacity of a room defines the maximum number of people it is able to accommodate under optimal conditions. This means that if more people use this room than the capacity states, the quality of this event will suffer. For example during a lecture the surplus people will not have a seat. More dramatic is the consequence for labs: the capacity gives the number of workplaces; if more people attend they cannot participate in the laboratory course.

5.7.2.3 Category

The category of a room defines its equipment and therefore the type of lectures that can be held. For example there are very different requirements for classrooms for drawing for architects and laboratory rooms for chemists.

- * Lecture halls: for more than 60 people
- * Seminar rooms: < 60
- * Laboratory rooms
- * Computer rooms
- * Drawing rooms

5.7.2.4 Divisibility

The concept of dividable rooms is generally not a new idea. At the Technical University it has not been used so far due to the old design of most rooms. During the course of renovation the possibility is given to remodel some rooms in a way that they may be united to one large room or separated with dividing walls to be used as several smaller rooms

5.7.2.5 List of Events

Each room has a list of events that will take place in it. This list contains the exact time and date of the event, as well as the information about the event.

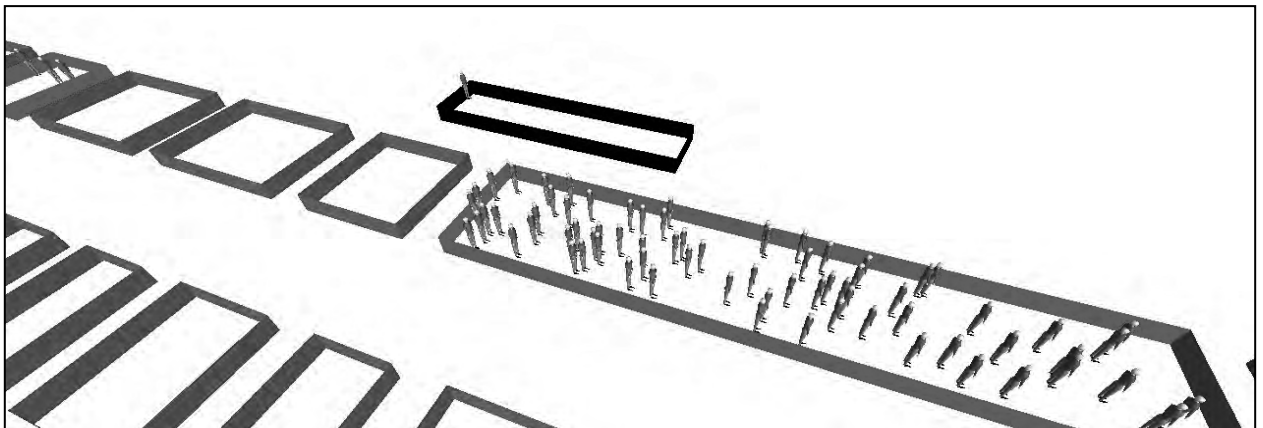


Figure 5-13 Simulation Model 3D in ED

5.7.3 Courses

The list of all courses is interpreted and processed. This list is the main basic data input that for simulation of the booking procedure as well as the semester run.

The course atom contains at least one sub atom; in case the course is split into several parallel groups it will contain one sub atom per group. The sub atoms main use is to

hold the list of lectures for each group, complete with time and place they are held as well as the number of students that are expected to attend. This information is crucial for the generation of students.

5.7.4 Student

The student atom contains the behaviour of the student as well as their list of lectures they have to attend. This list is generated at the time of creation of the student and results from their semester as well as field of study. The generation is based on probability functions to achieve a wide variety of lecture combinations to represent the inhomogeneous groups of students that occur especially in courses in higher semesters.

The behaviour of the students is based on their goals – attending their lectures – as well as on the overall situation. In case of collision of events they wish to attend students have to make a decision.

5.7.4.1 Event Matrix

Each student atom owns a matrix of lectures it will attend. The size of the matrix $n \times 4$ is given by the number of lectures n the student is assigned to.

This matrix L contains the basic information for the behaviour of the student s during the simulation.

$$L_s = \begin{pmatrix} l_{1,1} & \dots & l_{1,4} \\ \vdots & \vdots & \vdots \\ l_{i,1} & \dots & l_{i,4} \end{pmatrix}$$

The element $l_{i,1}$ contains the begin time of the lecture i .

The element $l_{i,2}$ contains the end time of the lecture i .

The element $l_{i,3}$ contains the ID of the course attended at lecture i .

The element $l_{i,4}$ contains the pointer to the room of the lecture i .

5.7.4.2 Attending a Lecture

Here the event for the attending of a lecture is set. The time the student will need to reach the location of the lecture hall has to be taken into consideration.

T_c	current time
T_i	start time of the lecture
\hat{T}_i	end time of the lecture
$\tilde{T}_i = T_e + t_w$	start time of the lecture considering walking time

T_e	time for the student to start walking
t_w	Time it takes the student to reach the lecture hall from their current position
t_e	time until T_n

If $i \leq n$:

$$\begin{aligned} \exists T_i &= L_S(i, 1) = l_{i,1} \\ \exists C_i &= L_S(i, 3) = l_{i,3} \\ \exists R_i &= L_S(i, 4) = l_{i,4} \end{aligned}$$

With

$$\begin{aligned} l_{i,1} &\geq T_c \\ C_i &\neq C_c \end{aligned}$$

Hence:

$$t_e = \max(T_e - T_c, 0)$$

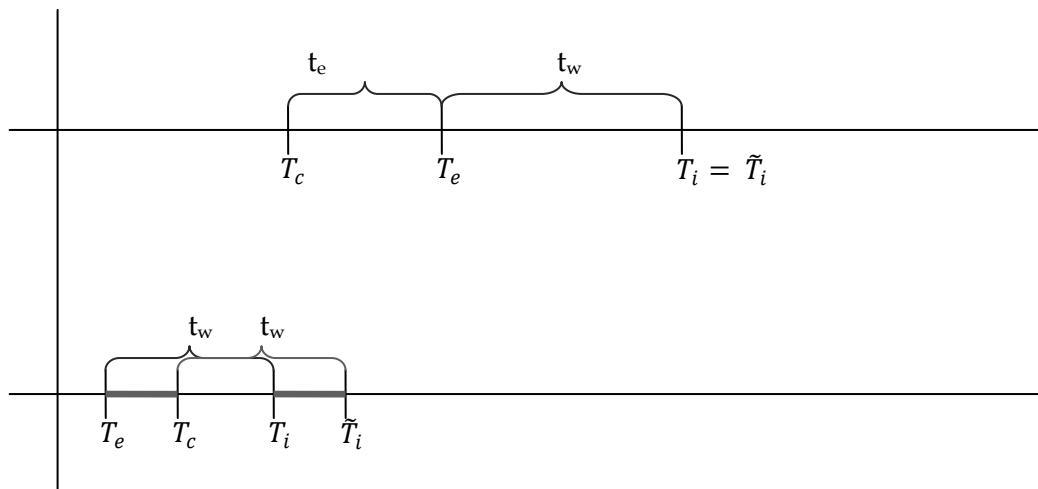


Figure 5-14: Calculation of the time of event

If $i > n$ the students have finished their list of lectures and can leave the simulation.

5.7.4.3 Queuing

During this activity the student moves into the queue in front of the lecture hall.

The according event for leaving a room is set by evaluating the next lecture the student will visit.

$$\begin{aligned}T_i &= L_s(i, 1) = l_{i,1} \\ \hat{T}_i &= L_s(i, 2) = l_{i,2}\end{aligned}$$

Under the following condition:

$$\begin{aligned}T_i &\geq T_c \\ C_i &\neq C_c\end{aligned}$$

$$T_{i+1} < \hat{T}_i \Rightarrow \text{Conflict !!!}$$

According to probability $\mathbb{P} = \frac{1}{2}$ the student will chose one of the two possibilities for the time T_e when he will leave the current lecture:

$$T_e = \begin{cases} \hat{T}_i \\ \max(T_i - t_w, 0) \end{cases}$$

And:

$$t_e = \max(T_e - T_c, 0)$$

Chapter 6: Functional Description of MoreSpace

MoreSpace is developed in Enterprise Dynamics, a Discrete Event Simulation Environment. This indicates that all changes in the system are regulated via events that take place at certain points of time. For modelling the individual students the concept of Agent Based Modelling is used, a concept that has to be integrated in the DEVS environment of ED.

Each student is represented as an atom, sharing the common structure of all elements in ED.

The behaviour of the students is not a complex one: they have a couple of courses they want to attend and have to move from one lecture to the next. Only in the case of consecutive lectures the aspect of the walking time has to be taken into consideration as well. If the time between two lectures considerably exceeds the average walking time to cross the distance between the two locations they take place in the walking time itself is not so much of interest as the effect the number of people in the building may have.

A student may be in one of the following states:

- * Waiting for the next lecture
- * Moving to the location of the next lecture
- * Entering a room
- * Attending a lecture
- * Leaving a room

6.1 Booking Procedure

The Booking Procedure encompasses all functions concerning the booking of rooms for lectures.

To carry out all options that may be activated for the booking procedure it is parted into three loops:

6.1.1 Best Fit Loop

In this loop a room is only booked if all four booking criteria hold:

- * Strong Best Fit Criterion: $C_R = N_C$
- * Weak Best Fit Criterion: $C_R \geq N_C$
- * Optimal Environment Criterion: $E_R = E_C$
- * Room Setup Criterion: $S_R = S_C$

Basically this means that only a room that has the required setup and environment as well as the exact capacity needed. A room with higher capacity than required for this lecture will not be booked in this loop.

6.1.2 Optimal Environment Loop

In this loop a room is booked if three of the four booking criteria hold:

- * Weak Best Fit Criterion: $C_R \geq N_C$
- * Optimal Environment Criterion: $E_R = E_C$
- * Room Setup Criterion: $S_R = S_C$

Any room that has at least the capacity that is required available can be booked now.

6.1.3 Room Setup Loop

In this loop a room can be booked if only two of the four booking criteria hold:

- * Weak Best Fit Criterion: $C_R \geq N_C$
- * Room Setup Criterion: $S_R = S_C$

Any room that has the setup and at least the capacity that is required available may be booked now.

If certain options are not activated the according loop is not executed. If e.g. the best fit option is set to OFF the booking procedure will start with Loop 2, looking only for a room with an optimal environment but not for the best fit.

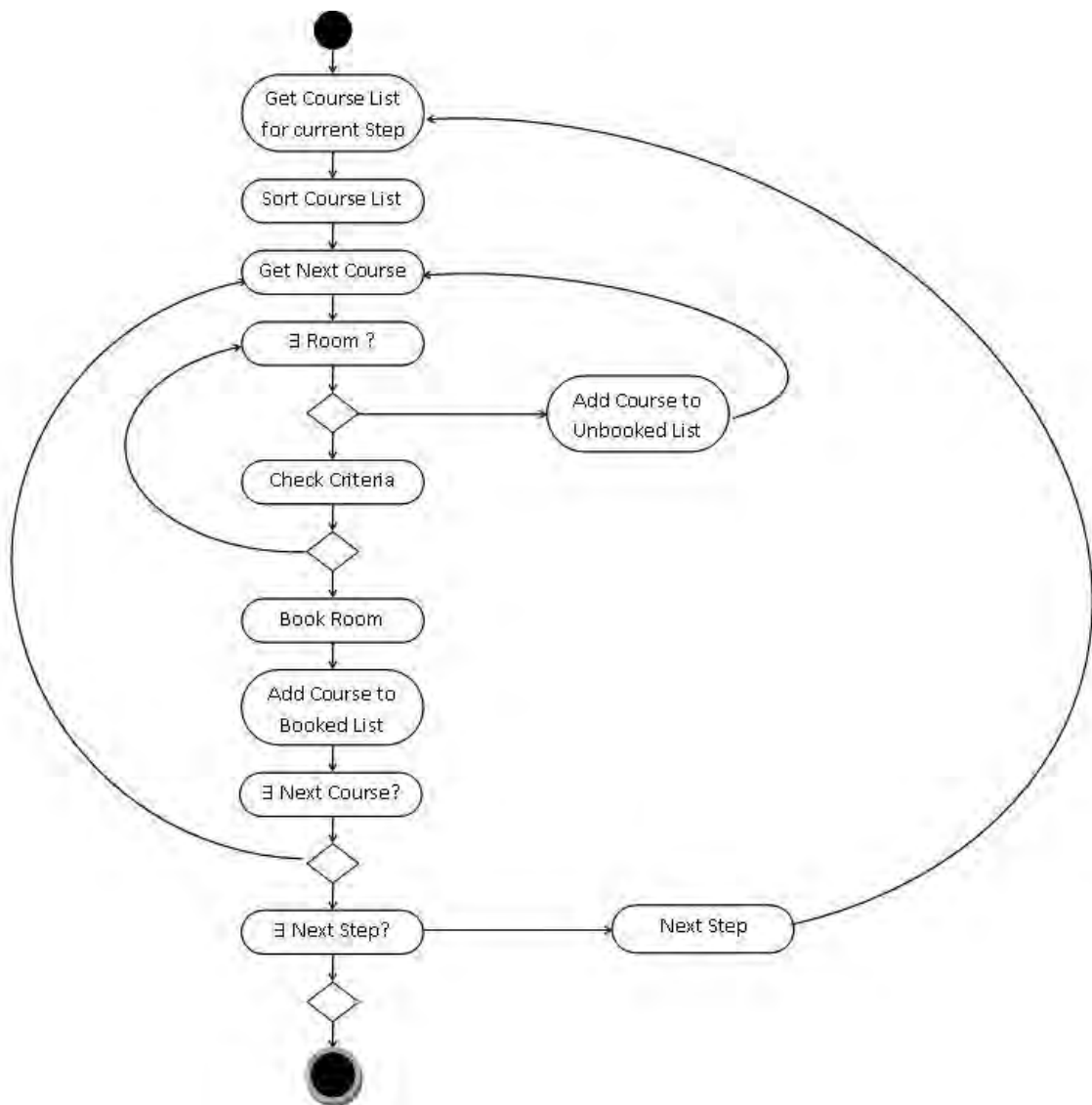


Figure 6-1: UML Diagram: Booking Procedure

6.2 Student Creation Procedure

This function generates the students at the beginning of a dynamic simulation run. The number of students is a result of the number of courses and the number of students that are expected to attend each of them.

Each student is created as an individual copy of the student atom in the MoreSpace library. It has a set of rules ingrained that defines the behaviour of the student in their reaction to certain situations.

As soon as it is created the student atom its attributes $study_s$ and sem_s are accordingly set to $study_s$ and sem_s . The function loops over all studies and semesters, so these values are increasing during the generation procedure.

The number of mandatory courses m_s for a student s is derived as

$$m_s = random(1, \min(m_{max}, m(study_s, sem_s)))$$

where

- * m_{max} is the maximum number of compulsory courses a student may attend. This value may be changed by the user.
- * $m(study_s, sem_s)$ is the number of compulsory courses for this study and this semester still available. It depends on the number of students already assigned to the courses.

The number of non mandatory courses n_s for a student s is derived as

$$n_s = random(1, \min(n_{max}, n(study_s)))$$

where

- * n_{max} is the maximum number of elective courses a student can attend. This value may be changed by the user.
- * $n(study_s)$ is the number of elective courses still available for $study_s$. It depends on the number of students already assigned to the courses.

The number of free courses for a student f_s is derived as

$$f_s = random(1, \min(f_{max}, f))$$

where

- * f_{max} is the maximum number of optional courses a student can attend. This value may be changed by the user.
- * f is the number of compulsory courses still available. It depends on the number of students already assigned to the courses.

N_s is the number of all different studies that are offered at the TU Vienna.

N_m is the number of all compulsory courses for all studies and all semesters:

$$N_m = \sum_{i=1}^{N_s} \sum_{j=1}^{10} N_m(i, j)$$

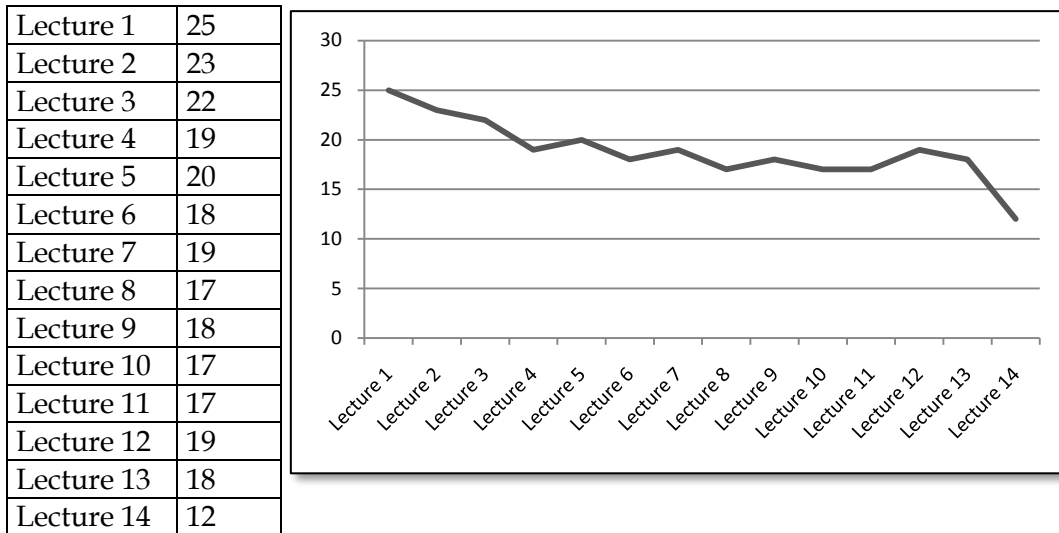


Figure 6-3 Student Numbers

The maximum number of students is 25, so 25 students will be assigned to attend this course. Due to the varying number of attending students not all of them will be assigned to all lectures.

6.2.1.1 Assignment of Compulsory Courses

Compulsory courses are specific for the study and the semester of the student s . The number of these mandatory courses for a certain study and semester is $m(study_s, sem_s)$. Let M be the set of mandatory courses with power $\#M = m(study_s, sem_s)$.

$$M = \{C_1, \dots, C_n\}$$

Each course C_i is split into g groups G where $g \geq 1$; each group contains l lectures L .

$$C_i = \{G_{i1}, \dots, G_{ig}\}$$

$$G_{ij} = \{L_{ij1}, \dots, L_{ijl}\}$$

where

$$L_{ijk} = (l_{ijk1}, \dots, l_{ijk5})$$

With

- * l_{ijk1} ID number of the course
- * l_{ijk2} Begin time of the lecture
- * l_{ijk3} Length of the lecture
- * l_{ijk4} Room
- * l_{ijk5} Expected number of students

Let M_s be the set of groups the current student s is assigned to. The number of groups equals $\#M_s = r$; at the end of the procedure it has to be: $r = m_s$.

$$M_s = \{G_{s1}, \dots, G_{sr}\}$$

with

$$G_{si} \neq G_{sj} \quad \forall i, j \in \{1, \dots, r\}$$

Let M_u be the set of courses the current student s is not assigned to with

$$\#M_u = m(study_s, sem_s) - r$$

$$M_u = \left\{ C_i \mid \#C_i > 0 \wedge G_{ij} \notin M_s \quad \forall G_{ij} \in C_i, \quad \forall i \in \{1, \dots, \#M_u\} \right\}$$

For assigning a course random variables v, w are derived as

$$\begin{aligned} v &= \text{uniform}(1, \#M_u) \\ w &= \text{uniform}(1, \#C_v) \end{aligned}$$

v is uniformly distributed between 1 and the number of available courses.

w is uniformly distributed between 1 and the number of groups the course C_v consists of.

The student is assigned to the group $G_{vw} \in C_v \in M_u$. This means:

$$M_s = M_s \cup G_{vw}$$

With

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\}$$

The group G_{vw} has l lectures; for each lecture k l_{ijk5} contains the number of expected students. For each student assigned this number is decreased by one. If $l_{vwk5} = 0$ the number of expected students has been reached and the according lecture will no longer be assigned to students.

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\} \setminus \{L_{vwk}\}$$

If the group does not contain any lectures any more, it will be removed from the set.

$$\text{if } \#G_{vw} = 0: C_v = C_v \setminus \{G_{vw}\} = \{G_{v1}, \dots, G_{vg}\} \setminus \{G_{vw}\}$$

If the course C_v does not contain any groups any more it is removed from the set of available mandatory courses:

$$\text{if } \#C_v = 0: M = M \setminus \{C_i\} = \{C_1, \dots, C_n\} \setminus \{C_v\}$$

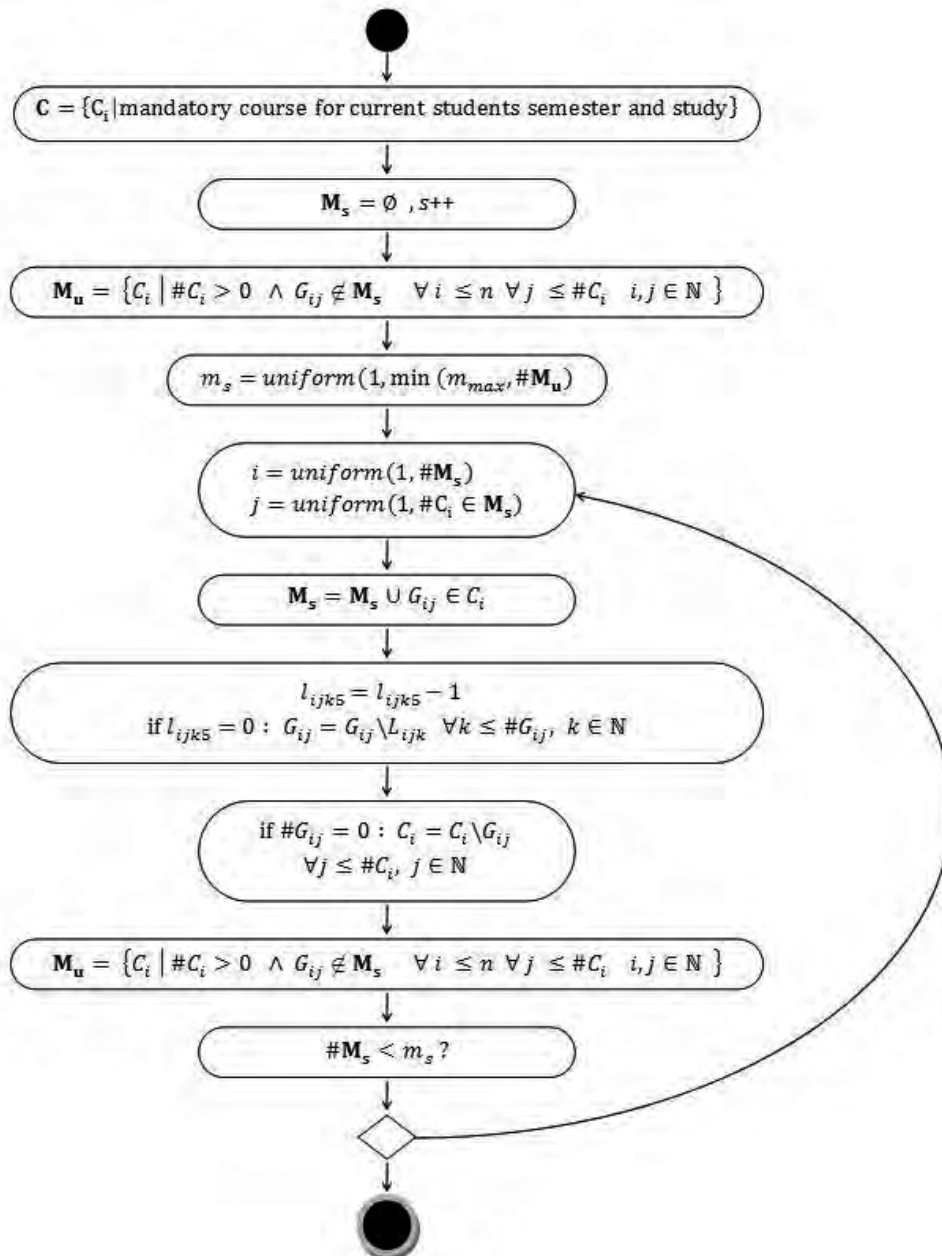


Figure 6-4: Assignment of Compulsory Courses

6.2.1.2 Assignment of Elective Courses

Elective courses are courses that are specific for the study of the student s but not for the semester. They are usually attended during the last segment of the studies and have to amount to a certain number of semester hours. They can be chosen from a pool of courses where compulsory courses that are mandatory for some studies may be elective for others.

The number of elective courses for a certain study and semester is $n(study_s)$. Let N be the set of available non mandatory courses with power $\#N = n = n(study_s)$

$$N = \{C_1, \dots, C_n\}$$

Each course C_i is split into g groups G where $g \geq 1$; each group contains l lectures L .

$$C_i = \{G_{i1}, \dots, G_{ig}\}$$

$$G_{ij} = \{L_{ij1}, \dots, L_{ijl}\}$$

where

$$L_{ijk} = (l_{ijk1}, \dots, l_{ijk5})$$

With

- * l_{ijk1} ID number of the course
- * l_{ijk2} Begin time of the lecture
- * l_{ijk3} Length of the lecture
- * l_{ijk4} Room
- * l_{ijk5} Expected number of students

Let N_s be the set of groups the current student s is assigned to. The number of groups equals $\#N_s = r$; at the end of the procedure it has to be: $r = m_s$.

$$N_s = \{G_{s1}, \dots, G_{sr}\}$$

with

$$G_{si} \neq G_{sj} \quad \forall i, j \in \{1, \dots, r\}$$

Let N_u be the set of courses the current student s is not assigned to with

$$\#N_u = n(study_s) - r.$$

$$N_u = \left\{ C_i \mid \#C_i > 0 \wedge G_{ij} \notin N_s \quad \forall G_{ij} \in C_i, \quad \forall i \in \{1, \dots, \#N_u\} \right\}$$

For assigning a course random variables v, w are derived as

$$v = \text{uniform}(1, \#N_u)$$

$$w = \text{uniform}(1, \#C_v)$$

- * v is uniformly distributed between 1 and the number of available courses.
- * w is uniformly distributed between 1 and the number of groups the course C_v consists of.

The student is assigned to the group $G_{vw} \in C_v \in N_u$. This means:

$$N_s = N_s \cup G_{vw}$$

With

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\}$$

The group G_{vw} has l lectures; for each lecture k l_{ijk} contains the number of expected students. For each student assigned this number is decreased by one. If $l_{vwk} = 0$ the number of expected students has been reached and the according lecture will no longer be assigned to students.

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\} \setminus \{L_{vwk}\}$$

If the group does not contain any lectures any more, it will be removed from the set.

$$\text{if } \#G_{vw} = 0: C_v = C_v \setminus \{G_{vw}\} = \{G_{v1}, \dots, G_{vg}\} \setminus \{G_{vw}\}$$

If the course C_v does not contain any groups any more it is removed from the set of available elective courses:

$$\text{if } \#C_v = 0: N = N \setminus \{C_i\} = \{C_1, \dots, C_n\} \setminus \{C_v\}$$

6.2.1.3 Assignment of Additional Courses

Additional courses can be chosen freely with no regard to the studies. They are usually attended during the last segment of the studies and have to amount to a certain number of semester hours. They can be chosen from the pool of all courses available at the TU Vienna. As there is no rule how to choose these courses in the simulation model only those courses are regarded, that do not already appear in the group of elective or compulsory courses.

The number of freely selectable courses is f . Let F be the set of freely selectable courses with power $\#F = f$

$$F = \{C_1, \dots, C_n\}$$

Each course C_i is split into g groups G where $g \geq 1$; each group contains l lectures L .

$$C_i = \{G_{i1}, \dots, G_{ig}\}$$

$$G_{ij} = \{L_{ij1}, \dots, L_{ijl}\}$$

where

$$L_{ijk} = (l_{ijk1}, \dots, l_{ijk5})$$

With

- * l_{ijk1} ID number of the course
- * l_{ijk2} Begin time of the lecture
- * l_{ijk3} Length of the lecture
- * l_{ijk4} Room
- * l_{ijk5} Expected number of students

Let C_s be the set of groups the current student s is assigned to. The number of groups equals $\#C_s = r$; at the end of the procedure it has to be: $r = m_s$.

$$C_s = \{G_{s1}, \dots, G_{sr}\}$$

with

$$G_{si} \neq G_{sj} \quad \forall i, j \in \{1, \dots, r\}$$

Let F_u be the set of courses the current student s is not assigned to. $\#F_u = f - r$.

$$F_u = \left\{ C_i \mid \#C_i > 0 \wedge G_{ij} \notin F_s \quad \forall G_{ij} \in C_i, \quad \forall i \in \{1, \dots, \#F_u\} \right\}$$

For assigning a course random variables v, w are derived as

$$\begin{aligned} v &= \text{uniform}(1, \#F_u) \\ w &= \text{uniform}(1, \#C_v) \end{aligned}$$

- * v is uniformly distributed between 1 and the number of available courses.
- * w is uniformly distributed between 1 and the number of groups the course C_v consists of.

The student is assigned to the group $G_{vw} \in C_v \in F_u$. This means:

$$F_s = F_s \cup G_{vw}$$

With

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\}$$

The group G_{vw} has l lectures; for each lecture k l_{vwk5} contains the number of expected students. For each student assigned this number is decreased by one. If $l_{vwk5} = 0$ the

number of expected students has been reached and the according lecture will no longer be assigned to students.

$$G_{vw} = \{L_{vw1}, \dots, L_{vwl}\} \setminus \{L_{vwk}\}$$

If the group does not contain any lectures any more, it will be removed from the set.

$$\text{if } \#G_{vw} = 0: C_v = C_v \setminus \{G_{vw}\} = \{G_{v1}, \dots, G_{vg}\} \setminus \{G_{vw}\}$$

If the course C_v does not contain any groups any more it is removed from the set of available not mandatory courses:

$$\text{if } \#C_v = 0: F = F \setminus \{C_i\} = \{C_1, \dots, C_n\} \setminus \{C_v\}$$

At the end of the procedure each student owns a set of groups that contains the list of all lectures to attend:

$$S_s = M_s \cup N_s \cup F_s$$

6.3 Simulation Run

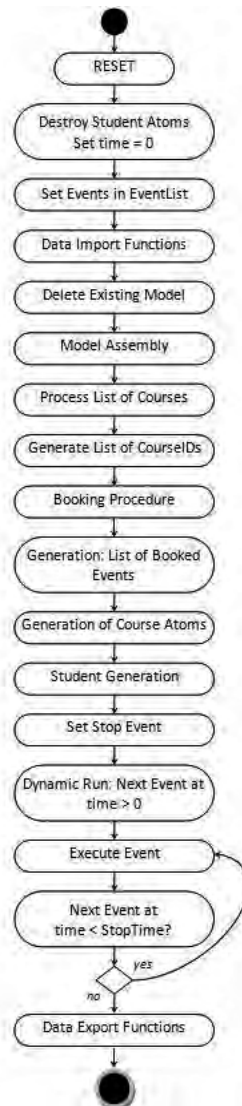


Figure 6-5: UML Diagram: Operational Sequence of MoreSpace

The UML diagram shown in Figure 6-5: UML Diagram: Operational Sequence of MoreSpace demonstrates the sequence of functions executed to start the MoreSpace Simulation Tool and run a scenario. A more detailed description of the different steps is given in the following sections.

6.4 Functions regarding Data Exchange

MoreSpace_ADOConnect: This function established and tests the connection to the database. The database can be selected via the GUI.

6.4.1 Interface between ED and the Simulation Database

Enterprise Dynamics has an ADO (ActiveX Data Objects) Interface for communication with external programs integrated in its environment. Using this interface ED is able to connect with the MoreSpace simulation database that has been developed in Microsoft Access.

The ADO data model consists of three major components:

- * Connection – it must contain the place and name of the database and establishes the connection
- * Recordset – the construct that contains data in form of tables and/or queries
- * Command – is used to transmit SQL commands to the database

6.4.2 Interface to TUWIS++ / TISS

The interaction between the TUWIS++ database and the simulation database is kept as simple as possible. The future development will include the switch from TUWIS++ to TISS but it will most probably not be necessary to change the current way of interaction. Until now TUWIS++ data was exported in CVS files that can be imported into a database table without much effort. The major future change will be to add a function to the database to import the file from a given location on command that may even be triggered from the ED environment.

Mandatory for a useful simulation is the high quality of input data, something that has up to now only been given with lots of additional work to ensure the data about courses has been complete. The newest developments for TISS should ensure a good quality without additional adding or editing of data.

The premise for complete data is a clear definition of the data requirement for the simulation model.

6.5 Interface from ED to the CA Model

Because the JAVA program for Cellular Automata is of course only a part of the simulation system, it is necessary for data exchange to establish a connection between the two simulation models.

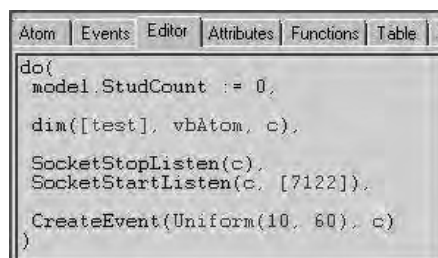
This interface was implemented with the aid of the Transmission Control Protocol and the Internet Protocol, or short TCP / IP.

Basically, the communication is divided into two different parts sending and the receiving data. Differences between these two states are the allocation of server and client functions. In the implementation of this interface it is only a unidirectional traffic towards server possible, however, for this simulation bidirectional data traffic is required, so both JAVA and ED act as server and client at the same time.

The red area in the picture shows the most important information for communicating. From starting above the first three lines contains information about the JAVA client. In the case that both programs are running on the same computer JAVA can send data to the ED server using the "local host", the loopback address "127.0.0.1" and the port 7122. Conversely JAVA acts as a server and receives data on port 7123. The underlying two yellow boxes with the content "JAVA" and "ED" represent the two programs, which are connected via a network. The status of the connection is symbolized by the two bars in the middle: red signals a faulty connection; green indicates the status is ok. The status check is done using a message ("PING") by sending from the JAVA environment to ED that must be countered with the message ("ALIVE") within a given response time interval. ED offers several interfaces to communicate with external programs, for example DDE and ADO for communication with databases.

- * 4d script commands for execution of DDL functions
- * Exchange of socket messages to allow communication via the TCP/IP ports
- * Sending and receiving emails to communicate worldwide

For this particular simulation the communication using socket messages is used. It allows exchanging ASCII messages via a TCP/IP port. These messages are sent and received by a specified atom in ED; incoming messages trigger, the so called ONMESSAGE-Event Handler of this atom that contains the code sequence to be executed.



```
Atom | Events | Editor | Attributes | Functions | Table | 2
do(
  model.StudCount := 0,
  dim([test], vbAtom, c),
  SocketStopListen(c),
  SocketStartListen(c, [7122]),
  CreateEvent(Uniform(10, 60), c)
)
```

Figure 6-6: Screen Shot of Code sending a socket message

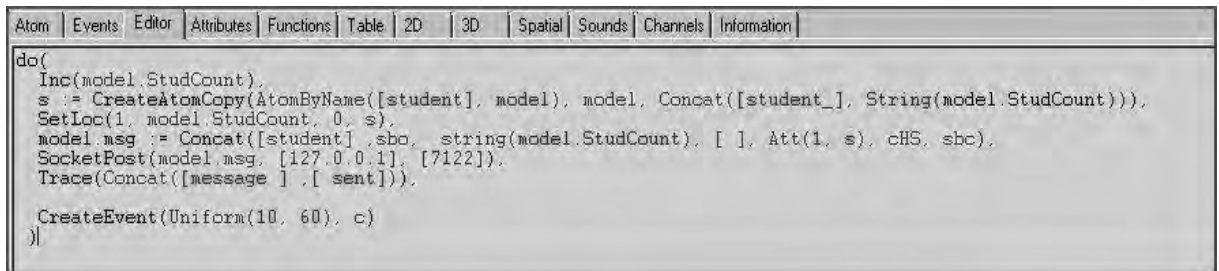
Enterprise Dynamics sends a message containing the students ID, the current position and the room the student has to move to, to the JAVA model.

[student 123456 HS1 HS2]

The corresponding student is ‚frozen‘ in the ED model. A confirmation message ensures that the message was received; The JAVA model creates the student with ID 123456 and places it in HS1. The destination is given as HS2, so the student starts moving through the cellular automata towards HS2. After reaching the destination the JAVA model sends a return message that contains the ID of the student as well as the description of their movement: the current position as well as the time needed to cover the distance between former and current position is sent to ED:

[Student 123456 HS1 HS2 67, 8]

The frozen student is reactivated and updated.



```
Atom | Events | Editor | Attributes | Functions | Table | 2D | 3D | Spatial | Sounds | Channels | Information
do(
  Inc(model.StudCount).
  s := CreateAtomCopy(AtomByName([student], model), model, Concat([student_], String(model.StudCount))),
  SetLoc(1, model.StudCount, 0, s),
  model.msg := Concat([student], sbc, string(model.StudCount), [ ], Att(1, s), chS, sbc),
  SocketPost(model.msg, [127.0.0.1], [7122]),
  Trace(Concat([message ] , [ sent])).

  CreateEvent(Uniform(10, 60), c)
)
```

Figure 6-7: Screen shot of reacting on an incoming socket message in ED

So the interface is implemented with a few lines of code. As the system has to be called for every change for every student at the moment a kind of pulsing was integrated to collect requests.

6.6 Functions regarding Alternative Scenarios

6.6.1 Reduction of Not Booked Lectures

The system checks the result data and is able to identify a shortcoming of a certain room category during a distinct period of time. A new scenario is created that tries to outbalance the difference between required room and available room by using the divisibility of rooms to rearrange the room structure. I.e. a large room for capacity > 300 is needed but not available the system will try to join several smaller rooms during that time to fulfil the requirement.

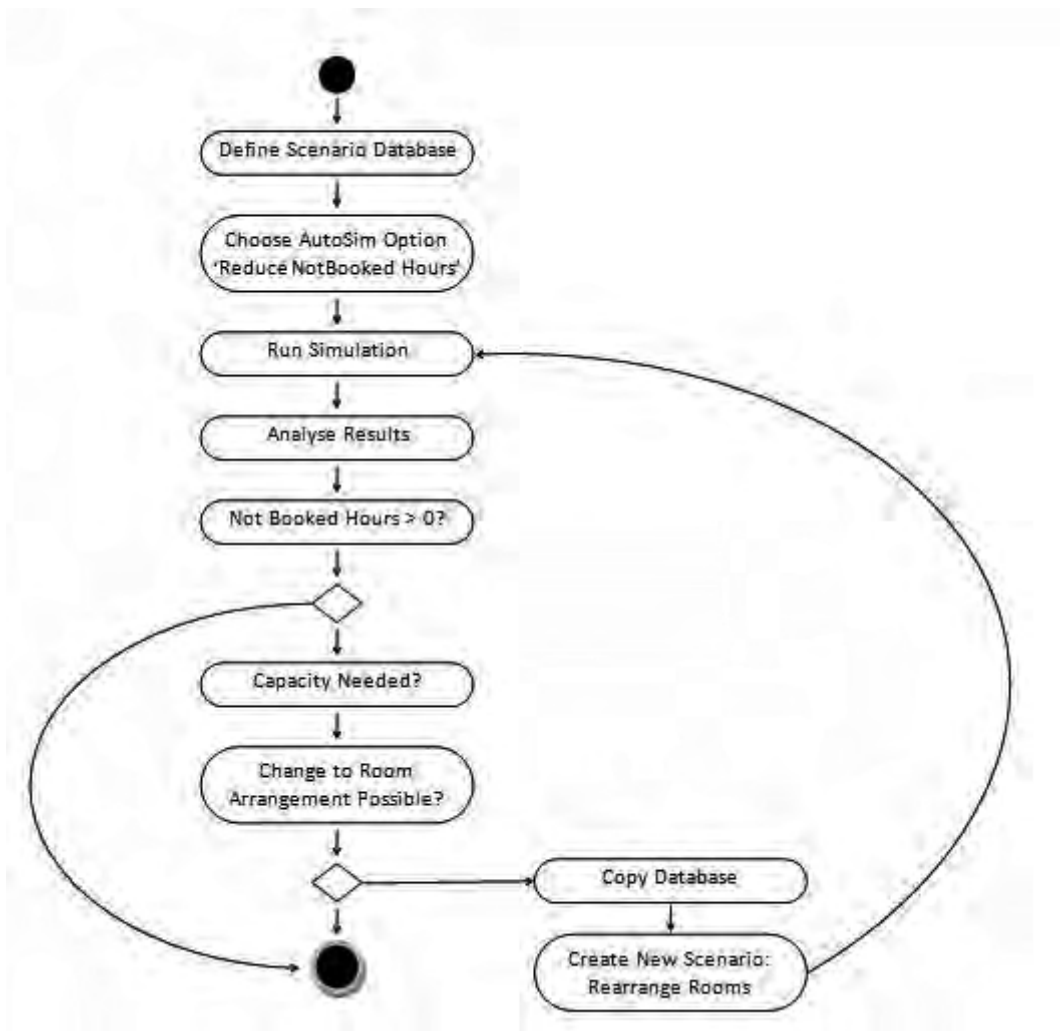
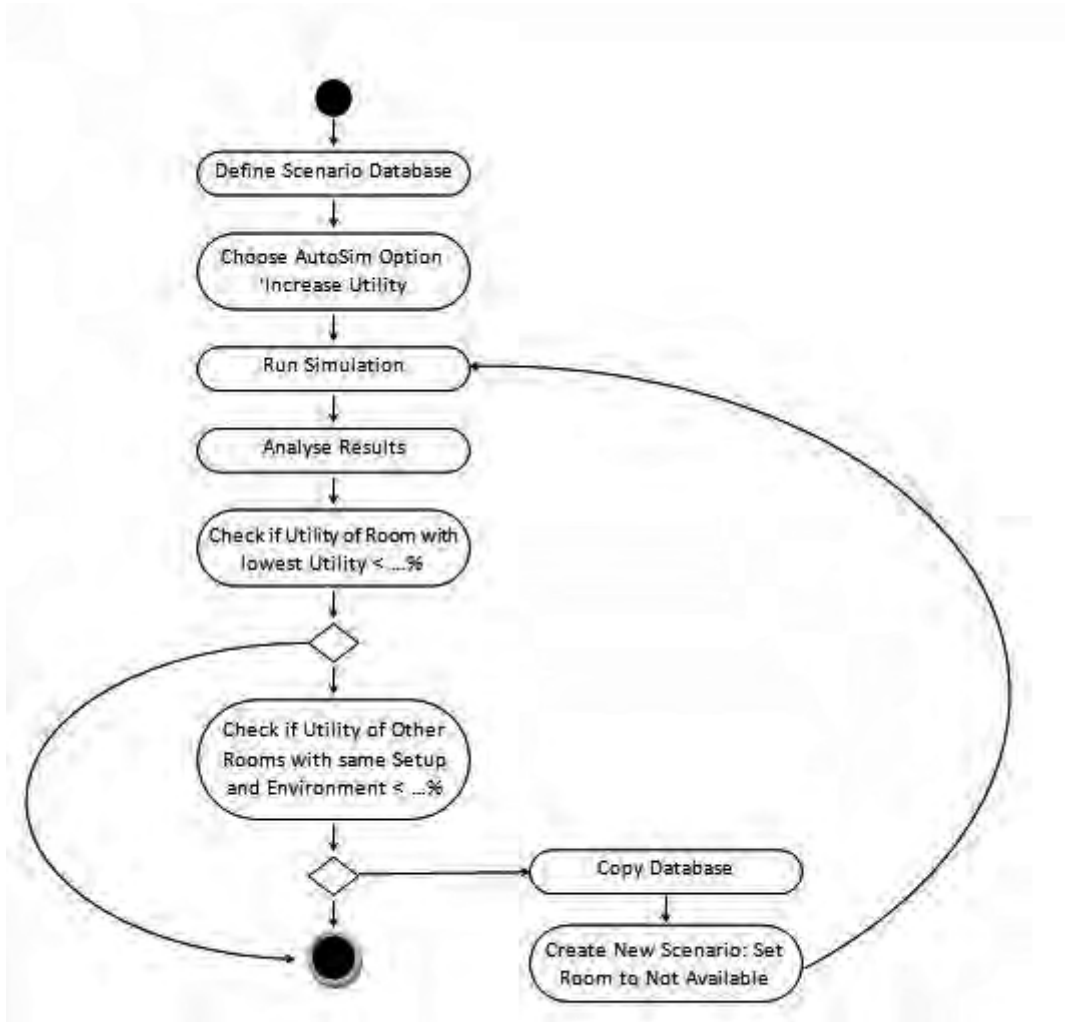


Figure 6-8: Creation of alternative scenario for Room Structure

6.6.2 Increase of Utilisation - Keeping of 'Free' Rooms



The system evaluates the utilisation of the rooms with each room setup category. In case of low utility scenarios are created where less rooms of this setup category are available for booking. This allows reducing the available room until the first cases of not successful booking arise.

Chapter 7: Experimenting with MoreSpace

To create scenarios the two aspects of MoreSpace – design and behaviour – can be modified to test different strategies.

The results of each simulation run are stored in a database. The current version uses Microsoft Access databases and due to its limitations each database does contain the data of only one simulation run. Several databases can be compared in regard to certain key data using Microsoft Excel. Switching to another database system like Oracle would allow keeping all result data in one database and managing the result aggregation without an external tool but. This is planned for the future development, and the database interface using ADO would need only minimal adapting, but it is not implemented yet.

7.1 MS GUI

The MoreSpace GUI has been developed to offer a clear overview over the parameters that can be set for a simulation run. All parameters entered are saved to the database table that is used for the simulation run to document the parameter set used and enable the user to repeat the run at a later time.

The parameters to be set are:

Database - The database to be used for the simulation run must be selected from a pull down menu. If the database in question does not appear it has to be added to the list of available databases using the 'NEW DATABASE' Button. The Tracer Window in the simulation environment shows a message that either confirms the successful connection to the chosen database or reports an error message if the connection could not be established.

Booking Management Rule - The strategy to be used in this simulation run for the assignment of rooms to the courses can be selected from a pull down menu.

It is possible to combine two of these rules. The tracer window shows an affirmation of the selected rules.

Walking Time - The walking time for students from one lecture to the next can be determined in MoreSpace in two different ways: the cellular automata model can be activated and connected to the ED model or the walking time is calculated from a matrix that contains an average value for walking from one room to another. This is set using the according parameter in the GUI. If the cellular automata model is used, the port number and the IP address of the computer running the CA model needs to be entered. If it is not used the probability function for calculating the walking time from the average value can be selected from a pull down menu.

Simulation Run Control - The GUI offers several buttons for starting the simulation run, stopping and resuming. The ED environment does have a run control window, but the simple run command executed there does not involve the model build-up and initialisation required for the successful run of the MoreSpace simulation model.

AutoSim Modus - to use the automatic generation of scenarios as described in section Chapter 6: the AutoSim Modus has to be enabled. The simulation model will automatically evaluate the results of the simulation run and decide if another scenario with altered input data needs to be run. There are three options to choose between: a variation of booking strategies, the reduction of the not successful room assignments and the increase of room utilisation. Depending on this selection the new scenario is created based on the original input data but slightly altered to gain better results. This is done repeatedly, until no improvement can be achieved any more.

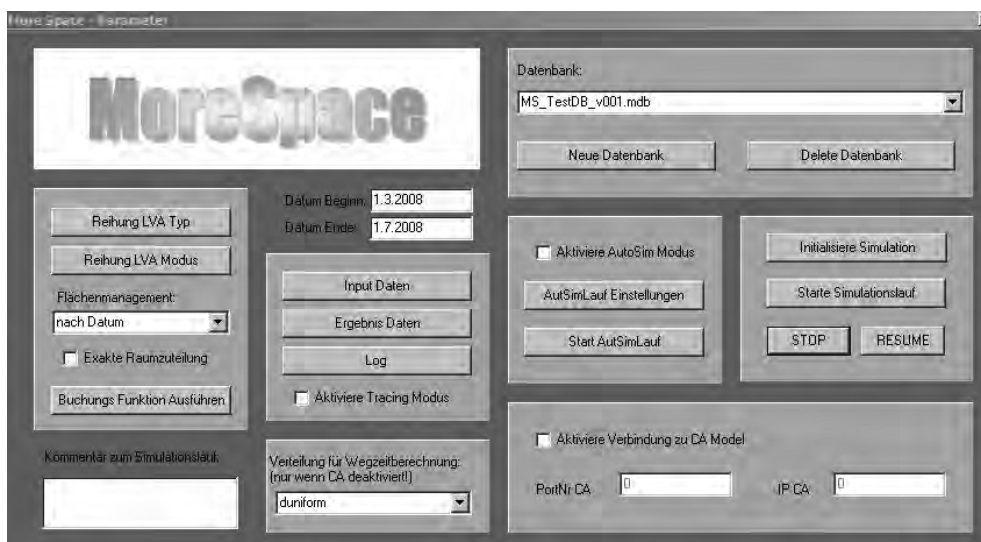


Figure 7-1 MoreSpace GUI

7.2 Changes to Design

7.2.1 Room Structure

Rooms may be added or removed
Buildings may be added or removed
Rooms may be grouped and assigned to different faculties

7.2.1.1 Assigning Rooms to Groups

Rooms and courses can be assigned to groups (institutes, faculties, buildings) in order to restrict certain courses into certain lecture rooms.

7.2.2 Courses

List of Courses may change
Modus of courses may change: more blocking,

7.2.3 Students

Number of students may rise or fall

7.3 Tables for Input Data

7.3.1 Studies

This table contains the list of all studies offered at the Technical University of Vienna as well as their IDs as they will be used in ED. These IDs are unique and consecutively and necessary for correctly and easily addressing in ED.

7.3.2 Courses

This table contains the list of all courses to be held at the TU Vienna, complete with their attributes and the date and time they should take place.

- Course number
- Course category: weekly, blocked, single ...

- Semester
- Begin date
- End date
- Begin time
- End time
- Institute
- Type of course
- Number of expected students
- Number of student exams last year
- Field of study
- Building
- Setup required

7.3.3 Building

This Table contains the list of all Buildings related to the TU Vienna. This list may be changed if needed to achieve certain changes in the simulation model. The object BUILDING can be regarded in a more abstract way as a mean to group rooms together and assign them to certain Faculties or Institutes. It is only essential to also assign the courses accordingly.

- Building ID
- Name

7.3.4 Room Structure

This table contains the list of all lecture halls and their attributes:

- Name
- ID
- Capacity
- Setup
- Institute
- Building

7.3.5 Room Reservation

This table is used to control the availability of rooms during the semester. It contains the information if a room is not available during a certain period of time. The divisibility of rooms can be handled from here as well: the time a large room consisting of several smaller joined rooms is available is noted here as well. This automatically disables the according smaller rooms.

- Name
- ID
- Room
- From Date
- To Date

7.4 Changes to Behaviour

7.4.1 Behaviour of Rooms

7.4.1.1 Locking of Rooms

es können zeitliche Perioden angegeben werden, in denen bestimmte Räume nicht verfügbar sind.

7.4.1.2 Joining and Splitting of Rooms

At certain times rooms can be joined to a bigger room , or a big room can be split into smaller rooms available for two or more lectures in parallel.

7.4.2 Behaviour of Students

Collision of schedule: decision where to go

7.4.3 Booking Behaviour

7.4.3.1 Sorting of Courses for Booking a Room:

The sorting rules have been explained in previous chapters. For experimenting the selection of the sorting order can be variegated to achieve comparable results and allow the analysis of the effect changes have on the utilisation and assignment of rooms.

7.4.3.2 Weekly Courses stay in the same Room

One preference is quite commonly shared by all lecturers at the TU Vienna: the option of keeping the same room for weekly lectures. The additional effort to keep up with the information where the lecture will be held the next week is unwanted by all, lecturer as well as student.

7.4.3.3 Bachelor Studies stay in the same Room

The attempt to keep the students during their first years of study as much as possible in the same room leads to the effect of very good utilization of these rooms. But it also makes the according rooms unavailable for any other lectures. The pro and contra of this approach can be analyzed using MoreSpace.

7.4.3.4 Looking for Best Fit

The strong best fit criterion allows the assignment of a room only if it fits the required capacity perfectly. A room too big or too small is not booked. If this requirement is really of much benefit can be tested by activating the according option.

7.5 Alternative Scenarios

7.5.1 Comparison of Booking Rules

The comparison of the different rules for the assignment of rooms can be tested against each other. The system automatically copies the scenario database and adjusts the parameters to achieve a new scenario with a different set of rules. This is done repeatedly until all possible combinations of rules are simulated.

7.5.2 Reduction of Not Booked Lectures

The system checks the result data and is able to identify a shortcoming of a certain room category during a distinct period of time. A new scenario is created that tries to outbalance the difference between required room and available room by using the divisibility of rooms. I.e. a large room for capacity > 300 is needed but not available the system will try to join several smaller rooms during that time to fulfil the requirement.

7.5.3 Increase of Utilisation - Keeping of 'Free' Rooms

The system evaluates the utilisation of the rooms with each room setup category. In case of low utility scenarios are created where less rooms of this setup category are available for booking. This allows reducing the available room until the first cases of not successful booking arise.

7.6 Tables for Simulation Results

7.6.1 Documentation

The Documentation Table is used to store the parameters given over the GUI to ensure the simulation run can be reproduced. It contains

- the space management rules
- percentage of students
- CA modus
- Probability function for the travelling time calculation
- Simulation begin and end time

7.6.2 Not Successfully Booked Lectures

This table contains the list of all lectures the system was not able to book into any rooms.

- Course number
- Begin date
- End date
- Required Capacity

7.6.3 Room Utilisation

- Room
- Utilization

7.6.4 NichtExact

This list does contain the lectures the system was not able to book into a room with optimal size. This list is used for the second loop in the booking procedure.

- Course number
- Begin date
- End date
- Length
- Type of lecture
- Modus if lecture
- Required capacity
- Required setup
- Number of field of study
- Building

7.6.5 NichtGebäude

This list does contain the lectures the system was not able to book into a room with optimal size. This list is used for the second loop in the booking procedure.

- Course number
- Begin date
- End date
- Length
- Type of lecture
- Modus if lecture
- Required capacity
- Required setup
- Number of field of study
- Building

7.6.6 List of Booked Lectures

This list does contain the lectures the system was able to successfully book into a room during the booking procedure.

- Course number
- Begin date
- End date
- Length
- Type of lecture
- Modus if lecture
- Required capacity
- Required setup
- Field of study
- Building

7.7 Results: Database Reports

7.7.1 Utilisation of Lecture Rooms

This data shows the number of hours each lecture room was booked by the booking procedure. This shows the theoretical utilisation, the time the room is booked, but not the time the room is truly used. As past experiences have shown sometimes rooms may be booked for a lecture that does not take place. This happens if i.e. the lectures of this particular course only take place at the beginning of the semester to be continued by practical work done by the students at home but the room is booked for a weekly lecture. The system allocates the lecture room for the whole semester even though it would only be needed for the first five weeks.

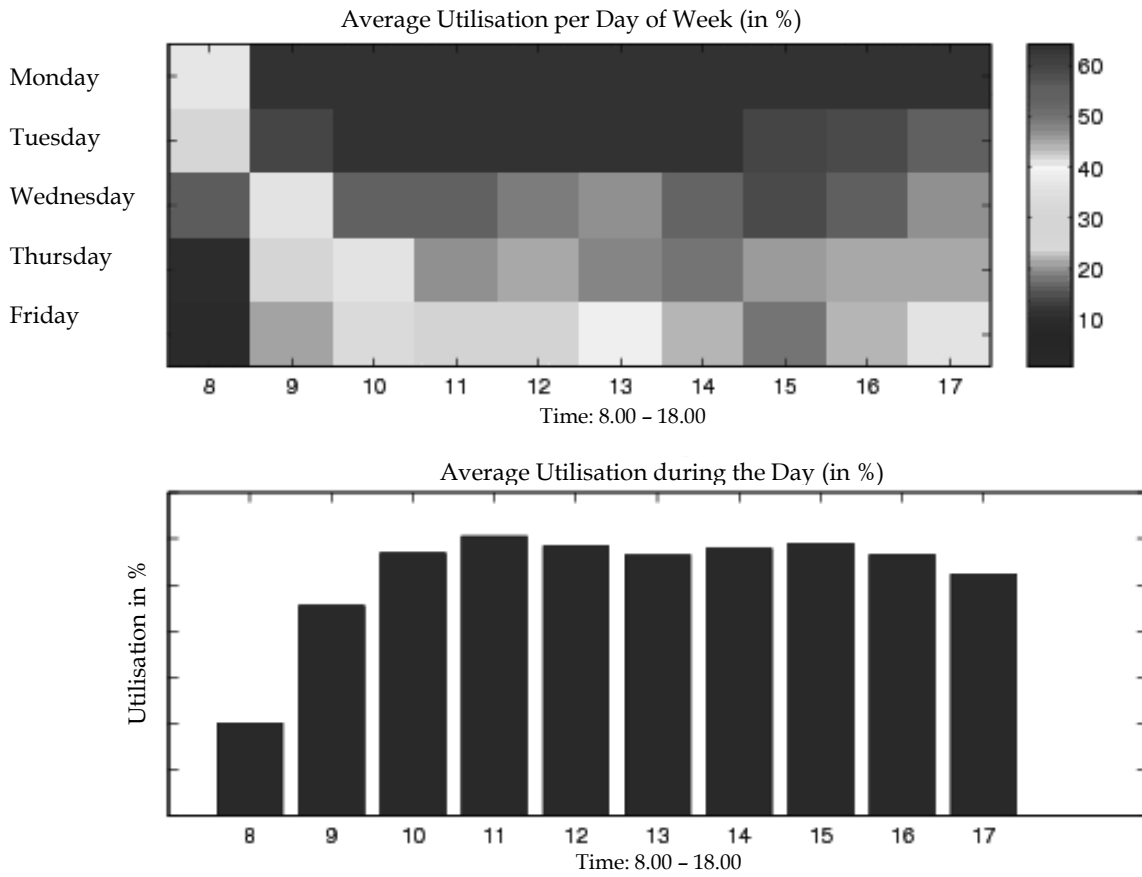


Figure 7-2: Utilisation

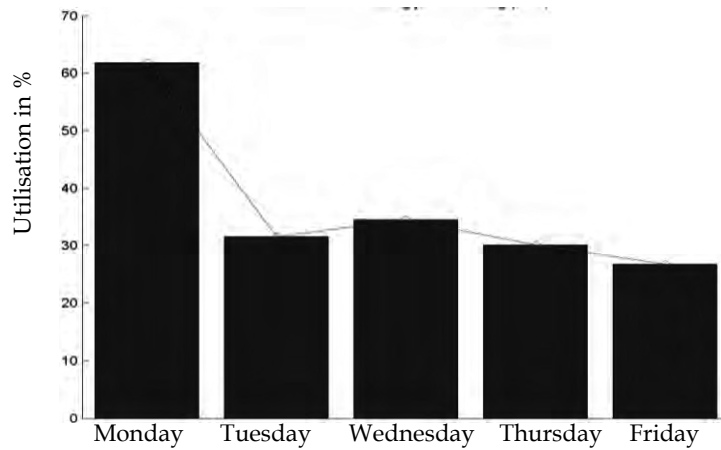


Figure 7-3 Average Utilisation of a Lecture Room per Day of Week

Sometimes rooms are booked at the beginning of the semester for courses that never take place but the reservation is not cancelled.

Sometimes two rooms are reserved in expectation of a high number of students but only one is needed after some time due to a drop in the number of students attending the lecture, but again the reservation is not changed.

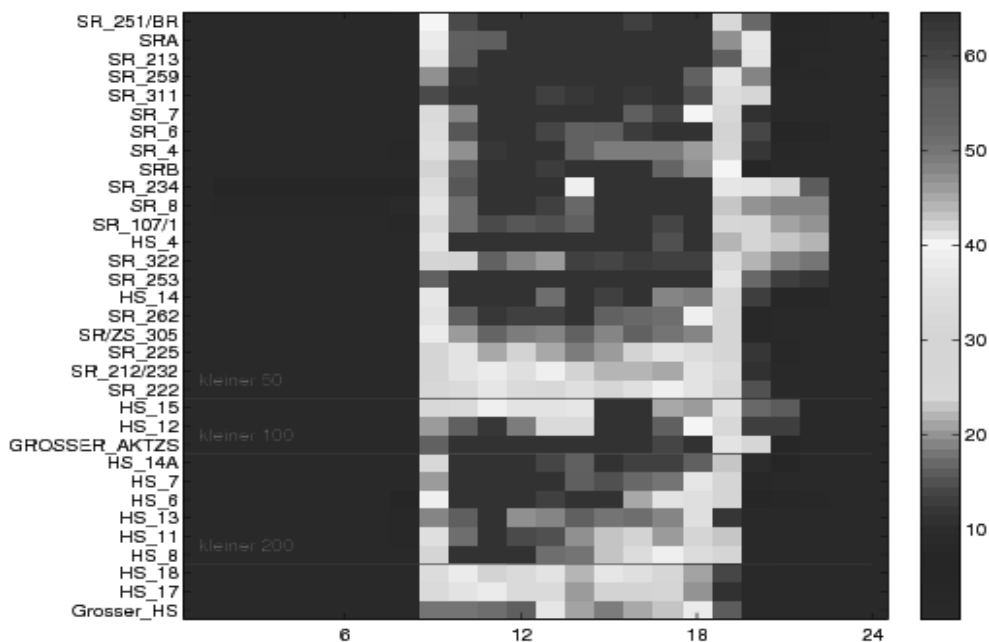


Figure 7-4: Utilisation of Rooms

7.7.2 Capacity Utilisation of Lecture Rooms

This data shows how many students did attend a lecture in the simulation. The number expected is given; the according number of students is assigned. If the number of attending is lower than that it hints at a problem at the accessibility of the course.

7.7.3 Not Successful Booking

The booking procedure tries to find a lecture room for each lecture planned. If it is not able to assign a room the according lecture is listed in this data. For the comparison of several simulation runs one has to make a distinct decision on which aspect to focus the attention.

Depending on this the key data has to be selected.

The following example illustrates how easily data can be misinterpreted in the comparison of two scenarios:

Scenario 1: 1 lecture from 11.00 to 15.30 for 56 students could not be booked in any lecture room.

Scenario 2: 2 lectures from 10.00 to 11.30 for 23 students and from 15.00 to 16.00 for 41 students could not be booked in any lecture room.

	Scenario 1	Scenario 2
Not Booked Lectures	1	2
Not Booked Hours	4.5	2.5
Not Booked Students	56	23 + 41 = 64
Hours*Students	4.5*56 = 252	1.5*23 + 1*41 = 75.5

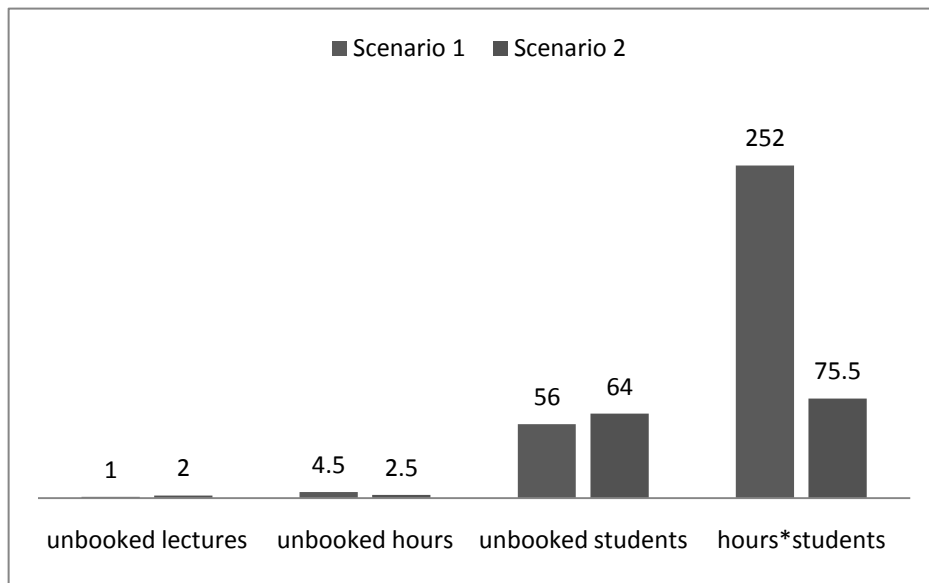


Figure 7-5: Key Data for Not Successful Booking

Figure 7-5: Key Data for Not Successful Booking shows the importance of defining the correct key data; Depending on which value is considered the assessment of the simulation results can be interpreted completely different.

Considering the number of not booked lectures Scenario 1 seems to deliver the better result. The number of not booked hours quickly shows another picture: where in scenario 2 both lectures together result in 2.5 hours that could not find a room, Scenario 1's 1 lecture requires 4.4 hours of time.

The picture again changes if one looks at the number of students that cannot attend a lecture without a room: Scenario 1 is the better one in this regard. But taking the

hours of lecture each student misses into account Scenario 1 suddenly loses highly against Scenario 2 again.

7.8 Accessibility of Lectures

The accessibility of lectures can be interpreted in two different meanings:

Temporal accessibility: this indicates if a lecture overlaps with another lecture.

The spatial accessibility indicates if a lecture can be reached in time: this considers lectures that take place after each other, even with a time gap between them but the location of the rooms is such, that it is not possible to reach the second lecture on time.

While the first kind can be easily determined by evaluating the given data, the second is much more difficult to estimate: the real time it takes from one lecture hall to another depends on far more than the spatial distance: The density of people moving through the corridors, the waiting time at elevators, the distance to staircases influences the walking time. This makes the evaluation of the spatial accessibility to one of the simulation results as it is able to deliver far more accurate results than estimation by distance.

Chapter 8: Conclusion

Using an agent based approach for simulating the students in the MoreSpace project proved to be a good approach to cover several demands:

- * The possibility to model the exact movement of students during the corridors and across the TU Campus, even covering the eventuality of travelling between buildings that are further apart or emergency evacuation simulation.
- * Students have to be regarded as entities with individual preferences and decisions based on their previous behaviour and state.
- * The need to hand control over their actions to the students themselves. They are not routed through the system via a course of server and queues but move on their own.

The implementation in ED did result in a hybrid agent atom: it has the basic attributes that describe an agent:

- * **Autonomy:** each agent acts on its own and decides its own behaviour
- * **Social ability:** agents are able to communicate with each other
- * **Reactivity:** agents react to their environment and changes therein
- * **Pro-activeness:** Agents do not only react to their environment but act on their own as well

But due to the ED configuration the agent atom still holds the basic functionality that relates to the DEVS concept. This enables the interaction with the ED model at certain points according to the general input/output procedure. This is the case every time a student enters the queue in front of a door leading to a lecture hall: the basic configuration of any atom is also present in the agent, thus making it able to react to the event of entering and exiting another atom. This is used to update the state of the agent

For the specific project MoreSpace this solution was the best combination as the atomic agent belongs to both worlds, making it some kind of double agent that is able to use the resources of DEVS as well as ABM.

The agent based approach provides a very detailed insight into the movements and activities of the single students and therefore of the state of the lecture rooms. The

amount of data that can be collected is enormous; the key data extracted has to be clearly defined to allow a useful comparison of several scenarios.

Table of Definitions

Definition 1: Discrete Event System Specification (DEVS)	20
Definition 2: Coupled Model	21
Definition 3: Cellular Automaton (CA)	22
Definition 4: Agent Based Model (ABM)	26
Definition 5: Agent	26
Definition 6: Environment	26
Definition 7: Utility Function	52
Definition 8: Strong Best Fit Criterion	61
Definition 9: Weak Best Fit Criterion	61
Definition 10: Optimal Environment Criterion	61
Definition 11: Room Setup Criterion	61

Table of Figures

Figure 1-1 Predator – Prey Relationship.....	19
Figure 1-2 Simple Server – Queue Model.....	19
Figure 1-3: von Neumann Neighbourhood.....	22
Figure 1-4: Moore Neighbourhood.....	23
Figure 1-5: Torus.....	23
Figure 2-1 Flexibility contra User Friendliness.....	29
Figure 2-2 : Life Cycle of a Simulation Model.....	30
Figure 2-3 User Interaction via GUI.....	32
Figure 3-1: Map of the TU Campus.....	39
Figure 3-2 Original Building on Karlsplatz.....	41
Figure 3-3 Keeping larger Timeslots.....	43
Figure 4-1 MoreSpace.....	46
Figure 4-2 Entity Relationship Model.....	48
Figure 4-3 Structure of Room Model.....	48
Figure 4-4: Course with Mandatory Attendance.....	56
Figure 4-5: Course with Interim Tests.....	57
Figure 4-6: Course without Interim Tests.....	57
Figure 4-7: Number of Attending Students.....	58
Figure 4-8: Model Assembly.....	59
Figure 4-9: Simulation of Booking Procedure.....	60
Figure 4-10: Dynamic Simulation.....	62
Figure 4-11 Simulation Model in ED.....	63
Figure 5-1: Places represented by servers.....	66
Figure 5-2: Pseudo-3D Visualisation of an area with obstacles.....	67
Figure 5-3: Spatial Attributes of an Atom in Enterprise Dynamics.....	68
Figure 5-4 GUI in JAVA with CA of the currently simulated level.....	70
Figure 5-5: About 276.000 Square Meters have to be integrated into the CA.....	71
Figure 5-6: Walking time from room HS1 to room HS2.....	73
Figure 5-7 Simulation Environment of ED.....	75
Figure 5-8: The Atom Editor.....	76
Figure 5-9: ED Eventlist.....	78
Figure 5-10 Attributes.....	78
Figure 5-11 Basic ED Organisation.....	79
Figure 5-12 Lecture Hall with Attributes.....	82
Figure 5-13 Simulation Model 3D in ED.....	83
Figure 5-14: Calculation of the time of event.....	85

<i>Figure 6-1: UML Diagram: Booking Procedure.....</i>	<i>89</i>
<i>Figure 6-2: UML Diagram: Student Generation Procedure.....</i>	<i>91</i>
<i>Figure 6-3 Student Numbers</i>	<i>92</i>
<i>Figure 6-4: Assignment of Compulsory Courses</i>	<i>94</i>
<i>Figure 6-5: UML Diagram: Operational Sequence of MoreSpace</i>	<i>99</i>
<i>Figure 6-6: Screen Shot of Code sending a socket message</i>	<i>101</i>
<i>Figure 6-7: Screen shot of reacting on an incoming socket message in ED.....</i>	<i>102</i>
<i>Figure 6-8: Creation of alternative scenario for Room Structure.....</i>	<i>103</i>
<i>Figure 7-1 MoreSpace GUI</i>	<i>106</i>
<i>Figure 7-2: Utilisation.....</i>	<i>113</i>
<i>Figure 7-3 Average Utilisation of a Lecture Room per Day of Week</i>	<i>114</i>
<i>Figure 7-4: Utilisation of Rooms</i>	<i>114</i>
<i>Figure 7-5: Key Data for Not Successful Booking.....</i>	<i>115</i>

References

- [1] Breitenecker, F. and Solar, D.: *Models, Methods, Experiments - Modern aspects of simulation languages*. In: Proc. 2nd European Simulation Conference, Antwerpen, 1986, SCS, San Diego, 1986, 195 - 199.
- [2] Emrich S., et al., *Simulation of Influenza Epidemics with a Hybrid Model - Combining Cellular Automata and Agent Based Features*, In: Proc. of the ITI 2008 30th Int. Conf. on Information Technology Interfaces, 2008
- [3] Tauböck S., Breitenecker F.: *Features of Discrete Event Simulation Systems for Spatial Pedestrian and Evacuation Dynamics*. In: Proc. PED 2005, Third International Conference on Pedestrian and Evacuation Dynamics 2005, Vienna, Austria
- [4] Bernhard P. Zeigler, Herbert Praehofer, Tag Gon Kim (2000). *Theory of Modeling and Simulation - Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press.
- [5] Bauer Heinz. *Wahrscheinlichkeitstheorie*, Walter de Gruyter, Berlin; New York.
- [6] Barry L. Nelson.(1995). *Stochastic Modeling – Analysis & Simulation*, McGraw-Hill Inc.
- [7] F.E. Cellier: *Continuous System Modeling*. Springer-Verlag, New York, 1991. ISBN 0-387-97502-0
- [8] Robert G. Sargent, *Verification And Validation Of Simulation Models*, Proceedings of the 1998 Winter Simulation Conference, D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds
- [9] Mosteller, Frederick and Tukey, John (1977), *Data Analysis and Regression*, Addison-Wesley.
- [10] V. Volterra. *Variations and fluctuations of the number of individuals in animal species living together*. In Animal Ecology. McGraw-Hill, 1931
- [11] Jennings, R., Wooldridge M., *Agent Technology: Foundations, Applications, and Markets*, Springer Verlag
- [12] Stan Franklin and Art Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*; Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer Verlag, 1996
- [13] Bakthi Satyabudhi, Stephan Onggo, *Running agent-based models on a discrete event simulator*, Department of Management Science, Lancaster University
- [14] Bernhard P. Zeigler, Arizona Center for Integrative Modelling and Simulation, *DEVS Today: Recent Advances in Discrete Event Based Information Technology*

- [15] Gray, L. *A Mathematician Looks at Wolfram's New Kind of Science*. Not. Amer. Math. Soc. 50, 200-211, 2003.
- [16] Goodchild, Longley, *Geospatial Analysis - a comprehensive guide. 3rd edition, 2006-2009* de Smith
- [17] A.M.Uhrmacher, B.Schattenberg, *Agents in Discrete Event Simulation*, European Simulation Symposium - ESS'98, Nottingham, October 1998.

About the Author

Shabnam Michèle Tauböck studied Technical Mathematics at the Vienna University of Technology. She very quickly got into the field of discrete, event-driven simulation and gained great experience in the development of computer-aided simulation models in the field of process optimization and logistics with a special focus on database-driven adaptive simulation models. She worked for several years as a simulation expert for the Austrian Research Centers Seibersdorf and for Profactor Research GmbH. During this time she specialized in the development of hybrid simulation approaches to apply them in various fields from biomedical engineering, clinical studies and supply chain management to production plants and business processes. In 2007 she returned to the Institute for Analysis and Scientific Computing of the Vienna University of Technology. In 2010 she completed her dissertation as part of a research project on the optimal utilization of lecture room resources at university, using a hybrid modelling approach combining agent-based modelling with discrete event simulation systems.



About the Book

This ambitious PhD-thesis is an outcome of the MoreSpace Project of TU Vienna, which is aiming for a better utilization of the lecture rooms. The author, key researcher in this project - introduced dynamic simulation at various levels in order to solve veritable bottleneck problems and lecture room shortages. This volume presents the complex overall simulation model, the generation of this simulation model from databases, and experiments with the model for better lecture room utilization and other system improvements. The simulation model consists of a combination of an agent-based model, describing the behaviour of the students when attending lectures, and a process model for the lecture rooms, including pathways within and between university buildings. The simulation model is gene-rated automatically from university databases: inscription data and curricula data drive the agent-based model, and building maps and lecture room data drive the process model. The prototype implementation in Java and Enterprise Dynamics is controlled by a graphical experimentation interface, allowing various simulation frames: classical semester investigations, long-term simulations over years for planning of university building construction, and short-term simulations for rescheduling in case of interruptions of the schedules - and can be used also for other teaching institutions due to the database parametrization.

About the Series

The ASIM series *Advances in Simulation / Fortschrittsberichte Simulation* (FBS) presents new and recent approaches, methods, and applications in modelling and simulation. The topics may range from theory and foundations via simulation techniques and simulation concepts to applications. As the spectrum of simulation techniques and applications is increasing, books in these series present classical techniques and applications in engineering, natural sciences, biology, physiology, production and logistics, and business administration, upcoming simulation applications in social sciences, media, data management, networking, and complex systems, and upcoming new simulation techniques as agent-based simulation, co-simulation, and deep learning, etc.

The series puts emphasis on monographs with special character, as PhD theses, habilitation treatises, project reports and overviews on scientific projects. ASIM - Arbeitsgemeinschaft Simulation, the German Simulation Society (part of GI - Gesellschaft für Informatik) has founded the series *Advances in Simulation / Fortschrittsberichte Simulation* together with ARGESIM Publisher Vienna in order to provide to the international simulation community a quick and cost-efficient print and e-book series with open access.

ISBN print

ISBN 978-3-903024-85-4

TU Verlag, Vienna, 2019

www.tuverlag.at

ISBN ebook

ISBN 978-3-903347-19-9 DOI 10.11128/fbs.19

ARGESIM Publisher, Vienna, 2016

www.argesim.org

DOI ID

DOI 10.11128/fbs.19