Supporting Work Force Management in Customer Contact Centers

Estimating performance and scheduling shifts

Version:	1.0
Date:	16 June 2005
Author:	Peter Kampstra
Type:	Master's thesis

Supporting Work Force Management in Customer Contact Centers

Estimating performance and scheduling shifts

Peter Kampstra

Master's Thesis

Anago
Achterom 16D
3995 EB Houten

Faculty of exact sciences
Vrije Universiteit
De Boelelaan 1081a
1081 HV Amsterdam

June 2005

FOREWORD

In the final phase of my study Business Mathematics and Informatics a six-month internship is obligatory. I am glad that Anago B.V. in Houten gave me the opportunity to conduct my internship. Although Anago is not very well known (yet), its software and innovations have great opportunities. The month I joined the company, Anago grew even 50% (from 4 to 6 people).

My work at Anago mainly consisted of preparing the Anago software for the facilitation of customer contact center planning. A customer contact center is simply a call center that also does things like handling faxes. Customer contact center planning is an exciting, active area of scientific research (exciting at least to me). The main idea is that I try to estimate the service level, if you know how many calls you expect. Then I turn it the other way around: How many agents do you need to achieve a certain service level? Answering these questions can be made as simple or hard as you want, so it is an ideal internship project.

Many thanks therefore go to Anago, for supplying this great project. Furthermore, I would like to thank:

- My supervisors Thomas de Nooij and Geert-Jan Franx for their guidance.
- Elena Marchiori for being the second reader.
- Wendy Baltussen and Fraser Wilson for their support with the integration within Anago.
- Auke Pot for his advise on the scheduling
- My friends and relatives for their support.
- Anyone who should be here but is missing…

Anyway, have fun reading!

SUMMARY

Staffing in a customer contact centers is a balancing act between keeping the service level high, while keeping costs low and employees happy. The problem originates from the fact that both arrival and service processes are random, with intensities that may vary in time. This makes straightforward calculation of the required staff a hard job. Therefore, simulation optimization is used in this thesis to calculate the required shifts (employee schedules).

Traditionally, (extended) Erlang-C models are used to calculate the expected service level for a certain given call center. These models have been implemented in many software packages, including ours. While making fast estimations, they make more assumptions than necessary, which leads to worse results.
For that reason, a simulation model of a customer contact center has been made. It is able to handle a modern multi-skill, multi-task environment, with outbound calls and preemptive non-telephone work. Validation of the model with data from a small (and therefore harder) multi-skill, multi-task call center shows that the simulation produces good results. The extra computation time of simulation compared to the Erlang models seems to pay off well.

Next to the performance estimation component, a scheduling component using the simulation or extended Erlang model as an estimator was developed. Using a newly developed heuristic based on covering call load and greedy search, good shift arrangements are generated quickly. Other implemented methods like simulated annealing and evolutionary computation are far slower, but also produce good arrangements when run long enough.

Finally, these components have been integrated into Anago, a software platform for management calculations.

As a side-result, to facilitate the simulation, a general state-of-the-art simulation framework was developed. It involves pending event sets, random number generators, random variates and statistical classes. It uses the fastest methods currently found in the literature. These methods are referenced in this thesis. As another interesting result, a flaw was discovered in the Microsoft .NET System.Random class, which makes it almost unusable for reliable scientific simulation.

CONTENTS

1 INTRODUCTION

1.1 Problem description

Customer contact centers handle various customer contacts for a company. These contacts can be calls, mail, fax, email, web chat, etc. Because the moment somebody calls is random, it is hard to estimate the staff needed to handle these contacts. This thesis is about tackling this hard problem of workforce management.

For scheduling employees, the workforce management process typically involves the following steps:

1. Forecast demand
2. **Convert demand forecasts into staffing requirements**
3. **Schedule shifts optimally**
4. Rostering: Assign employees to shifts

This work focuses on solving problem 2 & 3. Step 1 can be easily undertaken using existing Anago software, so that problem is not described further. Step 4 is not even necessary for all situations, but if necessary it might be solved using existing Anago software.

1.2 Anago

1.2.1 Introduction

Anago is a software production company established in Houten, the Netherlands. Anago was founded 6 years ago by Thomas de Nooij (35) and Philip Koenders (40), two management consultants with extensive Management Information, Logistics & Software Engineering experience. Anago develops and sells Anago Studio, a component based modelling environment for management control applications.

1.2.2 Marketplace and solution

Anago focuses on labour intensive organizations such as banks, insurance companies and governmental agencies. Currently, Anago's management control solutions are implemented at organizations like ABN AMRO, UWV, Bouwfonds and Stater.

Management control is an approach to tune the use of resources to the expected workload. The goal is to realize operational targets (i.e. customer service, cost reduction) as efficiently as possible. In practice the approach generates a 5 – 15% efficiency gain on use of resources. Management control solutions are used in all production oriented units such as back offices, front offices, call centres and IT departments.

1.2.3 Anago Studio

Anago Studio is a component based modelling environment to develop tailor made management control applications for all business areas mentioned. Anago Studio offers customers the functionality for a broad range of solutions and to develop them very efficiently.

The product contains high quality mathematical components that support techniques like forecasting, planning, optimization and simulation.

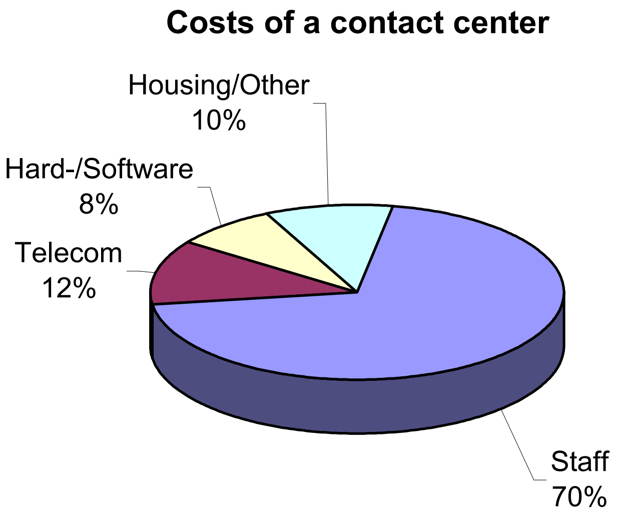
1.3 Further content

In chapter 2, a more detailed description for a customer contact center is given. In chapter 3, work force management is covered. In chapter 4, the customer contact center is simulated to determine statistics like the service level. Then in chapter 5, another approach using an Erlang-inspired model is described. With these procedures, what-if scenarios can be readily made. These what-if scenarios have been tested in a case study in chapter 6. They are also used for the scheduling of agent shifts described in chapter 7. In chapter 8 the what-if and the scheduling components are integrated into Anago. Finally, some conclusions are drawn in chapter 9.

In appendix I, some other software and scientific literature on work force management is described. Appendix IV contains a description of all inputs, outputs and parameters that are used in our Anago engines. In appendix V, a good state-of-the-art simulation framework is described. This simulation framework contains some of the fastest algorithms currently available for simulation problems. Finally, our implementation at work in Anago can be seen in appendix VII.

2 A CUSTOMER CONTACT CENTER

A call center basically is a place where (many) calls are handled. A customer contact center not only handles calls, but also handles other customer contacts like fax, email, chat, etc.
A customer contact center mainly consists of the resources personnel, computers and communication equipment. Typically, the agents who handle calls work in an office space with cubicles. Most of the costs come from personnel costs, as you can see in the following graph based on data from the Dutch association of contact centers (Vereniging Contactcenters Nederland):



An interesting aspect of a customer contact center from a mathematical point of view is that the required number of agents is not easily calculated. It however does forms the biggest cost component.

2.1 Different types of customer contact centers

The software described in this paper is mainly for inbound callcenters. For outbound callcenters, you have a completely different situation. The problem that causes the hard calculation for the required number of agents, namely non-deterministic arrivals, does not occur for outbound calls. Therefore outbound calls are only interesting from a mathematical point of view if they are combined with inbound calls.

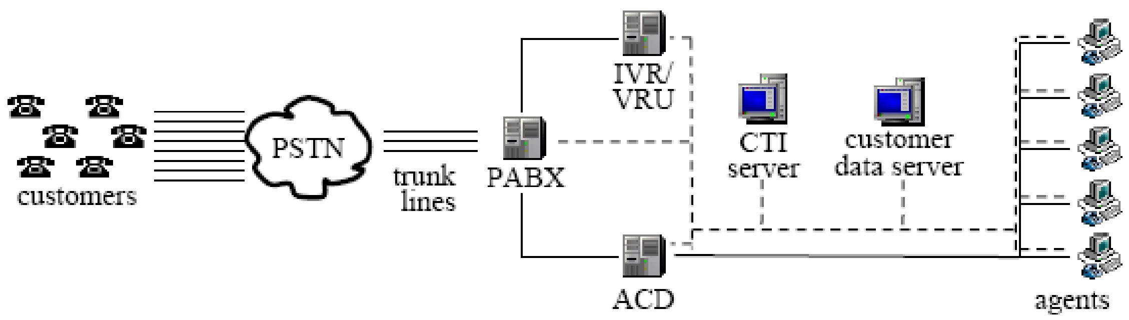
Another interesting difference between callcenters is whether skill-based-routing is used. With skill-based routing, agents can have different skills and calls are routed to agents based on their skills. For example, it might be that only a certain group of agents are able to handle Dutch calls, while all agents are able to handle English calls.

2.2 Typical call handling

Calls enter the call center via the Public Service Telephone Network (PSTN). If the call center has multiple telephone numbers, they can be distinguished via the Dialer Number Identification Service (DNIS). Different customers may also be distinguished via their Automated Number Identification (ANI).

The call then goes from the public network to the private network, managed by the Private Automatic Branch Exchange (PABX). There are a number of telephone lines available to transfer calls, which are called lines, trunk lines or trunks. If there is no line available for a call, blocking occurs.

If no blocking occurs, the call may well be routed to a Voice Response Unit (VRU) or Interactive Voice Response (IVR) unit. These units are specialized computers that allow customers to express their needs. A call may be handled entirely by the IVR. For example, a customer may look up a telephone number without human intervention from the call center site. This equipment is quite expensive, so most of the times only a few options are presented for further routing to employees. For example, this is the typical ‘press 1 for customer services’.



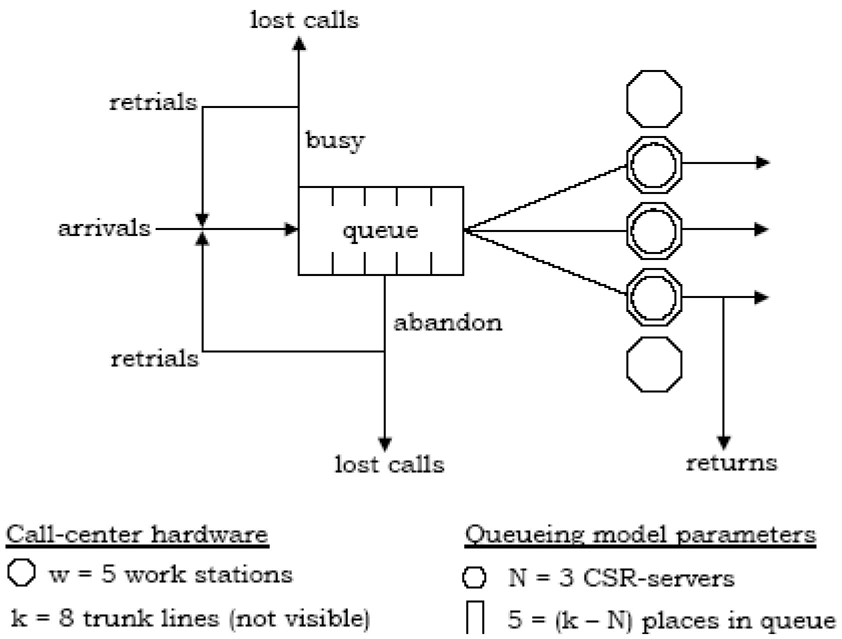
After the call type is determined, the automatic call distributor (ACD) routes the call to an appropriate employee if possible. These employees who answer calls are called agents or

Customer Service Representatives (CSR). The agent may be selected using the required skill(s) of the call; this is called skill-based routing (SBR). If all appropriate agents are busy, the call is queued until a suitable agent becomes available.

Customer data and Computer Telephone Integration (CTI) may be used to assist the agents in their work. If the ANI is known for example, CTI can look up customer data even before a single word is spoken.
If an agent handles the call, the agent may need some time to update data or to do some other stuff. This time is called wrap-up time.

2.3 A model of arriving calls in a call center

The following graph (adopted from [5]) shows a simple model for analyzing a call center. This model clearly illustrates the relationship of call center modeling with queuing theory.



If a customer tries to call the call center, but all trunk lines are busy, this results in a lost call. If not all lines are busy, but all suitable agents are busy, the call is put in a queue. It might be that a customer hangs up while in the queue, this is called abandonment and is another lost call. If a customer from a lost call calls again, this is called a retrial. As it is often hard to gain data on retrials (especially when no number identification is available), retrials are not modeled in our final model. If there are little blocked customers and abandonment, this does not really matter anyway. A good call center will have little retrials.

Normally, perhaps after spending a while in the queue, calls are routed to an appropriate agent (CSR) who works at a workstation. The number of workstations might have some influence if there are more agents in the call center then workstations during a shift switch. Agents

normally work in shifts, which are just schedules with a start and end time (and possibly some pauses).
Even if a call is handled, it might be that the call is not resolved the first time. It might also be that the customer may want to report good service back. If the customer calls again, this is called a return.

2.4 Collected data and measuring performance

For each call, data is collected. This data includes the time the call comes into the system, the skill type of the call, the amount of time spent in the queue, the start and end time, whether the call was an abandonment, etc. From these data, service metrics are calculated. Usually, the current service level is shown on the wall. This service level is the percentage of calls answered before a certain amount of time. Usually, the desired minimal service level is that 80% of the calls should be answered within 20 seconds. This is called a service level of 80/20.

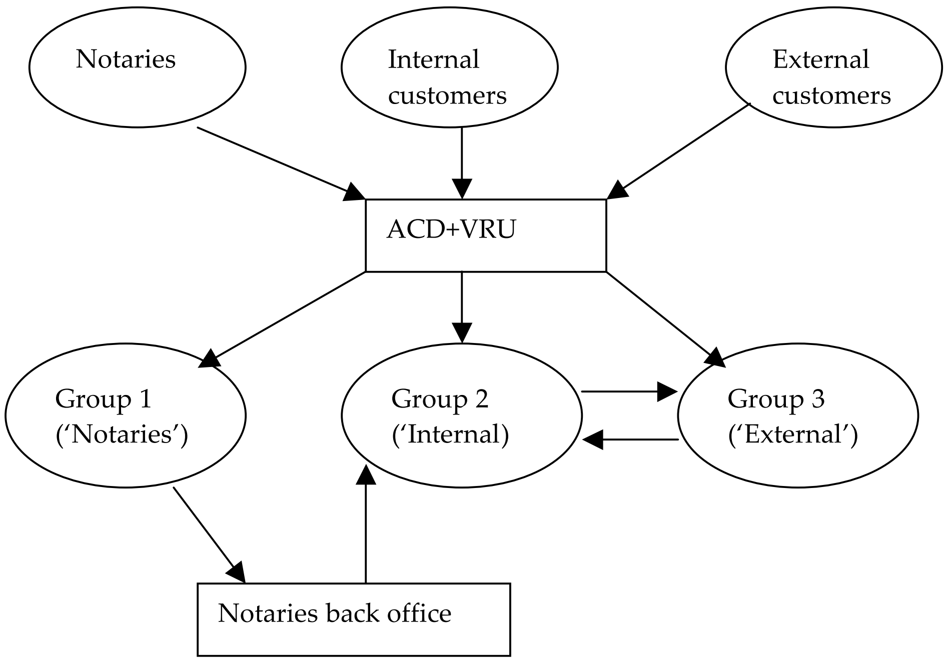
In older systems, only summarized data is stored over 30- or 15-minute intervals. This restriction may also be built into the user interface, so that per call data is available via the underlying database. If no per-call data is available, further research and planning on these data might be less accurate (see 3.4).

Except for the service level, other aggregated reports are possible. The number of incoming calls and the Average Handling Time (AHT) are among these. Another measure commonly reported is the fraction abandonment, which is the fraction of calls that have been aborted by abandonment. The Average Speed of Answer (ASA) or the waiting time is also often reported. The ASA is sometimes used as the main Key Performance Indicator instead of the service level. Another service indication would be the fraction of customers that is blocked. This information is however not directly available, as blocked calls are not seen at the call center. The information that could be used is the duration that no calls can be received, but this is not always recorded.

Of course, in a customer contact center, not only calls are handled. Depending on the procedures, hardware and software used data is also available for emails, fax, etc. The amount of incoming data, the average handling time and the service level are meaningful measures to report.

2.5 Case: a mortgage department

At the mortgage department of bank Z, there are different types of incoming calls. Notaries may call for information, internal customers may call and external customers may call. These 3 different customers can be distinguished by the number dialed (DNIS).



Notaries are top-priority and have absolute priority over the other calls. Notaries can call from 8.30 to 17.30h. Others can call from 9.00 to 17.00h. If a Notary calls, the call is routed to the Notaries group. If there are no agents free in that group, the call is routed to the back office. If there is nobody available there, the call is routed to group 2. If there is nobody available to group 2, the call is routed to group 3. Finally, if nobody is available there, the call is queued. Internal or external customers get routed to their groups, or to the other group (2/3), if no one is available.
Despite the absolute priority of the notaries, it sometimes happens that their service level is lower then the service level of internal customers. This happens for example because the phone is not picked up fast enough, which causes calls to be rerouted. The ACD records this, which may have results for the agent involved if the number gets too high.

Within the VRU, one or two buttons have to be pressed by a customer to determine the call type. Within a group, agents can have priorities (0-6) for each call type. If the priority is six, an incoming call of that type should be preferably routed to that agent. A zero means that the call should never be routed to that agent.

The service level is measured over the day, so if the service level is bad during lunchtime, the afternoon can be used to compensate the bad service level. For all groups, the target is that 80% of the calls should wait less than 20 seconds.

There are 25 ports available for incoming calls, which makes the number of lines available for calls equals to 50+the number of agents calling. This makes sure that (almost) no blocking will occur.

Agents are happy when handling 8-10 calls per hour. If it gets less, talking occurs. If it gets 12 or higher, crying occurs… The problem for the manager is of course to make the customers happy and keep the agents happy, while keeping the costs low. One way to do this is letting the agents do other tasks. These tasks are things like handling faxes.

3 SCHEDULING EMPLOYEES FOR A CUSTOMER CONTACT CENTER

3.1 The Workforce Management Process

As seen in the introduction, the workforce management process for scheduling employees typically involves the following steps:

1. Forecast demand
2. **Convert demand forecasts into staffing requirements**
3. **Schedule shifts optimally**
4. Rostering: Assign employees to shifts

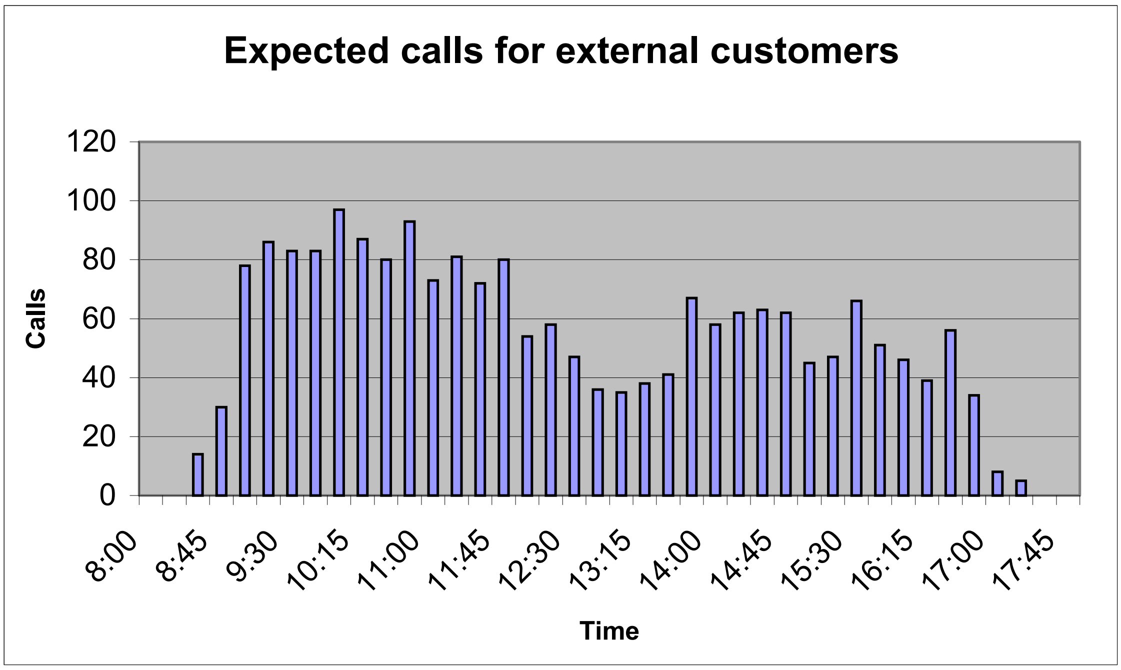
This work focuses on step 2 and 3. Traditionally, step 2 & 3 are undertaken separately, because solving them at once was deemed unfeasible. Unfortunately, the problem with taking step 2 & 3 separately is that it leads to sub-optimization, as there is not a single optimal solution to step 2. If step 2 is taken independently, then there is no way to reliably produce the configuration of agent requirements that fits the shifts best.

Therefore, those steps are tackled using a combined approach. The problem is split into two components targeted at solving both step 2 & 3 at the same time. The following components are used:

1. An evaluator, that given a set of inputs is able to determine the service level
2. A scheduler, that optimizes the staffing by using the evaluator

3.2 Forecasting demand

As seen above, a forecast step precedes our components. The purpose of this step is to determine the stochastic demand that should be handled. Of course, not all hours are equally busy during a day. Therefore, the demand is normally determined for each period of 15 or 30 minutes. The output of this step will therefore be something like this, for each call type (in the example for the type External Customers):



Of course, not only the number of calls needed is required. The service time distribution, the time it takes to handle a call, is also necessary to forecast the demand. In our model, the service time may also vary over the day. The same goes for the distribution of the time until abandonment.

Next to that, other information is required to evaluate a schedule, like the number of agents per period and per group.

3.3 Inputs and outputs

Based on the description of the callcenter and the callcenter workforce management process, we can now sum up the required inputs and outputs for the components that will be developed. More detailed information on these inputs and outputs can be found in appendix IV.

The inputs and outputs for the evaluator will be as follows:
Inputs:

1. The (parameters for the) inter-arrival distribution for each period and each skill
2. The (parameters for the) service time distribution for each period and each skill
3. The (parameters for the) abandonment distribution for each period and each skill
4. The agents available for each period and each agent group
5. The number of lines available
6. The acceptable waiting time for each skill
7. The (parameters for the) outbound call duration distribution for each period and each agent group
8. The threshold for outbound calls for each agent group
9. The threshold for useful time for each agent group

10. The skills that each agent group has

With these inputs, the following outputs are determined using for example simulation:

1. The average waiting time
2. The average waiting time for customers with infinite patience
3. The service level
4. The service level for customers with infinite patience
5. The fraction of blocked calls
6. The fraction of abandonment
7. The agent utilization
8. The time gathered from the situation that more than x agents are free, called the useful time left
9. The number of outbound calls made

So there are quite a lot of inputs and possible outputs. There can be a lot of input required in case of multiple skills, agents groups and/or outbound calls.

For our scheduling component, even more inputs and outputs are required. These inputs and outputs are further discussed in 7.1. An overview of all inputs and outputs for both components is given in appendix IV.

3.4 Common pitfalls to be avoided for estimations

During the study of 'what others have done' (see appendix I), the following problems were spotted.

3.4.1 Assuming a steady state during each period (quarter)

The approach to use an (extended) Erlang model for estimation of the service level assumes that a steady state takes place. The steady state is the state into which a system ends up and remains after a certain amount of time passes when the input parameters stay the same. The problem is however, that in reality this steady state does not take place, because the input parameters vary over time. For an extensive explanation of this problem, see for example [8].

3.4.2 Assuming that only the mean of the service distribution matters

Most call center applications only take the mean of the service time distribution into account. Most of the times, only the mean of the service distribution is reported in call center reports. This information is not sufficient however, as for example our case study (see 6) shows. In [5] the explanation is given that probably the call center process is not understood well enough or that the software is old (and prevents non-existing storage problems). Therefore, if able, the whole known distribution for the service time should be used.

3.4.3 Others

While looking at the work of others, some other mathematical mistakes have been found. The only application that showed a simulation did one run. Simulations using only one run lead to quite unreliable results. Therefore, multiple runs should be done.
Another thing is that simulations generally use a random number generator. The default random number generators can give incorrect results. Testing with multiple random number

generators is recommended. For example, the default random number generator that comes with Microsoft .NET introduced less reliable results in our simulations (see 4.5.1).

4 SIMULATION OF A CALLCENTER

4.1 Implementation

For the implementation of the simulation, a framework was set up to facilitate this kind of simulations. As this falls out of the normal scope of this paper, see appendix V for more information on this framework.

Our framework uses discrete event-based simulation. This means that a set with events that occur in the future is kept. The time consistently jumps to the next event and this event is executed. The time in-between events is not simulated, as nothing relevant for the results occurs there.

4.2 The simulation model

The simulation can be described by describing the possible events that can occur in a callcenter. There are currently 5 different events occurring:

4.2.1 Arrival

Arrival is the event that occurs when an agent arrives into the system. Each Arrival has a certain skill. As only the next arrival is scheduled, the number of scheduled arrivals equals the number of skills.
If an arrival occurs, the following happens:
If there is an agent available, an agent is occupied and a ServiceCompleted event is scheduled. Otherwise, the customer is put into the queue if able, and an Abandonment is scheduled.
After this, the next arrival is scheduled. To make that possible, the next inter-arrival time is drawn from an inputted distribution. If an arrival would fall into a new period (of 15 minutes), the next arrival is scheduled at [the start of that period+a newly drawn inter-arrival time].

4.2.2 ServiceCompleted

ServiceCompleted is the event that occurs when an agent who was busy with a customer is ready.
If a service completion occurs, the following happens:
If there is a suitable customer (with the correct skills) in a queue left, then the agent will start helping the first customer in the queue. If not, then the agent will be made available. If there are more agents available than a certain threshold, an outbound call is done and the agent is not made available. If the agent is not made available, the event is scheduled again.

4.2.3 FakeArrival

Sometimes, Service Levels for customers with an infinite patience are calculated. A customer with an infinite patience is simply a customer that does not abandon itself. To be able to do this, fake customers are put into the system by the FakeArrival event. These customers have service time 0, an infinite time to abandonment and are not counted as members of the queue. The rest of FakeArrival behaves the same as the Arrival event.

4.2.4 Abandonment

This event checks if an associated customer has already left the queue, and if not it abandons the customer from the queue. It is scheduled by Arrival.

4.2.5 NextPeriod

This event adds agents to the agent pool, or removes them from it. An extra variable 'removedAgents' (for each agent group) is used if agents who leave are busy, which is checked at ServiceCompletion. The next NextPeriod is scheduled if needed.

4.3 Doing the simulation

To start up the simulation, an Arrival and a FakeArrival is scheduled per call type. Also, a NextPeriod is scheduled. The simulation ends after a certain amount of time has passed.

Simulating a day in a call center with 100 agents takes less then a quarter of a second (even on an old Pentium II 600Mhz).

4.4 Some specific design issues

This section contains some specific design issues faced when simulating a contact center and some future directions. I decided not to rewrite this section, for a better lookup of these questions. In case you are not looking for a specific question, it is safe to skip this section.

4.4.1 Multi-task

- Will the arrival of email/post/etc. be calculated inside the simulation, or will a 'useful time left' be outputted? For the case of outbound calls (which are non-preemptive) this is of course essential information, therefore they should be simulated.
  Decision: useful time left is outputted, the time that more than x agents were free
- If other distributions then exponential are used for the arrival and service distribution, should the distribution mean be settable only for each period, or for all distribution parameters (including the 'which' distribution to be used)?
  Decision: The distribution parameters are settable for each period and the distribution type is not.
- How do preemptive tasks work? If started, are they always finished? Does the call distribution system know about the preemptive tasks (and schedules non-busy agents first)?
  Decision: Only the time 'useful time left' is outputted and the preemptive tasks are not modeled. This is the time that agents do nothing, not counting free agents equal or less than a certain threshold.

4.4.2 Extension to skills-based routing

- (Is the input agents or agent groups?)
  Decision: It is already decided to schedule agent groups
- How does skill-based routing work in our callcenter:
  - Is there one queue or are there multiple queues?
  - Is the call type known in advance (see putting calls through)

    - Are there routing priorities, and how advanced are they?
- Putting calls through
    - Is there a table with from skill, to skill% (with calls arriving at type “general”) or something like that?
    - How will this be modeled (statistically): is there an “internal” service level and waiting time? Do internal calls have priority over external calls?
- Helping each other
    - Will helping be modeled?

Decision: Putting calls through and helping each other will not be modeled, so the type of the calls is known in advance. 2 routing priorities will be modeled, which will be a model parameter. The first one is the FIFO principle (first come, first served). The second one is FIFO too, but each call type can be assigned extra seconds of waiting time that are gained immediately. For example, some calls may start with a waiting time of 10 seconds, instead of 0 seconds for normal calls.

4.4.3 Redials

- Will ‘outbound redials’ (call backs) be modeled (in case a problem is too difficult to be handled)
- Does the redial distribution differ per time period?

Decision: redials are too difficult to be estimated anyway, so redials will not be implemented.

4.4.4 Next period problems (which are mostly negligible)

- If a next arrival is to be scheduled in the next period, should the arrival be drawn again from the distribution of the next period (with time=[starting time next period]+[sample of distribution of next period])?
  Decision: yes, redrawing is implemented
- Should waiting times/service level (non) compliances that are known in period x, but where the customer arrived in period x-1, be accounted to period x or period x-1?
  Decision: to period x-1
- How are queues being handled after the end time?
- Do agents that leave always finish their current call/email task if they have one? If they do, and they have a break afterwards, will they return later? In this case, should the number of desks available and the number of leaving agents also be inputted?
  Decision: The number of leaving agents will not be inputted. Calls are always finished, and later returns will not be taken into account. The last period, all customers in the queue are handled. If there are no agents available to handle them, they will be discarded.

4.5 Verification

Whenever simulation is used, it is always important to check the results of the simulation. Almost always, numbers will be outputted, but the real question is whether these numbers are correct. In this subchapter, certain test cases will be verified with other possible programs.

Note that this is not the only measure taken to assure a correct working of the program. The other measures include common measures to ensure a correct working of a program. This includes things like clear programming, documentation of each procedure, following guidelines [42], debugging and testing all program lines, etc. Also some trivial test cases have been done (with no agents, no arrivals, etc.).

4.5.1 Steady state behavior: The extended Erlang calculator with abandonment and lines

The steady state behavior of certain simplified call centers (the behavior if the time is unlimited) can be calculated with enhanced Erlang-C models. This assumes that all distributions used are exponential.

The non-trivial test case used here is:

Lines	20
Arrivals/min	5
Service time (min)	2
Agents	10
Average time until abandonment (min)	2
Acceptable waiting time (min)	20/60

For simulation, 100 runs are done with 2 periods of length 150000 min. with these parameters and with seed 1. The outputs for the second run are compared with the exact values.

At first, this verification test was *not* passed, because the Random.nextDouble() function of the .NET Random class did produce bad results. Therefore other methods for creating Random numbers have been implemented and tested too. These other methods did provide the correct results, proving that the problem resided in the .NET Random class. For an explanation of these methods, see V.3.

The following table contains the (approximated) 95%-confidence interval, based on a normal approximation. The exact values have been calculated using the steady Erlang model of the estimation component, which gives the same results as the Erlang-X calculator on the Internet by Ger Koole & Marco Bijvank [1].

Method	Exact	System.Random + The inverse method for generating Exponentials	Ran2 + The inverse method for generating Exponentials	SHR3CONG + The Ziggurat method for generating Exponentials
Time taken (sec)	0	2634	2791	2442
Patient avg. waiting time	0.305025	*0.304495* +/- *0.00046046*	0.304965 +/- 0.00045538	0.305065 +/- 0.00041565
Patient Service Level	0.659817	0.659921 +/- 0.00044883	0.65984 +/- 0.00045614	0.659845 +/- 0.00039897
Fraction Blocking	0.00186905	*0.00171035* +/- *0.000017008*	0.00187112 +/- 0.00001774236	0.00186348 +/- 0.00001873481
Fraction abandonment	0.123671	*0.123501* +/- *0.00016048*	0.123660 +/- 0.00017184	0.123708 +/- 0.000159146
Utilisation	0.874691	0.874579 +/- 0.00018563	0.874708 +/- 0.000172288	0.8747198 +/- 0.000166986
Useful time left	187963	188132 +/- 000278.457	187938 +/- 000258.432	187920 +/- 000250.478

When looking at the results with System.Random, especially the fraction blocking is significantly too low. Recalculating the standard deviation and confidence interval from the individual values per run using Excel leads to the same values. In fact, these analyses showed that for the blocking, all values but one are lower than the exact value.

Another simple test with generating one billion random numbers with seed 1 turned out that the average with System.Random.nextDouble() was 0.499965867, which differs 0.000034132 from 0.5. This half width of an estimated 95% confidence interval[1] was 0.0000182577, so this value lies well out of the confidence interval. Therefore, we can statistically say that System.Random does not function correctly. (In fact, the chance that this value occurs is even less than 0.5%) Because it could be that seed 1 is just a bad seed, the test was repeated with seed 12345 (another seed I frequently use). This resulted in the same problem (average was way too low). These results have been checked for round off errors caused by using doubles. The test was rerun with ExtraAccurateAverageStatistic (see V.5.4), but the effect did not come from round off error.

4.5.2 Steady state behavior: The extended Erlang calculator with outbound calls

There is another calculator by Ger Koole & Marco Bijvank for calculating call center performance [2], which considers outbound calls. It takes the service time the same for both incoming and outbound calls. Blocking and abandonment are not supported. For a test, this is not a problem however.

The non-trivial test case used here is:

[1] Even a broad one, because 2 was used as the multiplication factor of the standard deviation instead of 1.96

Lines	1000 (infinity, the calculator does not support blocking)
Arrivals/min	6
Service time (min)	3
Agents	25
Average time until abandonment (min)	Infinity (the calculator does not support this)
Acceptable waiting time (min)	20/60
Threshold for outbound calls	6

The results are:

Method	Exact	SHR3CONG + The Ziggurat method for generating Exponentials
Time taken (sec)	0	2520
Patient avg. waiting time	0.0797459	0.07985247 +/- 0.0003328368
Patient Service Level	0.9145129	0.91444156 +/- 0.0002867622
Utilisation	0.8684474	0.86846007 +/- 0.000080492558
Useful time left	493322.189	493274.734 +/- 000301.847
Outbound calls	185559.270	185581.26 +/- 000205.366

For simulation, 100 runs are done with 2 periods of length 150000 min. with these parameters and with seed 12345. The outputs for the second run are compared with the exact values.

4.5.3 Non-steady state behavior: Comparing Uniformization with Simulation

For the non-steady situation, we simulated 4 similar, non-trivial quarters. The test case used was:

Lines	1000 (infinity, the calculator does not support blocking)
Arrivals/min	6
Service time (min)	2
Agents	13
Average time until abandonment (min)	3
Acceptable waiting time (min)	20/60

The simulation was done with 10.000 runs, with seed 1, using 60 seconds computation time (on an old Pentium II 600Mhz).

At first, this test was not passed, as the uniformization method had some minor problems, leading to a difference of a few percent. Later, that problem was fixed. The uniformization values look like the values with simulation, and the small difference can be explained by the fact that uniformization does not give entirely exact results (due to some numerical approximations).

		Period 1	Period 2	Period 3	Period 4
Patient avg waiting time	Simulation	0.137672 +/- 0.002321	0.19619 +/- 0.002819	0.197878 +/- 0.002819	0.197254 +/- 0.002812
	Uniformization	0.135261	0.194868	0.194946	0.194946
Patient Service Level	Simulation	0.833517 +/- 0.002796	0.762535 +/- 0.003359	0.759728 +/- 0.003378	0.761177 +/- 0.003357
	Uniformization	0.836303	0.763511	0.763416	0.763416
Blocked %	Simulation	0.009303 +/- 0.000411	0.015082 +/- 0.00053	0.014828 +/- 0.000523	0.015112 +/- 0.000532
	Uniformization	0.010177	0.01613	0.016141	0.016141
Abandonment %	Simulation	0.038842 +/- 0.000702	0.055662 +/- 0.000836	0.056375 +/- 0.000847	0.056325 +/- 0.000845
	Uniformization	0.039008	0.05788	0.057907	0.057907
Agent utilization	Simulation	0.754585 +/- 0.001627	0.855134 +/- 0.001605	0.856363 +/- 0.001584	0.855361 +/- 0.00158
	Uniformization	0.751101	0.85553497	0.855587	0.855587
Agent usefull time left	Simulation	47.85583 +/- 0.317303	28.24894 +/- 0.313031	28.00916 +/- 0.308822	28.2047 +/- 0.308158
	Uniformization	48.53525	28.17068	28.16046	28.16045

Some other tests were conducted with other values, and shorter (1 minute) periods, which were also passed. At first, periods with 1-minute intervals did not produce simulation outputs, because sometimes there are no arrivals within that minute. This problem was fixed by ignoring these intervals.

4.5.4 Non-steady state behavior with unstable parameters: Comparing Uniformization with Simulation

One of the non-trivial test cases used here is:

Period	1	2	3	4
Lines	30	30	30	30
Arrivals/min	6	4	8	5
Service time (min)	2	2.2	2.1	2
Agents	13	14	18	14
Average time until abandonment (min)	2	3	3	2
Acceptable	20/60	20/60	20/60	20/60

waiting time (min)				
Threshold for outbound calls	1	1	1	1

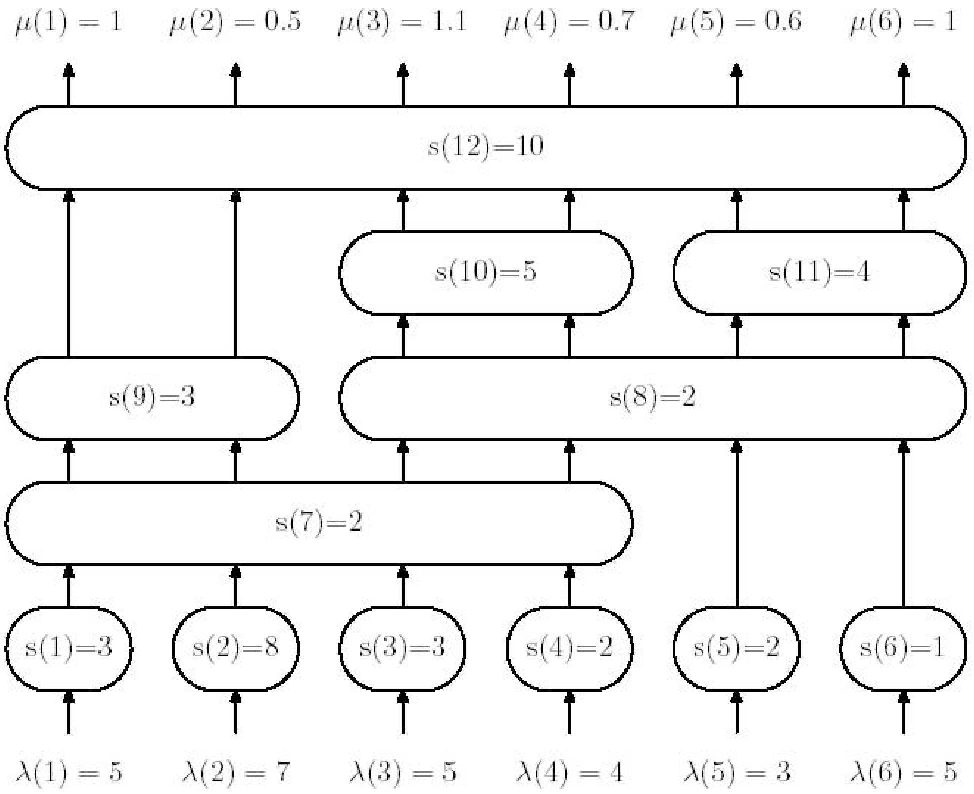
The results obtained are:

		Period 1	Period 2	Period 3	Period 4
Patient avg waiting time	Simulation	0.32814 +/- 0.00259	0.18595 +/- 0.00176	0.35066 +/- 0.00277	0.26237 +/- 0.00212
	Uniformization	0.33101	0.19732	0.33561	0.21762
Patient Service Level	Simulation	0.60743 +/- 0.00281	0.78677 +/- 0.00215	0.58791 +/- 0.00312	0.69814 +/- 0.00235
	Uniformization	0.60552	0.77262	0.59382	0.74447
Blocked %	Simulation	0.00001 +/- 0.00001	0.00000 +/- 0.00000	0.00395 +/- 0.00023	0.00010 +/- 0.00003
	Uniformization	0.00001	0.00000	0.00414	0.00000
Abandonment %	Simulation	0.13624 +/- 0.00107	0.05489 +/- 0.00071	0.10322 +/- 0.00083	0.10876 +/- 0.00096
	Uniformization	0.13671	0.06075	0.09803	0.0992
Agent utilization	Simulation	0.98239 +/- 0.00013	0.97052 +/- 0.00014	0.98968 +/- 0.00009	0.97766 +/- 0.00012
	Uniformization	0.98234	0.97109	0.98958	0.97589
Agent usefull time left	Simulation	3.43475 +/- 0.02462	6.19172 +/- 0.02848	2.78575 +/- 0.02318	4.69215 +/- 0.02608
	Uniformization	3.44387	6.07109	2.81275	5.0623
Outbound calls	Simulation	32.56390 +/- 00.17249	37.31800 +/- 00.20835	25.25720 +/- 00.21141	30.31750 +/- 00.19988
	Uniformization	32.6632	36.08706	25.56591	32.90494

The results are quite consistent. The last period has a relatively big difference. This difference can be explained by the way leaving agents are handled. If there are agents leaving the system, and there are more calls in the system than the new number of agents, different approaches are used. In reality, some agents will finish their call before they leave. With uniformization, a random sample of the current calls is picked and these calls are taken out of the system. With simulation, the sample is not random, but instead the first finished calls are taken. The calls also do not leave the system until the calls are really finished (which has a different effect for the blocking). It is therefore expected that the calculated service level is worse with simulation than with uniformization.

4.5.5 Skill-based routing comparison

There is a simulator by Auke Pot [3] available on the Internet for this purpose, which was used for several scientific publications. A few cases were tested, including:



With 6 call types, 12 groups and overflow routing with queuing. This is test instance 17 of [23].

The results, with queuing turned on for 10 runs and seed 1 are displayed in the following table. Note that SA-SIM does not support multiple runs, so it was run 10 times with seed 1-10. The values after '+/-' indicate an estimated 95%-confidence interval halfwidth.

	Runtime	SL 1	SL 2	SL 3	SL 4	SL 5	SL 6
SA-SIM	84020 sec	0.804154 +/- 0.0013802	0.786704 +/- 0.001090888	0.916015 +/- 0.000960913	0.889239 +/- 0.001221956	0.822262 +/- 0.001216603	0.793197 +/- 0.001226348
Our simulation	801 sec	0.805590 +/- 0.001660	0.788155 +/- 0.001916	0.916137 +/- 0.001009	0.888963 +/- 0.000936	0.823334 +/- 0.001707	0.794186 +/- 0.001758

The values differ less then 0.002 and lie within each other's 95%-confidence interval. This gives some confidence in both simulations. The difference in runtime is quite astonishing though.

5 ESTIMATION OF A CALLCENTER USING EXTENDED ERLANG MODELS

5.1 An extended Erlang-C model

Most callcenter WFM applications in this area feature an extended Erlang model. For more information on other applications in this field, see appendix I.
An Erlang model is a so-called Markov chain, where only the current state of the call center matters for analyses and not all the possible past events. The current state used is the number of customers in the system. For each number of customers in the system, a probability can be calculated that it occurs. From the probabilities of being in a certain state, expected performance metrics like the service level can be calculated. In schema, this calculation is shown for a simple situation in appendix II.

Most of the times, callcenter WFM applications only calculate the steady-state probabilities, as on the long term the state probabilities go to a certain steady situation. In most callcenters however, this situation does not happen in practice, because the period a certain configuration takes place is too short. Scientific studies show that predicting the service level with this steady state model can sometimes lead to bad results [8].

Therefore, another methods which does not use this steady state model is also implemented, namely the randomization method or uniformization method.

5.2 Calculating the probabilities of being in a state

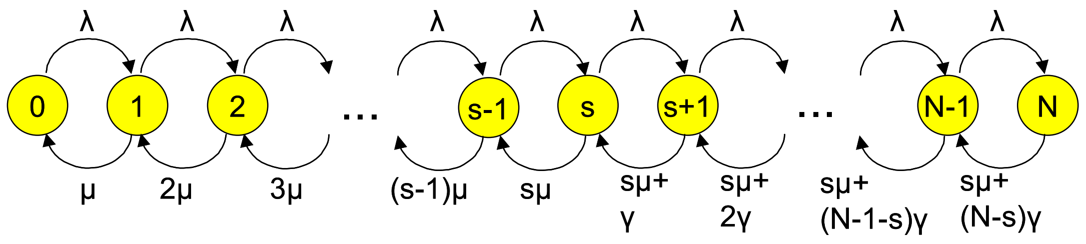
5.2.1 Possible events & Markov chain diagram

In our simple model, the following events (may) take place:

a. A customer arrives into the system
b. A customer is served and leaves the system
c. A customer abandons the system, before being served

Event a adds one to the state, while event b and c subtract one from the state.
If we define:
λ the arrival rate (so λ=2 means 2 arrivals per time-unit on average)
μ the service rate (so γ=3 means that the average service time is 1/3)
γ the abandonment rate (so γ=2 means that the average time until abandonment is 1/2)
s the number of agents
N the number of lines available

Then we can now show the state transitions in a Markov chain diagram:

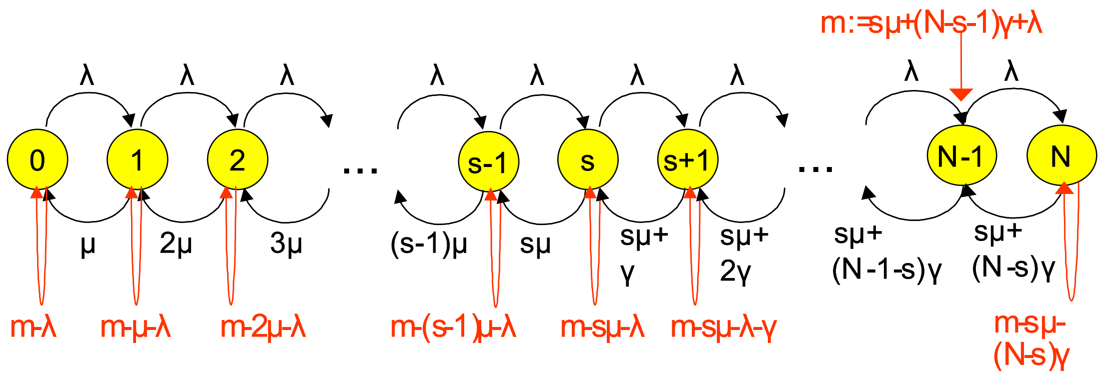


So for example in state 2, when there are 2 customers in the system, the following holds: With rate 2 μ you go to state 1 (because 2 customers can have a service completion) and with rate λ you go to state 3 (if an arrival occurs). Because of the properties of the exponential distribution, the chance of going to state 3 is $\lambda/(\lambda+2\mu)$, while the chance of going to state 1 is $2\mu/(\lambda+2\mu)$.

The steady state probabilities, the probabilities that occur after enough time has passed, can now be calculated. To do that, you can start in state 0. The probability of being in that state is unknown, lets say equal to p_0. The probability of being in state 1 then is $p_1=p_0*\lambda/\mu$. The probability of being in state 2 then is $p_2=p_1*\lambda/(2\mu)$. This way, all the probabilities can be calculated using the formula [the probability of the previous state]*[the incoming rate from that state]/[the outgoing rate back]. These probabilities are then only dependent on p_0. Since all probabilities of the states must sum up to one, we can calculate p_0. (In practice all probabilities are calculated as if $p_0=1$, and finally the probabilities are normalized so they sum up to 1.)

5.2.2 Uniformization/randomization (non-steady state)

If the rates in and out of each state are not the same, the probabilities after a certain amount of time cannot be calculated. Therefore dummy in and out rates are added to this system, to make sure the rates in and out are the same. This technique is called uniformization or randomization and was introduced for call center analyses by Ingolfsson [8]. Therefore, we get the following Markov chain:



If we know the starting probability and how many state transitions are being made during a period in this new system, we can calculate what the state probabilities are at the end.

The number of state transitions has a Poisson distribution. The probability of having x transitions in case of a rate of m is:

$$P(x,m) = \frac{e^{-m} m^x}{x!}$$

Note that this probability has a nice recursive relation in case you want to calculate it for multiple values of x (namely P(x, m)=P(x-1, m)*m/x).

The starting probability is given. If we know that there have been zero events (which occurs with chance $e^{-\lambda}$) then we know the probabilities at the end, because they are the same. We can also calculate the probabilities if exactly one event has occurred. We do this by using a property of the exponential probability. We have a rate A to the pervious state, a rate B to next state and a rate C to the same state. The probability of going to state A is A/(A+B+C). For each state, A+B+C is the same, because we have defined C that way. In case of more than one event, we can simply apply the procedure for one event multiple times.

So we know the probabilities if x events have happened and the probability that x event happen. With this information we can use conditioning over the amount of events. The state probabilities are simply calculated using:

$$state\_probability(i) = \sum_{x=0}^{\infty} P(x,m) * state\_probability\_after\_x\_events(i)$$

In reality we cannot sum until x reaches infinity. Therefore we stop summing if P(x,m) becomes low enough. There is also a numerical problem if the poisson rate gets too high. We overcome that problem by splitting the period in parts in order to lower the poisson rate.

5.2.3 Further issues with Uniformization/randomization (leaving agents, sampling)

If there are agents leaving, it is not entirely true that the state probabilities at the start of a period should be equal to the probabilities at the end of the previous period. The problem is that calls will not suddenly be put back into the queue. In most situations, agents will finish their calls. Therefore we use the following algorithm. If there are more calls already taken than agents in the period, the surplus is subtracted from the state. For example: Let us pretend there are 8 agents, while the state was 13 and in the previous period there were 10 agents. The corrected state used in that case is 13-(10-8)=11.

Note that this behavior is not entirely the same as we used with simulation. With simulation, we assumed that the *first* calls following the period ending were the calls that leaving agents took. With uniformization, we assume that *random* calls are 'taken away' by leaving agents.

Another problem is the sampling of the performance metrics. A period is split in a number of sub-periods, which we set to 30 on default. For each subperiod, the performance metrics are calculated. These performance metrics are then averaged. (The average is weighted so that the start and end measures of the period are only counted for .5, while the in-between measures are fully counted. The weight therefore equals the number of periods.)

5.3 Calculating performance metrics from state probabilities

From the state probabilities, performance metrics can be calculated. For different performance metrics, different tactics are being used.

5.3.1 The blocking probability

Customers who arrive into the system, see the average state the system is in if the arrivals are poison. (This is the so-called Poisson Arrivals See Time Averages (PASTA) property.)
As we have exponential inter-arrival times, this is always the case. Therefore, the blocking probability is simply the probability of being in state N.

5.3.2 The average waiting time

The state probabilities of the system give us the probability that there is a certain number of customers in the system. If we know that there is a certain number of customers in the system, we can calculate the expected waiting time. By conditioning, we get the average waiting time for an arbitrary arriving customer.

With the probability of being in state s, the number of customers in the system is equal to the number of agents. This means an arriving customer will be the first customer in the queue. The expected waiting time for this customer then is 1/(s * μ). For a customer arriving when the queue length is one, the expected waiting time is 1/(s * μ)+1/(s * μ+ γ). And for a customer arriving when the queue length is two, the expected waiting time is given by:

$$Expected\_waiting\_time(queuelength) = \sum_{i=0}^{queuelength} \frac{1}{s \cdot \mu + \gamma \cdot i}$$

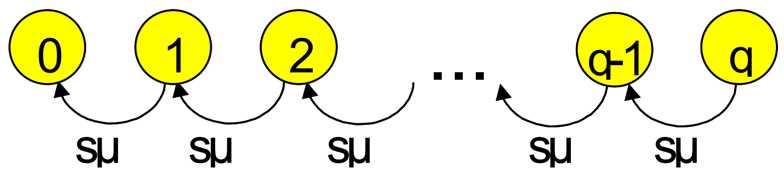
So by conditioning, the average waiting time can be calculated (So the result is the sum of i over all states of [probability of system in state i]*[expected waiting time if system in state i]).

Note that this average waiting time is the average waiting time for customers that do not abandon. So the empirical average waiting time will be slightly different.

5.3.3 The service level

The service level can be determined by solving another uniformized markov chain. We want to determine how many customers will have to wait less than r seconds. If we know the chance of having a certain queue length, which we know, then we are able to calculate this. If the queue length is 0, the service level is of course 1 (because the customer does not have to wait).

We use another markov chain to calculate the probability that the customer will be helped soon enough. The state is the queue length right after the arrival of a customer (so state q is equivalent to state s+q-1 in de previous Markov chain). Now we want to calculate the probability that the chain will be in state 0 in less then r seconds.



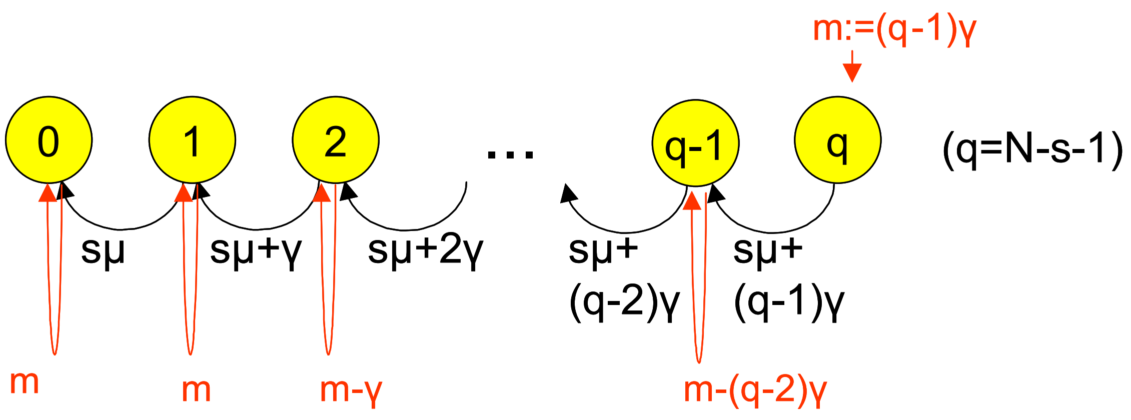
As the chances of having x events in a Markov chain are Poisson, we can state the formula to determine the service level, without abandonment:

$$service\_level = \sum_{q=0}^{n-s+1} \frac{e^{-s\mu}(s\mu)^q}{q!} * p_q$$

(With p_i the probability that the system is in state i at the end of a period (of the original system).)

If there is abandonment, the same idea can be used. We only need to use uniformization again, because the queue length decreases if somebody abandons the queue.

Therefore the probability that a customer has to wait less than r seconds when abandonment can occur is the probability that q or more events occur in a Poisson(s*(μ+λ)) distribution. The rest of the calculation is the same as the calculation of the service level without abandonment. Note that Uniformization is needed here again to calculate the probabilities after a certain amount of time. We calculate the service level for customers that do not leave.



Note that this service level is the service level for customers that do not abandon. So the empirical service level will be slightly different.

5.3.4 Queue length

We know the chance of having a certain queue, so we can easily condition to get the queue length.

5.3.5 Abandonment

The following relation is used to calculate the abandonment:

Abandonment=Queue length* γ / (λ * The chance of not being blocked)

5.3.6 The utilization

If we know we are in a certain state, we know how many agents are busy, and consequently know what the utilization is. Hence, we can use conditioning again to calculate the utilization.

5.3.7 The useful time left

If we know we are in a certain state, and how long we are in that state, we know how much useful time we get. Consequently we can once again use conditioning.

5.3.8 Outbound calls (extension)

Outbound calls are a simple extension to the previous Markov chain model. In this case, there cannot be fewer calls than a certain threshold in the system. This is because if there were fewer calls, an outbound call was done. It is assumed that outbound calls have the same service duration as inbound calls, which is a major limitation of this method.

An Outbound call is done if we are in the lowest possible state (agents-threshold) and we have had a service. We know the fraction of time that we are in the lowest possible state, and we know the rate that services occur if we are in that state. Therefore we can calculate the outbound calls with outbound calls=[probabilitiy in lowest possible state] *[period length]*[lowest possible state]*μ

5.4 Extension to multiple calls, skills, etc.

Our simulation model is able to tackle multiple calls and multiple agent groups. It is also able to tackle outbound calls with a different service time distribution. Our current (uniformized) Erlang model does not support this.

To be able to use the same interface for both models, some simple manipulations of the inputs are used. The agents for all groups are summed up. This number is used as the number of agents. The inter-arrival time is averaged. The service time is averaged, weighted by the expected number of customers. This is done with the abandonment too. In case of non-exponential arrival times, the mean of the distribution is calculated and the distribution is seen as exponential. In case of outbound calls with a different service time, the service time is averaged. The average is a weighted average by the number of expected calls. The number of outbound calls is not known, but estimated by the time agents are available and the load of the normal calls (it is in fact over-estimated as it is expected that agents will be busy all the time).

These extensions are generally not really accurate, but they do make sure that some estimation is given under these circumstances, so that the Erlang model is compatible with the simulation model. It is possible to do a lot better by using different models.

5.5 Verification

See 4.5 where it was done.

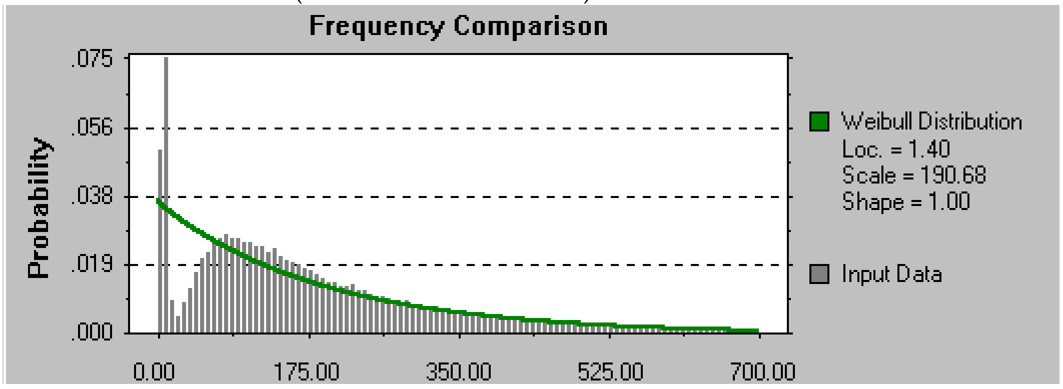
6 A CASE STUDY FOR THE ERLANG AND SIMULATION MODEL

The case of a small call center at a mortgage department introduced in 2.5, was used as a test case for the Erlang and the simulation model. This chapter contains some of the interesting things that have been observed.

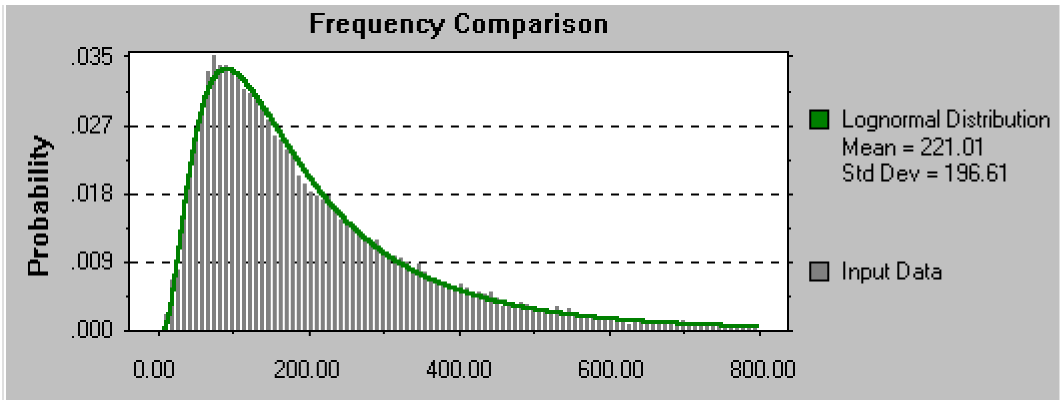
6.1 Data analyses

6.1.1 Call duration

One of the main inputs of our model is the distribution of the call duration. In our model, this time includes the time needed for taking the call. Not only the mean of this distribution matters; especially in a small call center the whole distribution matters. The distribution in this call center looked as follows (call duration in seconds):



There are quite a lot (6%) calls that have a very short duration. If we leave calls shorter than 15 seconds out of the comparison, we get something that looks a lot like a lognormal distribution. This result agrees with the literature for this situation (see [5], which contains about the same example, also with a problem with very short calls, which had another reason).



6.1.2 Further analyses of the short calls

We looked at the individual call level to gain more details about the very short calls. It appeared that most, if not all of these calls were made from internal telephone numbers. From some internal telephone numbers, the call center was dialed for the wrong reason over and over again. From external telephone numbers, almost no calls with a very short duration were seen. (Due to privacy considerations the corresponding data/graph is not shown in this report.)

6.1.3 Call priorities

In the case description (see 2.5), it was said that notaries have absolute priority over the other customers. By looking at the individual call level, it was shown that this was in fact not the case.

6.1.4 Examining bad service levels

One of the interesting aspects of having information at individual call level is that you can examine what agents were doing when the service level was not met. Using a random sample from this data, it seems that in case of a bad service level almost always at least one agent is doing something else. In case of a good service level, it sometimes happens that everybody is 'Ready'.

One of the main inputs of the model is the number of agents that are really available. Without considering agents in 'NotReady' state, validation did not work. It is clear that the number of agents being 'NotReady' has great impact on the service level.

In the current model, all agents that are available are really available at all times. It is therefore expected that there is no relation between the business and the moments that agents go 'NotReady' or leave the system. In the following graph, a comparison is shown for the business (length of the queue) between some random moments and the moments that agents left the system. It appears that there is almost no relation:

	Aug 2004		Jan 2005	
Queue length	Average	When leaving system	Average	When leaving system
0	80.18%	79.77%	88.92%	89.83%
1	6.61%	7.11%	5.57%	5.51%
2	3.21%	3.98%	2.74%	2.74%
3	2.50%	2.38%	1.27%	0.92%
4	1.48%	1.22%	0.85%	0.68%
5	1.15%	1.36%	0.24%	0.14%
6	0.77%	0.82%	0.10%	0.07%
7	1.03%	0.85%	0.03%	0.07%
8	0.58%	0.61%	0.10%	0.03%
9	0.71%	0.44%	0.07%	0.00%
10	0.06%	0.31%	0.10%	0.00%
11	0.19%	0.34%	-	-
12	0.06%	0.14%	-	-

13	0.06%	0.00%	-	-
14	0.19%	0.14%	-	-
15	0.45%	0.10%	-	-
16	0.26%	0.20%	-	-
17	0.38%	0.14%	-	-
18	0.00%	0.07%	-	-
19	0.13%	0.03%	-	-

It seems that there is very little relation between the business and the leaving of agents. It has not been tested statistically (no need to), but it could even be that you cannot statistically find a relation.

The advantage of this situation is that the models premise holds, but of course it has some disadvantages for the callcenter involved. If agents only leave when there is no queue, the service level probably would have been better.

In the next table it is shown what the status of the agents was at some random moments the service level was or was not met. Being idle means that certain agents were not available because of skills or because of a 15 second prescribed idle time after taking a call.

Time	sl	loggedon	incoming	routed	pause	notready (unknown)	Work meeting	other work	outgoing	idle	(outgoing & notready)
3-1 13:41:01	good	10		4				3	1	2	
3-1 13:57:43	bad	10	1	4		1		3	1	1	1
3-1 15:13:08	bad	10		3	1	1		4	1		
3-1 16:41:31	bad	8	1	2				3	3		1
4-1 11:35:45	bad	11		4		1		4		2	
4-1 15:45:43	good	11		3	1			3		3	
5-1 16:09:39	bad	8		4	1			2	1		
10-1 13:09:42	bad	10		5	4			1			
13-1 16:49:10	good	7		2	1			1	2	1	
13-1 16:58:02	bad	6		1		2		2		1	
14-1 11:15:20	bad	8		5	1			2	1		1
19-1 9:31:38	bad	9		5				2	1	1	
19-1 11:06:43	good	9		5						4	
20-1 10:55:58	bad	9		1	1		5		1	1	
20-1 13:53:19	bad	9		4	1	1		1	2		
20-1 14:49:20	good	9		3			3	1	1	1	
24-1 12:41:11	bad	9		3	3	1		2			
31-1 10:19:46	bad	8		4		1		2	2	0	1
2-2 16:32:27	good	7		1				1	1		
2-2 16:36:43	bad	7		5					1	1	

It looks like the other work is not done in a preemptive fashion. This could well be one of the most important reasons for not meeting the service level.

6.2 Validation of the simulation and erlang model

The service levels obtained using our simulation and Erlang models have been compared with the real service level for all three types of customers. It appears that in most cases the values obtained by simulation are less than 5% off compared to the real values. These result are pretty good, considered that the call center was quite small. (The smaller a call center, the harder it is to predict it. Large call centers are far more deterministic in their behavior.)

For modeling, some simplifications have been made. The number of agents that was really available has been inputted. The problem is that this number per period (quarter of an hour) is not exactly an integer, while an integer is required for the models. The solution to this problem was rounding. I have also done a test with rounding to above and rounding to below, the results are shown in appendix VI. It appears that the values for the service level are almost always in-between the rounding to below and the rounding above.

Other simplifications include not taking abandonment (1-4%) and redials into account.

In the following table you see a comparison between the two different Markov models ('steady state' and 'uniformization' of chapter 5), the simulation model (chapter 4) and the true values.

datum	real			simulation			randomization			Steady state		
	ext	not	int	Ext	not	int	ext	not	Int	ext	not	int
3-1	72%	72%	70%	71%	64%	67%	79%	74%	75%	78%	71%	72%
4-1	87%	88%	82%	88%	88%	86%	92%	92%	91%	90%	90%	88%
5-1	97%	95%	91%	94%	95%	93%	96%	97%	96%	95%	96%	96%
6-1	87%	81%	74%	86%	84%	83%	90%	88%	88%	88%	86%	86%
7-1	79%	76%	76%	83%	82%	79%	86%	86%	86%	85%	85%	85%
10-1	88%	91%	83%	89%	91%	89%	93%	94%	93%	91%	93%	92%
11-1	75%	87%	72%	83%	84%	82%	87%	88%	87%	86%	87%	86%
12-1	64%	84%	60%	81%	79%	77%	87%	87%	87%	85%	85%	86%
13-1	58%	74%	55%	77%	79%	76%	87%	88%	88%	85%	86%	86%
14-1	55%	76%	65%	76%	74%	74%	84%	83%	87%	81%	81%	86%
17-1	73%	83%	85%	80%	83%	79%	84%	87%	86%	80%	84%	83%
18-1	84%	85%	84%	90%	88%	88%	92%	90%	91%	91%	88%	89%
19-1	65%	82%	80%	81%	83%	84%	84%	86%	87%	81%	84%	84%
20-1	54%	59%	52%	55%	47%	49%	66%	61%	66%	66%	64%	66%
21-1	63%	76%	71%	73%	74%	75%	79%	81%	83%	73%	77%	80%
24-1	46%	64%	58%	69%	71%	70%	76%	78%	80%	72%	75%	77%
25-1	88%	92%	85%	88%	88%	85%	92%	92%	92%	90%	91%	91%
26-1	81%	89%	94%	88%	88%	86%	91%	90%	90%	90%	88%	88%
27-1	66%	68%	54%	72%	70%	67%	84%	81%	83%	81%	78%	80%
28-1	79%	79%	78%	79%	82%	76%	82%	85%	79%	80%	83%	74%
31-1	60%	67%	66%	62%	62%	65%	77%	77%	81%	74%	73%	79%

1-2	69%	75%	72%	75%	74%	77%	82%	82%	84%	76%	77%	82%
2-2	73%	75%	73%	77%	72%	72%	82%	79%	80%	79%	74%	78%

It seems like the model gives good results. For example for the notaries, simulation is on average just 3% off, and the steady Erlang model 4%. The extra computation time for the uniformization model does not seem to pay off, as it usually overestimates the service level by 6%.

The service level for external customers seems a bit high though. One of the reasons is that in the model one agent group was assumed. Further analyses showed that there was one agent that only served notaries and one that did not serve external customers. The problem is that for this setting no historical data was available. This might well be the reason for the low real service level for external customers.

6.3 Some simple what-if scenarios

It appears that the models are quite handy for doing what-if scenarios. For example, what would happen if we would give the notaries absolute priority? If we do that, for example for the first day(3-1), the service levels for the different call types go from 0.708(ext.), 0.641(not.) and 0.674(int.) to 0.708, 0,677 and 0,672. By a simple change in the priority of the notaries, their service level can be approximately 3% higher.

Another example is the very short calls. It would be nice if these calls stopped, but which impact do they have? If the very short calls stop, simulation shows a further increase of the service level to 0.712(ext.), 0.682 (not.) and 0.677 (int.). So the increase for the service level of the notaries would be approximately 4%.

7 SCHEDULING SHIFTS

In the previous chapters, we tackled the problem of estimating the service level and other performance statistics. The scheduled staff (agents) was seen as an input.
Normally however, the problem is the other way around. Certain targets for the service level are given and agents have to be scheduled. This chapter tackles that problem.

7.1 Problem description

The inputs and outputs are more or less the same as with the estimation components. The main difference is that the number of available agents is not an input, but a desired minimum service level is. Next to that, we also need to know which shift is when. The possible shifts are considered to be known.

An overview of all inputs and outputs is given in appendix IV.

Basically we want to minimize the cost of the shifts used, while making sure that the desired service level is met. If two options have the same cost, we want to select the one with the highest service level.

7.1.1 A more formal mathematical formulation

If we define:

- Shifts, the number of possible shifts
- Groups, the number of agent groups
- Periods, the number of periods (quarters)
- Skills, the number of skills (tasks)
- Cost(shift,group), the cost for using a certain shift/agent group combination
- Used(shift,group), the number of used shift/agent group combinations
- MinShift(shift,group), the minimum number of agents to be used per shift/agent combination
- MaxShift(shift,group), the maximum number of agents that can be used per shift/agent combination
- MinSL(skill,period), the minimum desired service level per skill and period
- MinSL(skill), the minimum desired overall service level per skill
- SL(Used,skill,period), a magic function that is able to determine the service level per skill and period. It also uses some of the other inputs. (A performance estimator is used here)
- SLOverall(Used,skill), a magic function that is able to determine the service level per skill and period. It also uses some of the other inputs. (A performance estimator is used here)

The problem we want to solve is:

$$\text{Max} \sum_{shift=0}^{shifts} \sum_{group=0}^{groups} -Cost(shift, group) \cdot Used(shift, group)$$

Under the following conditions:

Used(shift,group)>=MinShift(shift,group)	∀ group ∀ agent
Used(shift,group)<=MaxShift(shift,group)	∀ group ∀ agent
SL(used,skill,period)>=MinSL(skill,period)	∀ skill ∀ period
SL(used,skill)>=MinSL(skill)	∀ skill

Note that this mathematical description skips the fact that we want to maximize the service level as a second objective (in case the cost stay equal).

7.2 Fitness function

As the problem is basically seen as a combinatorial optimization problem, each possible solution should get a value. We decided to incorporate the service level constraints into the objective function. By doing this, there is no special handling required of the constraints, as violating the constraints will lead to a bad value. Incorporating the constraints into the objective function is probably the simplest way of handling constraints. (For other methods, see for example [44].)
Therefore the objective function consists of:

- The costs for all used shifts (resulting in negative number)
- The difference between the call service level per task & period and the required service level (if there is a required service level) multiplied by
    - A large number (1000*the highest costs of a shift), if the required service level is not met (resulting in a negative number)
    - A small number (1/(1000*the highest costs of a shift)), otherwise, to encourage a higher service level (resulting in a positive number)
- The difference between the overall service level per task and the required service level multiplied by
    - A large number (10000*the highest costs of a shift), if the required service level is not met (resulting in a negative number)
    - A small number (1/(500*the highest costs of a shift)), otherwise, to encourage a higher service level (resulting in a positive number)

The fitness of a solution (or value of a solution) is therefore always negative. The better a solution is, the higher its fitness. The large numbers make sure that the constraints will never be violated. The small numbers make sure that a better service level is only seen as a second objective and lower costs are always seen as better. You can choose them bigger, but that will only lead to more chance of round-off error. You might want to choose them lower, in order to carefully weight between the different objectives (see also 7.7.4).

7.3 Other work estimation

The service level for the other work is estimated using the 'useful time left' output of the customer contact center performance estimator (the simulation or Erlang calculation). For each piece of other work, we estimate the service level assuming FIFO (first in, first out). We assume that all the other work must be done the same day, and that a service level can be in terms of period (quarters). For the other work, an allowed waiting time can be specified in time periods (of 15 minutes). If the work is done within the allowed waiting time in periods, the service level is seen as met. If the work is done later, the service level is seen as not met. All the other work must be done the same day.

We simply go through all the useful time left and let the useful time left contribute to doing other work as much as possible. When a contribution is done, we check whether the other work was done in time, and update our service level statistics accordingly.

I will explain the algorithm used by an example, with one type of other work, and 1 period allowed waiting time:

Period	1	2	3	4	5
Useful time (available in period)	30	30	23	40	80
Other work (incoming in period)	90	20	10	30	20

We start at period one. We deduct 30 from 90, and get 60 left other work in period 1. We add 30 to a 'service level met' variable.
Then we go to period two (for the useful time). We deduct 30 from the 60 left. We add 30 to a 'service level met' variable.
Then we go to period three (for the useful time). We deduct 23 from the 30 left. We do *not* add 23 to a 'service level met' variable, as the 23 time-units other work was more than one period (=the allowed waiting time) away.
Then we go to period 4. We deduct 7 from the 7 left. We go to period 2 for the other work. We deduct 20 from the 20 left. We go to period 3, and deduct 10 from the 10 left. We add 10 to a 'service level met' variable. Finally, we deduct the last 3 time-units from the other work of period 4. We add 3 to a 'service level met' variable.
Finally we go to period 5. We deduct 27 from the 27 left of period 4. We add 27 to a 'service level met' variable. We go to period 5 for the other work, deduct 20 from the other work and add 20 to a 'service level met' variable. Now there is no other work left, so we just skip the rest of the useful time.

The total other work was 170, of which 150 was 'service level met'. The estimated service level was 150/170≈88%.

The other work is also put into the fitness function. The following is incorporated into the function value calculation:

- The difference between the overall service level per other work type and the required service level multiplied by
  - A large number (5000*the highest costs of a shift), if the required service level is not met (resulting in a negative value)
  - A small number (1/(10000*the highest costs of a shift)), otherwise, to encourage a higher service level (resulting in a positive value)

- The difference between the amount of 'useful time left' and the amount of other work multiplied by
  - A large number (5000*the highest costs of a shift), if there is too little 'useful time left' (resulting in a negative value)

7.4 Implemented approaches for scheduling

7.4.1 'Covering minimum load'

'Covering minimum load' is a heuristic specially designed for this project. The main idea is that in normal situations, the number of agents should be at least enough to cover the load. The load is determined given the arrival distribution and the service duration. (So load= [expected calls]*[service duration])

This method is used to get a very fast, 'minimum' schedule. Normally, the generated schedule will still have a bad service level. This method does not rely on any estimator and is therefore extremely fast (takes 0 seconds computation time).

The algorithm works as follows:
First, the currently not handled load is calculated. Given the currently not handled load for each period and each call type, the places (period/call type combinations) where the not handled load is bigger than 0 are marked. A shift that has the largest covering of marked periods as possible is then scheduled (taking into account the length of the shift, so the covered period count is divided by the duration of the shift). This process repeats itself, until the entire load is covered.

I will explain this algorithm by an example. Assume 2 skills, 2 agent groups (of 1 skill each), 3 periods and 2 shifts. The first shift covers period 1&2, the second period 2&3. Assume the expected number of calls per period per skill times the service duration per skill gives the following load:

Load	Period 1	Period 2	Period 3
Skill 1	0.2 (bad)	1.3 (bad)	0.9(bad)
Skill 2	2.94 (bad)	1.4 (bad)	0.9(bad)

The first step of our algorithm, we can see that not all load is covered yet. We therefore go over all agent groups and all shifts. It appears the first agent group and the first shift cover a bad situation for all overlapping periods. The fraction of covered periods is therefore 1. We schedule this shift with this agent group. The new situation then is the following:

Load	Period 1	Period 2	Period 3
Skill 1	-0.8 (good)	0.3 (bad)	0.9(bad)
Skill 2	2.94 (bad)	1.4 (bad)	0.9(bad)

We again look over all shifts and find out that we have a covering of 0.5 for agent group 1 and shift 1, while we have a covering of 1 for all other combinations. So we schedule a shift for the second one(group 1, shift 2) and get the following:

Load	Period 1	Period 2	Period 3
Skill 1	-0.8 (good)	-0.7 (good)	-0.1(good)

Skill 2	2.94 (bad)	1.4 (bad)	0.9(bad)

Now all shifts of agent group 1 have a covering of 0. All shifts of agent group 2 have a covering of 1. We keep scheduling until all load is covered and finally get the following used shifts:

Used Shifts	Shift 1	Shift 2
Agent group 1	1	1
Agent group 2	3	1

Note that the combination shift 2/agent group 2 was added, although it only had a covering of 0.5 when it was scheduled.

7.4.2 'Do Highly Potentials'

'Do Highly potentials' is a heuristic specially designed for this project. This method is useful when we know a minimum required service level for each period. If there is no such minimum required service level given, we assume it is 1%, so this method is also useful for that case. The idea is that the periods in which the service level requirement is not met are marked (and all others are not marked). A shift that has the largest covering of marked periods as possible is then scheduled (taking into account the length of the shift, so the covered period count is divided by the duration of the shift). This is done the same way as in 'Covering minimal load'. This process repeats itself until there are no marked periods left (or there are no shifts to schedule left).

I will explain this algorithm in detail with the following example. Assume again that there are 2 skills, 2 agent groups (of 1 skill each), 3 periods and 2 shifts. The first shift covers period 1&2, the second period 2&3. We want to have a service level of at least 0.1 for each period. Assume the service level is as follows, with the current shifts used:

Service level	Period 1	Period 2	Period 3
Skill 1	0.02 (bad)	0.22 (good)	0.09 (bad)
Skill 2	0.53 (good)	0.12 (good)	0.40 (good)

There are 2 combinations with a covering of 0.5, namely agent group 1, shift 1 and agent group 1, shift 2. The others have a covering of 0. Therefore the first combination is scheduled, consequently agent group 1, shift 1 gets scheduled. We ask for an estimation of the service level again, and get:

Service level	Period 1	Period 2	Period 3
Skill 1	0.52 (good)	0.44 (good)	0.33 (good)
Skill 2	0.53 (good)	0.12 (good)	0.40 (good)

So the minimum service level is met for every period, and our algorithm is done.

7.4.3 'Fix bad service levels'

'Fix bad service levels' is also a heuristic specially designed for this project. The idea behind this algorithm is that if the service level is not met, it is a good idea to schedule agents for the occasions where the service level is bad.

The algorithm works as follows:
For the covering of each shift and agent group combination, the (not weighted) average service level of the covered time and skill combinations is determined. Note that a shift covers one or

more periods and that an agent group covers one or more skills. Then the shift and agent group combination with the lowest service level is selected and added to the schedule. This process repeats itself until the service requirements are met (or an unfeasible situation is detected).

I will explain this algorithm in detail with the following example. Assume again that there are 2 skills, 2 agent groups (of 1 skill each), 3 periods and 2 shifts. The first shift covers period 1&2, the second period 2&3. We want to have an average service level of 0.8 for all skills. Assume the service level is as follows, with the shifts used:

Service level	Period 1	Period 2	Period 3
Skill 1	0.52	0.44	0.33
Skill 2	0.53	0.12	0.40

The combination agent group 2 and shift 2 has the lowest average covered service level (namely 0.26). Therefore, this shift is scheduled. We then get the following situation:

Service level	Period 1	Period 2	Period 3
Skill 1	0.52	0.44	0.33
Skill 2	0.53	0.89	0.78

Now the service level covered by agent group 1 and shift 2 is the lowest (0.39). Therefore, this combination is scheduled.

Service level	Period 1	Period 2	Period 3
Skill 1	0.52	0.90	0.97
Skill 2	0.53	0.89	0.78

The average service level, weighted by the number of calls, is now 0.83 for skill 1 and 0.81 for skill 2. The wanted average service level for both skills was 0.8, so the algorithm stops.

7.4.4 Local search

The idea behind local search is to search for a better solution in the neighborhood of the current solution. The neighborhood used is all solutions with one shift added, removed or exchanged. If there are many shift and agent group combinations, this still gives a large neighborhood. Therefore, local search may take a while.

Local search is a very well known algorithm; see for example [12].Our local search is a bit tweaked towards this problem. In case the service level requirements are met, the 'add a shift neighborhood' is not searched. In case the service level requirements are not met, the 'remove a shift neighborhood' is not searched.

In pseudocode, our local search works as follows:

```
Begin:
If the constraints are met
       For each possible solution with one shift less than the current
       one
            If the solution is better, set that solution as the
            current one and go to begin
       Next for each
Else (if the constraints are not met)
```

```
       For each possible solution with one shift extra than the current
       one
            If the solution is better, set that solution as the
            current one and go to begin
      Next for each
End if
For each possible solution with one shift less and another shift extra
than the current one
       If the solution is better, set that solution as the current one
       and go to begin
Next for each
Done, there are no are no better solutions in the neighborhood
```

Local search always finds a local optimum, but it might be that there is a better, global optimum.

I will explain this algorithm in detail with the following example. Assume again that there are 2 agent groups (of 1 skill each) and 2 shifts. We want to have an average service level of 0.8 for all skills. Assume we have the following solution, with value –87.99943:

Used Shifts	Shift 1	Shift 2
Agent group 1	2	2
Agent group 2	3	2

As the service level constraints are currently met, we try to remove one of the shifts used (for all 4 possible combinations). If we do this, the constraints are always violated. This does not give a better solution.
We then try to remove one of the 4 combinations, while adding another one. We consequently try 4*3 other solutions. If we remove combination agent group1, shift group 2 and add combination agent group 1, shift group 2, we get a solution value of –87.99947. This is slightly better, so we take the new solution:

Used Shifts	Shift 1	Shift 2
Agent group 1	3	1
Agent group 2	3	2

As the service level constraints are currently met, we again try to remove one of the shifts used (for all 4 possible combinations). If we do this, the constraints are always violated. This does not give a better solution.
We then try to remove one of the 4 combinations, while adding another one. We consequently try 4*3 other solutions. This however also does not give a better solution. We are finished and have found a global optimum, with solution value –87.99947.

7.4.5 Greedy search

The idea behind greedy search is the same as with local search. The only difference is that local search goes to the first neighbor that is better, while greedy search searches all neighbors and goes to the *best* neighbor. Greedy search is a well-known algorithm; see for example [45] for a description.

Our special greedy search only searches the 'add a shift neighborhood' (and therefore does not remove or exchanges shifts).

In pseudocode, our greedy search works as follows:

```
While the constraints are not met yet
       Set best_solution_seen to Nothing
       For each possible solution with one shift extra than the current
       one
            If the solution is better than best_solution_seen, set
            that solution as the new best_solution_seen
       Next for each
       Set the current solution to be the best_solution_seen
Loop (but if there is no improvement for 5 subsequent steps, stop)
```

The improvement check prevents an infinite loop, for example for the situation where there is no way you can meet the service level.

I will explain this algorithm in detail with the following example. Assume again that there are 2 agent groups (of 1 skill each) and 2 shifts. We want to have an average service level of 0.8 for all skills. Assume we have the following solution, with value –789932.32:

Used Shifts	Shift 1	Shift 2
Agent group 1	1	1
Agent group 2	3	2

If we add combination agent group 1, shift 1, we get a value of –483384.43. If we add combination agent group 1, shift 2, we get a value of –483484.43. If we add combination agent group 2, shift 1, we get a value of –689889.87. If we add combination agent group 2, shift 2, we get a value of –578732.32. We schedule the best one, and get the following solution with value –483384.43:

Used Shifts	Shift 1	Shift 2
Agent group 1	2	1
Agent group 2	3	2

Again, the service level is not met, so we search for a better shift. If we add combination agent group 1, shift 1, we get a value of –87.99947. If we add combination agent group 1, shift 2, we get a value of –87.99943. If we add combination agent group 2, shift 1, we get a value of –444315.32. If we add combination agent group 2, shift 2, we get a value of –443236.82. We schedule the best one, and get the following solution with value –87.99947:

Used Shifts	Shift 1	Shift 2
Agent group 1	3	1
Agent group 2	3	2

The constraints are met, so we stop with greedy search.

7.4.6 Evolutionary Algorithms

Evolutionary algorithms are inspired by the well-known evolution theory of Darwin. The idea is that you have a population of solutions that you want to make better.

In pseudocode, a evolutionary algorithm works as follows:

```
INITIALIZE population with random candidate solutions
EVALUATE each candidate
Repeat until (TERMINATION CONDITION) is satisfied
      SELECT parents
      RECOMBINE pairs of parents
      MUTATE the resulting offspring
      EVALUATE new candidates
      SELECT individuals for new population
Loop
```

Evolutionary algorithms are an entire field of research, see [44] for more information on them. There are many different evolutionary algorithms possible.

In our implementation, an individual in the population (solution) is simply a vector with some numbers in it. These numbers are the number of agents that are going to do a certain shift.

For example, if we have 2 agent groups and 2 shifts, a solution is a vector with 4 numbers, like:

	Variable 1	Variable 2	Variable 3	Variable 4
Value	3	1	3	2

Each solution has a value (fitness), for example –87.99947. This is exactly the same solution as the one of the table above.

The process of creating a better population goes as follows (in our implementation):
We have a population, generated initially at random, with [population size, for example 60] solutions. We then select the best [selection size, for example 10] solutions, and throw away all others.
From these 10 solutions, we generate 59 new solutions. We do this as follows. We recombine two random solutions from the pool of 10 to get a new individual. We do the recombination by averaging all the values of the two solutions. The new solutions are then mutated by:

- Selecting a few random values and subtracting one to them
- Selecting a few random values and adding one to them
- Selecting a few times two positions and exchanging one agent

We have defined 'a few' as a random integer in the range [0,the number of variables/3+1].
We only keep the best individual from the 10 original solutions. So now we have a new, and hopefully better population of 60 solutions.

The number of generations can be given, so the number of evaluations used is known in advance. The implementation is currently very simple and does not use any problem specific information. The same implementation can be used for other problems that other Anago component might have as well.

Parameter control
One of the most straightforward extensions will be the implementation of parameter control. With parameter control, as needed more or less mutation can take place, depending on whether the obtained results for such a mutation have been good.

In our case, because the constraints are put into the objective function, at first we will have solutions that have much to many agents. Initially, many subtractions will be good. Later in the process however, only a few subtractions should be done.

Due to priorities I have not yet implemented parameter control (EA probably will take too much runtime in practice).

7.4.7 Simulated Annealing

Simulated annealing is quite similar to local search. It is a very well known algorithm and is described in the literature (for example in [30]). Simulated annealing can be seen as an evolutionary algorithm [44], but it is generally seen as another class of algorithms.
The main difference between local search is that neighbors are inspected in a random fashion and that sometimes worse neighbor solutions are accepted. We want to find a global optimum instead of a local optimum. By accepting worse solutions, we are hoping that we leave a local optimum so we can head for a global one. Over the time, the chance of accepting a worse solution goes down to zero. The chance is also related to the difference in the value of the new solution. The chance of picking a slightly worse solution is higher than the chance of picking a much worse solution.
In fact a solution is accepted if exp((new_value-last_value)/current_temperature)>U with U a random double between 0 and 1. current_temperature starts at a given temperature and (in our implementation) drops to 0, linearly with the number of evaluations done. The current linear dropping of the temperature, called the cooling scheme, might well be not the best one.
As the last solution seen certainly does not have to be the best solution seen, the best solution is always kept.

7.5 Tests, motivations and results

To establish a good approach for the scheduling, I have done a lot of tests. In this subchapter, some results are given. All results are from an old Pentium II 600Mhz.

7.5.1 A simple test case

I first used a very simple test case. I used 4 quarters of an hour with some simple parameters, and shifts of one quarter each. I then tried to schedule it using Greedy Search. At first, no result was made, because the service level for the last period remained 0 when adding an agent to the last period. Therefore, an initial minimum service level of 0.01 should have been reached. When we started with the minimum number of agents to reach that service level (using 'Do Highly Potentials'), we got an objective cost value of 50.997902 using 13 seconds computation time. With local search applied after that, we obtained a cost value of 49.998254 using a total computation time of 17 seconds. Using local search directly after 'Do Highly Potentials' gives the same results but takes 29 seconds to reach the same value, making greedy search look worthwhile.

However, if we use 'Fix bad service levels' instead of greedy search, we get the same result in just 14 seconds.

'Fix bad service levels' still requires one value evaluation per shift that needs to be added. This inspired the creation of 'Covering minimum load', which requires no evaluations at all. Almost all the required time comes from the evaluations, therefore 'Covering minimum load' takes about 0 seconds computation time. This reduced the amount of time needed to obtain the best value to just 9 seconds.

If we run simulated annealing right after 'Covering minimum load', with 1000 evaluations and starting temperature 5, we get (not surprisingly) a value of 49.998254 too. Simulated annealing uses 110 seconds for 1000 evaluations. I implemented it just to prove that local search does well.

Evolutionary computing, with 120 generations of 60 individuals and a selections size of 10, also gives a value 49.998254, but takes 14 min. to calculate. For this example test case, local search probably finds the global optimum, while taking just 9 seconds.

7.5.2 A more typical test case

I used some data from our case study (chapter 6) again and divided the inter-arrivals times by a factor 5. Therefore, it is basically the same call center with 5 times as many customers. I have done multiple tests with multiple days, but for this example I will use the data from the last day. I used three possible shifts, with the shifts covering the whole day (33 quarters), except for a two-quarter lunchtime. The lunchtimes were non-overlapping (quarter 12-13,14-15 and 16-17).

It takes using 'covering minimal load' for 80% of the load and 'fix bad service levels' 6 seconds to go to a solution with cost -619.999556. Local search after that step takes about one minute, to find a slightly better solution with cost -619.999499. Simulated annealing (using 1000 evaluations, and starting temperature 5) is not able to improve on that solution, taking 17 minutes. In all my test cases, simulated annealing and evolutionary computing give the same result as local search. This gives an indication that with three possible shifts, local search might well be optimal.

7.5.3 Scheduling our case without shifts (a harder test)

I simply used the data from the last day of our case study again, but now without shifts. This means there are 33 variables (one for each period). Local search in this case can take up to three hours, mainly because there are 33*32 possibilities to exchange a shift. Even without a five times busier call center local search takes about one hour. In this case, local search is not optimal anymore. Simulated annealing and evolutionary computing are able to come up with better results.

For example, for the original call center we found the following. Without local search it takes 8 seconds to reach a cost of 136.997508. With local search, it takes one hour to reach 129.997729. After a few hours of simulated annealing, I got 129.997720.

For the problem with 5 times as many arrivals, for a typical case local search gives a cost value of 493.988345. With hours of simulated annealing, I am able to get 487.988587. Without the 'shift exchanges' of local search, I get 503.988467 in 1.5 min. So with hours of work, it is possible to get a slightly better solution. For normal situations, I would not recommend that.

7.5.4 A test with more variable shifts

I also extended the case of 7.5.2 with three extra shifts, to have a test with multiple shift lengths. The shifts used were, including the original three (gray=happening at that quarter):

	Quarter of an hour																															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Shift 1	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Shift 2	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Shift 3	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Shift 4															▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Shift 5										▨	▨	▨	▨	▨	▨	▨	▨	▨	▨													
Shift 6	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨																		

I then used our final algorithm (see 7.6) to find a good solution. The results are given in the following table. As it is interesting to see the solution in this case, the solution is shown also.

Model	Method (after)	Solution value	Shifts used						Total time taken till this point (sec)
			1	2	3	4	5	6	
Steady erlang	Covering minimal load, fix bad SL	-597.999543	8	3	1	7	4	4	1
	Local search	-577.999602	2	5	5	7	2	4	3
Simulation	Covering minimal load, fix bad SL	-613.999541	2	5	5	9	2	4	6
	Local search	-603.999592	1	5	6	9	1	4	71
	Simulated annealing (1000 evals, temperature 5)	-603.999592	1	5	6	9	1	4	1040
	Simulated annealing (2000 evals, temperature 20)	-598.999578	5	9	3	4	0	0	1988
	Simulated annealing (4000 evals, temperature 40)	-597.999605	0	7	8	6	1	1	3788
	Simulated annealing (10000 evals, temperature 50)	-595.999603	2	6	8	5	1	0	18728

I was not able to improve upon the last solution; however there might of course be a better solution. Simulated annealing was run each time after local search, not after the previous simulated annealing.

It seems that the best solution found is only slightly better than the solution found much earlier in 6 or 71 sec. It is interesting that local search is not optimal. It is also interesting that multiple solutions with a very different arrangement of the shifts are all quite good. Another interesting thing is that the new shifts are chosen by our heuristics, but that they do not all seem to be incredibly useful if we look at the simulated annealing solutions. This probably is because there are fewer calls during lunchtime, while the new shifts are all during lunchtime. There also seem to be fewer calls in the morning.

7.5.5 Other work

The other work scheduling has not yet been extensively tested. As it is generally simpler to schedule if other work is involved, I just tried to add some other work to the previous test cases. The scheduling seemed to go fine. One of the interesting things that I found is the behavior of other work in combination with small shifts and abandonment. It seems that the algorithm makes sure that the abandonment is high in some periods, in order to get as much time useful time left as possible… This may be accepted behavior in the model, but it might not be wanted behavior in real life.

7.6 Final implementation

After some tests, it appears that a good solution can be found quickly if 'covering minimal load', 'Do Highly Potentials' (which can just do nothing most of the times) and 'fix bad service levels' are applied after each other. Finally, local search can be applied to find (hopefully) a global optimum.

Because simulation can take some time, the scheduling is first done using the steady Erlang method. Then the performance estimator is swapped for a simulation one and 'Do Highly Potentials,' 'fix bad service levels' and local search are executed again.

This solution can be optimized further to a global optimum using simulated annealing and/or evolutionary programming. Their use depends on the parameters given to the component.

In general, the best approach heavily depends on the number of shifts and agent groups. For an implementation of the scheduling component in a real call center, it might well be that some tests and tweaks will be necessary.

7.7 Possible further research on the scheduling

The scheduling is a subject that can be researched much further (for many years). In this paragraph some suggestions for further research are given.

7.7.1 More testing and tweaking

As the current testing on real test cases has been quite limited, it is probably a good idea to do some testing on some real test cases. It should not be too hard to come up with a combination of the currently implemented scheduling methods that gives good results fast.

7.7.2 Model extensions

In the current model, doing other work scheduling while modeling abandonment and having small shifts leads to a situation where abandonment is kept high in some periods, because more other work can be done in that situation. A possible extension might be to put the abandonment in the model and for example minimize the abandonment.
In addition, you might want to minimize things like blocking, etc.

7.7.3 New scheduling methods (examples)

Changing 'fix bad service levels'
Currently, this heuristic uses a normal, not weighted average. It might be a good idea to try a weighted average instead. I however expect that the results will be quite similar.

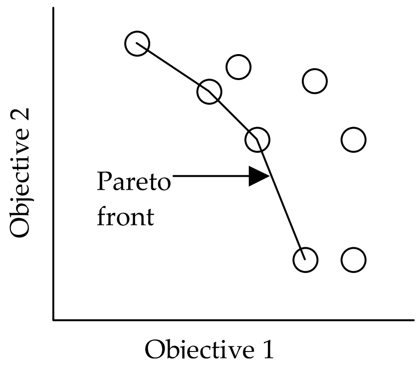
Combined scheduling methods (hybridization)
In the world of combinatorial optimization, good results are seen when combining multiple algorithms. For example, it could be a good idea to use the genetic algorithm to explore many options, while using some iterations of local search for each member of the population of each generation to determine a better solution. Local search gives a good solution in the neighborhood, while with a genetic search we hope to explore the global solution space. For example in 7.5.4 you can see that quite different solutions might all give good values. Instead of genetic search, you could also use some algorithm to generate a very different shift arrangement with the same costs, to explore the global solution space.

Tabu search
Tabu search is another general combinatorial optimization technique that has not been used yet. The idea is basically the same as with local search, but when a local optimum is found, the scheduling does not stop. Instead the scheduling might continue, by disallowing points previously visited (hence 'tabu'). You can find more information on tabu search (and on the other methods) using Google (http://google.com/search?q=tabu%20search) or the Google Scholar (http://google.com/scholar?q=tabu%20search).

Linear programming or branch-and-bound search
There are some interesting articles on the use of linear programming and/or branch-and-bound methods for this scheduling problem. To the best of my knowledge, there is not an article yet that solves the problem using shifts and multiple skills. (There are articles and tools with shifts, but without multiple skills [8][4] and there are articles without shifts but with multiple skills [9]). In the future, they might well be found using the Google Scholar for example (http://google.com/scholar?q=call+center+simulation+shift+scheduling) or it might be an interesting research subject.

7.7.4 Offering multiple solutions

As seen in for example 7.5.4, it might well be that multiple solutions perform almost equally well, while the shifts arrangements are quite different. A challenge might be to offer multiple good solutions, based on different objectives. (A keyword in this area of research is 'Pareto front', the solutions that are the best based on multiple objectives)



8 INTEGRATION IN ANAGO & IMPLEMENTATION

8.1 Position of the components within Anago

Anago software has a design program, called Anago Studio, where you can develop a new Anago application. An Anago application consists, for the calculation part, of models that consist of engines with certain properties and variables set. These applications are run in Anago, the run-time environment.

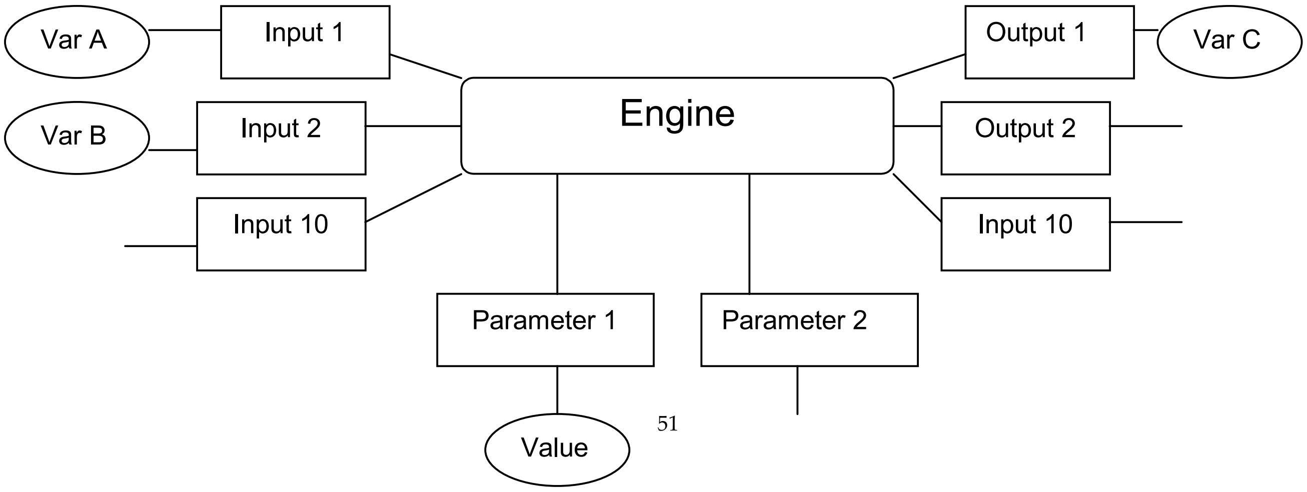
Our estimation and our scheduling are seen as two different engines that can be used in an Anago application.
At design time, the designer must be able to set certain properties and variables. The engine must somehow supply the properties and variables and must check (if possible) if they are set correctly.
At run time, the engine somehow has to be executed with the right variables and parameters by the application.

Our implementation at work in Anago is visible in appendix VII.

8.2 Interface for engines

To facilitate the above behavior there has to be an interface between the Anago (Studio) software and an engine. Because currently almost all engines were situated in a database environment instead of a VB.NET environment, a new interface had to be made. This interface should be able to handle both design and runtime situations. It was chosen that all Engines should derive from a .NET base class called 'Engine'. This Engine should not have any global variables that are used in-between calls to the public procedures and functions of 'Engine'. All information is therefore kept by Anago and not by the Engine. This design makes sure that engines are useable even if Anago releases a new version, the engine is used over a network, etc.

Seen from Anago an engine has some ports for inputs and outputs, to which variables can be attached. These variables do not have to be attached to it, because they can be optional or just not filled in yet at design time. An engine also has some parameters that can have values, but values can be optional too. Parameters can have default values.



However it is unexpected, an engine can also have other things that have not been defined yet, but that are not inputs, outputs or parameters. In case they have to be added later, inputs, output and variables have been grouped into one object. (As an example of what they might be, consider 'submodels' or 'functions')

Appendix III shows a class diagram of the situation of an Engine in Anago.

8.3 Mapping variables and checking dimensions

The customer contact center performance estimation and the customer contact center scheduling component do not derive directly from Engine. Instead they have the same super class CCCEngine that derives from Engine. CCCEngine contains general function used by both engines. These functions handle the mapping of the variables that it gets from Anago (a list of ports that contain variables) to the variables used internally. They also handle the checking of the dimensions. The variable and dimension specifications used can be found in appendix IV.

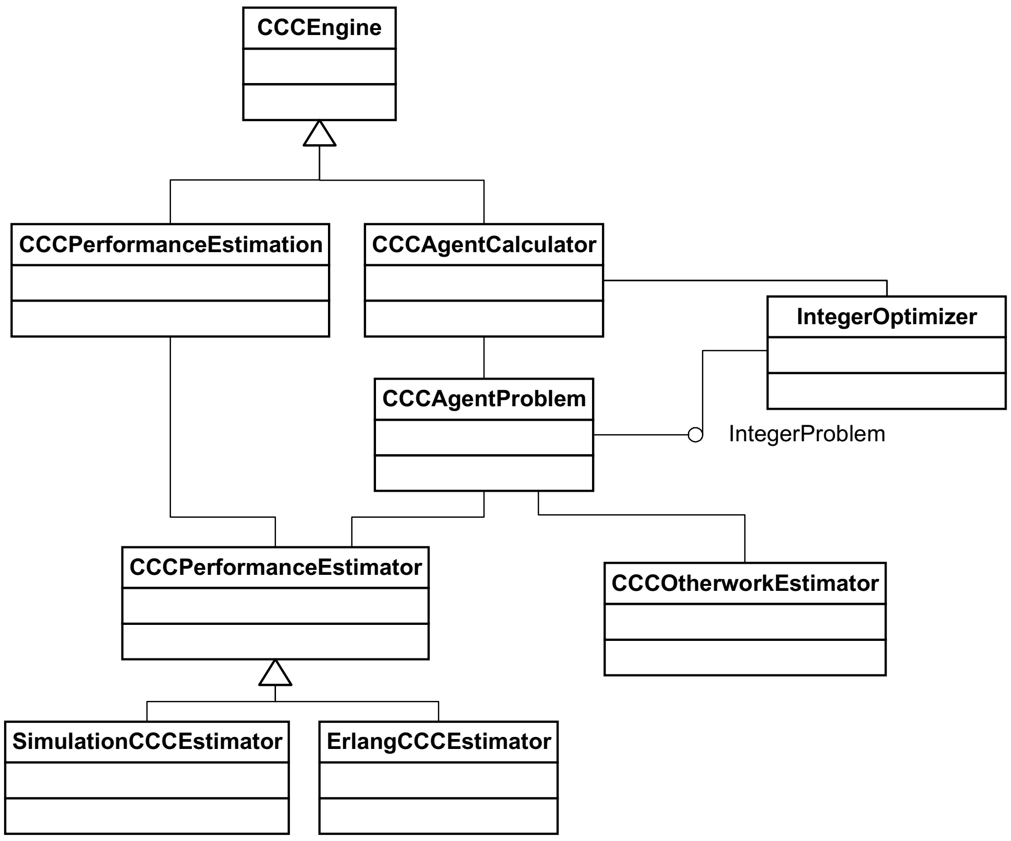
While the variable mapping is basically a simple select case state statement, the technique for checking the dimensions is perhaps worth noting. One of the problems involved is that the order of dimensions that a variable has should not matter. So a variable with 'agent group' as dimension 1 and 'call type' as dimension 2 should behave the same as a variable with 'call type' as dimension 1 and 'agent group' as dimension 2.

A special class 'InvestigatedDimension' is used for each dimension. It detects the possibilities still left for a dimension. It starts with 'anything is possible', and then shifts through all variables. If the number of possibilities goes down to zero, an error is detected. In that case, something was wrong with the input variables, and that combination of variables is not possible and should not be settable. (So CanSetInput or CanSetOutput return false.) Finally, elimination is used, because for example the agent dimension name should not be equal to the call type dimension name. If after elimination two dimension names are the same, an error is detected.

This method for detecting and testing dimensions can be used for other future engines too. In the current Anago implementation, this code makes sure that only valid choices for the variables are presented to the user.

8.4 The rest of our implementation

In the following diagram you can see a class diagram of our implementation (except for the Simulation Framework, see V).



Seen from Anago there are two Engines: CCCPerformanceEstimator (the performance estimator) and CCCAgentCalculator (the scheduler). These two have the same base class CCCEngine, which is explained in 8.3. The CCCPerformanceEstimator Engine is basically an interface to a CCCPerformanceEstimator, which currently can be our simulation or Erlang based model.

The CCCAgentCalculator, our scheduler, features some methods to optimize a given problem. CCCAgentProblem stores the information for a problem, including its evaluation. To evaluate its fitness, CCCAgentProblem uses a CCCPerformanceEstimater and a CCCOtherworkEstimater. CCCOtherworkEstimater tries to estimate the service level for the other work. CCCAgentCalculator also uses some generic search methods that are stored in IntegerOptimizer. To support generic problems, this optimizer solves an IntegerProblem instead of solving CCCAgentProblem directly. CCCAgentProblem therefore implements the IntegerProblem interface.

9 CONCLUSIONS AND DISCUSSION

9.1 Conclusions

We successfully created a piece of software to facilitate work force management in customer contact centers. Basically, we have created two components, one for performance analyses and one for the scheduling of shifts. The components have been successfully integrated into Anago, a software package for management calculations. In Anago, these components are called engines. Some screenshots of the usage of our engines in Anago can be seen in appendix VII.

The performance analyses are done using simulation or Erlang models. Simulation and Erlang models are also found in professional, existing software for this purpose. Verification and validation shows that the simulation and Erlang models are able to adequately analyze a customer contact center. The simulation framework is reusable, so that future simulations can be built upon it. In our case study, the uniformized Erlang model did not seem to be useful. Therefore, simulation is the preferred method. The stationary Erlang model is preferred if very fast calculations are needed.

The scheduling is done using various specially designed heuristics and general methods, like local search, simulated annealing and evolutionary algorithms. Some of these methods should be reusable in future projects involving scheduling. Tests show that the scheduling works correctly and gives good results in not too much time. If you are willing to spend a lot of computation time, it is even possible to get results that look optimal.

To encourage reusability and scalability, all the code has been documented properly. An example of reusable code is the code for testing whether inputs and outputs have the correct dimensions.

9.2 Remarks and improvement suggestions

Since there is always room for improvement, some suggestions are given here:

9.2.1 Better integration of the scheduling in Anago

Currently, the scheduling component is seen in Anago as a new model. For Anago, it is unknown that this model uses our performance estimation component. A nicer option would be that a more general scheduling algorithm solves a general model in Anago. This would mean that the model that needs to be optimized is alterable at design time in Anago, instead of at compile time of the engine. For example, this would allow someone at Anago design level to change the evaluation of solutions in such a way that service level constraints can be violated slightly. Or in such a way that new, complicated constraints are met.

9.2.2 Unknown arrival distribution

In our model, we assume that the arrivals are exponentially distributed. In reality, it might well be that the parameters of the exponential distribution cannot be known. It might be that we should first draw the business for the day, and simulate that day afterwards with multiple levels of business. The unpredictability of the arrival rate and the statement that the arrivals are not really Poisson is explained in [5]. Although it is easy to implement this, it leads to extra complexity. One of the main problems of this approach would be the prediction of the extra inputs.

9.2.3 Better Handling of shrinking (sickness, going to the toilet, etc.)

Currently, the performance estimation takes the number of available agents as an input. This number is an integer per period. This means that you cannot have 'half an agent', for example because a certain fraction of the agents is expected to be smoking, sitting on the toilet, etc. An improvement would be that this shrinking of agents is modeled some way. This however means that the model gets more complex. It also has the problem that there might be a relation between the business and the leaving of agents to the toilet, which needs to be modeled. For the sake of simplicity, this was not done yet. Currently, sickness might be accounted for by raising the agents needed per shift by a certain factor.

9.2.4 Better scheduling

The scheduling problem is an open-end problem. There probably is always room for new and better scheduling algorithms. Some ideas left are for example algorithms based on tabu search, branch & bound or integer programming.
Currently, the scheduling might require some tweaking when it is actually used and some extra testing on real test cases might be worth it.
For more information on further research on scheduling, see 7.7.

9.2.5 Support for different multi-skill routing protocols

Currently, basically one type of multi-skill routing is implemented. When a customer arrives, he is routed to the first suitable and available agent group. Another approach could be that agent groups have some priorities for different types of calls. Also, when an agent becomes free and there is a queue, the first-in first-out (FIFO) policy is used. It could also be that a different policy is wanted. It should be easy to support different routing priorities, specifically by just changing the functions 'GetFreeAgentGroup' and 'GetNextCustomer' in the implementation.

10 REFERENCES

[1] Ger Koole & Marco Bijvank, Erlang-X Calculator, http://www.math.vu.nl/~koole/ccmath/ErlangX.php

[2] Ger Koole & Marco Bijvank, Erlang-C Calculator met blending, http://www.math.vu.nl/~koole/ccmath/blending.php

[3] Auke Pot, SA-SIM, simulation tool for multi-skill call centers with skill-based call routing, http://www.math.vu.nl/~sapot/software/sa-sim/

[4] Auke Pot, Automatic shift scheduling tool for call centers and contact centers, http://www.math.vu.nl/~sapot/software/shift-scheduling/

[5] Noah Gans, Ger Koole, and Avishai Mandelbaum, Telephone call centers: Tutorial, review, and research prospects, Manufacturing and Service Operations Management 5:79-141, 2003

[6] Ger Koole, Call Center Mathematics - A scientific method for understanding and improving contact centers, Version of 23rd August 2004, book in progress.

[7] Ward Whitt, Engineering Solution of a Basic Call-Center Model, Management Science, to appear, 2004

[8] A. Ingolfsson, E. Cabral, and Xudong Wu, Combining Integer Programming and the Randomization Method to Schedule Employees, last revised October, 2003. Submitted (to Manufacturing & Service Operations Management).

[9] Julius Atlason, Marina Epelman and Shane Henderson, Optimizing call center staffing using simulation and analytic center cutting plane methods, submitted

[10] Raik Stolletz, Performance Analysis and Optimization of Inbound Call Centers, Lecture Notes in Economics and Mathematical Systems No. 528, Berlin et al. (Springer-Verlag) 2003 (Dissertation)

[11] A. Ingolfsson, M. A. Haque, and A. Umnikov, Accounting for Time-Varying Queueing Effects in Tour Scheduling, European Journal of Operational Research 139(3) 585–597, 2002

[12] Ger Koole and Erik van der Sluis, Optimal shift scheduling with a global service level constraint, IIE Transactions 35:1049-1055, 2003

[13] G. M. Thompson, Labor staffing and scheduling models for controlling service levels, Naval Research Logistics, vol. 44, 1997(8), 719-740

[14] G.M. Thompson. Assigning telephone operators to shifts at New Brunswick Telephone Company, Interfaces, 27(4):1-11, 1997

[15] Ward Whitt and Rodney B. Wallace, A Staffing Algorithm for Call Centers with Skill-Based Routing, working paper, 2004, Submitted to Manufacturing and Service operations Management

[16] Ward Whitt, Fluid Models for Many-Server Queues with Abandonments, Operations Research, 2005, to appear

[17] A. Ingolfsson, E. Akhmetshina, S. Budge, Y. Li and X. Wu, A Survey and Experimental Comparison of Service Level Approximation Methods For Non-Stationary M/M/s Queueing Systems, last revised June 2003. Resubmitted to INFORMS Journal on Computing.

[18] Vijay Mehrotra and Jason Fama, Call Center Simulation Modeling: Methods, Challenges, and Opportunities, Proceedings of the 2003 Winter Simulation Conference

[19] Philippe Chevalier, Robert Shumsky and Nathalie Tabordon, Routing and Staffing in Large Call Centers with Specialized and Fully Flexible Servers, Working paper, Latest revision: March, 2004.

[20] A. Fukunaga, E. Hamilton, D. Andre, O. Matan, J. Fama and I. Nourbakhsh, Staff Scheduling for Inbound Call Centers and Contact Centers, AI Magazine, 23(4), Winter 2002 (or In Proceedings, IAAI 2002)

[21] Vendors – Work force management, http://www.call-center.net/vendors-workforce-management.htm

[22] Workforce-management in Nederland, deel-1, Uw agent voorspelt of voorspeld?, http://www.telecommerce.nl/sf/sf.cgi?3675

[23] Geert Jan Franx, Ger Koole & Auke Pot, Approximating multi-skill blocking systems by HyperExponential Decomposition, Submitted 2004

[24] Blue Pumpkin wins patent lawsuit, http://www.destinationcrm.com/articles/default.asp?ArticleID=3560

[25] Multiskill, Multichannel Workforce Management - Is It Right for Your Contact Center?, Aspect

[26] Andre David and Illah Nourbakhsh, Agent scheduling system and method having improved post-processing step, Blue Pumpkin Software Inc., US patent 6,278,978

[27] Gary Crockett and Paul Leamon, Skills-based scheduling for telephone call centers, IEX Corporation, US patent 6,044,355

[28] Andre David, Serdar Uckun and Illah Nourbakhsh, System and method for skill based scheduling in a call center, Blue Pumpkin Software Inc., EU patent EP1248449 (US patent 20020143597)

[29] Alex Fukunaga, Scott Veach, Jason Fama, Edward Hamilton, Ofer Matan And Illah Nourbakhsh, Method and apparatus for skill based scheduling in a call centre, Blue Pumpkin Software Inc., EU patent EP1248448

[30] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes in C++, The Art of Scientific Computing, 2nd edition, ISBN 0521750334, Cambridge University Press

[31] George Marsaglia and Wai Wan Tsang, A simple method for generating gamma variables, ACM Transactions on Mathematical Software (TOMS) archive, Volume 26, Issue 3 (September 2000), pp 363 – 372

[32] Marsaglia, George and Tsang, Wai Wan, The ziggurat method for generating random variables, Journal of Statistical Software, volume 5, number 8, pages 1-7, 2000

[33] Marsaglia, George, Random numbers for C: End, at last?, Posting to sci.stat.math, 21 jan 1999

[34] P.H.W. Leong, G. Zhang, D. Lee, W. Luk and J.D. Villasenor, A comment on the implementation of the Ziggurat method, draft 2, submitted to Journal of Statistical Software, Nov 2004

[35] Andrea Emilio Rizzoli, IDSIA, Galleria 2, CH - 6928 Manno, Switzerland, A Collection of Modelling and Simulation Resources on the Internet, Last update November 2004, http://www.idsia.ch/~andrea/simtools.html#libraries

[36] Mathtools.net C,C++ Simulation, http://www.mathtools.net/C_C__/Simulation/

[37] HighMAST object-oriented simulation library, http://www.highpointsoftware.com/
[38] SWAN (Simulator Without A Name), National University of Singapore, http://137.132.165.173/swan/main.asp
[39] Douglas W. Jones, An empirical comparison of priority-queue and event-set implementations, Communications of the ACM, v.29 n.4, p.300-311, April 1986
[40] Robert Rönngren and Rassul Ayani, A Comparative Study of Parallel and Sequential Priority Queue Algorithms, ACM Transactions on Modeling and Computer Simulation (TOMACS), v.7 n.2, p.157-209, April 1997
[41] Wai Teng Tang, Rick Siow Mong Goh, and Ian Li-Jin Thng, Ladder Queue: An O(1) Priority Queue Structure for Large-scale Discrete Event Simulation
[42] Microsoft Corporation, .NET Framework General Reference, Design Guidelines for Class Library Developers, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconNETFrameworkDesignGuidelines.asp
[43] Mathew Reynold, .NET 247 : System.Random Class [Rotor, http://www.dotnet247.com/247reference/System/Random/__rotor
[44] A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer, 2003, ISBN 3-540-40184-9
[45] Fallahead.org, Greedy Search, http://halfninja.nickhowes.co.uk/learn/?id=Greedy+Search

I CALLCENTER WFM - WHAT HAVE OTHERS BEEN DOING?

Of course, it is always a good idea to embrace good ideas of others. This document describes some methods seen in this field. The problem we look at is day-to-day scheduling in a callcenter/customer contact center environment. Typically the following steps are involved in this process (see [8][9] for example):

1. Forecasting demand (Typically the expected arrivals per minute for each short period, where short periods are between 15 minutes and 1 hour long.)
2. Convert demand forecasts to staffing requirements (A number of agents needed for each short period.)
3. Schedule shifts (Staff shifts are selected that cover the requirements of step 2)
4. Rostering/Assigning employees to shifts

We are developing a component for step 2 & 3, so the forecasting and the rostering are not really investigated further.

Traditionally these steps are approached independently. This may however lead to a global suboptimal solution, because the individual steps have different goals[10]. Therefore, sometimes one or more steps of 2-4 are combined. As we want a good solution, and perhaps an individual rostering and evaluator component that might be reused more easily, we especially take a look at this combined approach.

I.1 Scientific articles

Callcenters are an active research field. A recent survey on this subject is [5]. It also contains a description of a call center/customer contact center.
I looked into the literature for suggestions on a scheduling and an evaluator component.

I.1.1 Performance/Service level (SL) estimation methods found in the literature:

- Erlang-C with extensions (see [6] for a detailed explanation 'without maths' or [7])
- Fluid approximations (assuming heavy load) (see [16] for an example)
- Simulation (see [18] for an example)
- Forward differential equations of a M(t)/M/s(t) queue as evaluator (see [11] for an example)

Simulation is probably the most accurate as it is able to match reality best, but needs much computation time. Erlang-C with extensions is the most common.

I.1.2 Period-linking

Traditionally, the periods (quarters) over a day are seen independently. This may lead to a bad SL approximation however. So if it is too busy in period 3, then the SL at period 4 will be less then expected without linking. For the linking of periods with performance estimation with Erlang-C extensions, the following methods are found (see [8]):

- No linking, the standard approach (SIPP, Stationary Independent Period by Period)
- The lag-max approach, which basically shifts the required agents one period ahead if they are higher

- Uniformization (or ‘randomization’), which uses the state probabilities with a Markov chain

Linking with uniformization takes computation time. Using no linking or the lag-max approach however, may lead to worse results in reality than expected.

There is some free software on http://www.math.vu.nl/obp/callcenters/ with extended Erlang models, which could be used for validation/verification.
A recent survey on queue approximations is [17], which, without considering fluid approximations, pleads for the uniformization method.

I.1.3 All methods that assume a stationary situation might have a serious problem (see [8]). Therefore, transient (time-dependent) solutions are to be preferred, for example uniformization, simulation, solving forward Kolmogorov equations, etc. Only simulation can deal with all types of service durations. Fluid approximations only work correctly in a situation with a huge overload (and long queues), which is *not* the situation with a good service level that you want to achieve.Parameters (abandonments, service time distribution)

The duration of the service time is often not truly exponential. Most of the times, though, only average data is available. Therefore, an exponential service time is normally assumed.
The abandonment distribution is really hard to estimate (because of the lack of data available) and depends per call center [18]. It seems however that the abandonment distribution does matter, as it may well give a change in required agents of 5% (from 99 to 104) [7].

I.1.4 Scheduling

For the scheduling component, the following implementations were found:

- Genetic algorithms (with a numerical solution to the forward differential equations of a M(t)/M/s(t) queue as evaluator) [11]
  The algorithm used was a simple genetic algorithm, and the constraints were put in the fitness function. Better results than with the traditional approach were found, but the computation took hours.
- Local Search (with Erlang as evaluator) for a global Service Level constraint [12]
  Under some conditions, local search leads to an optimal result.
- Simulated Annealing [13]
  With an Erlang (M/M/s) model with some linking for evaluating schedules. Simulated annealing was faster then Integer Programming and therefore used.
- Integer programming with a Simulation method as evaluator [9]
- Integer programming with an uniformization method as evaluator [8]
  The idea to the Integer Programming approach is that the model is simplified to an Integer Programming problem, and then the evaluator is run. As long as the evaluator gives a too low SL, constraints are added and the problem is solved again. The IP method is traditionally used when steps 2&3 are not integrated [12].

Unfortunately, there seems no comparison of these methods available

I.1.5 Skill-based routing

Skill-based routing is an active research field and a topic that draws much attention lately. Interesting results include:

- Simulation shows that performance is almost as good as the situation where agents have all skills if [15]:
  - Agents have appropriate combinations of 2 skills
  - The service-time distribution does not depend on the call type or the agent
- In the case of specialists (agent with one skill) and flexible agents [19]:
  - It is always optimal to send customers to a specialist first
  - If all appropriate specialists are busy, it is always optimal to send the customer to a flexible server, as long as the service time distribution of the flexible server is independent of the customer type.
  - An 80/20 rule works quite well, which means 80% of the budget goes to specialist, and 20% to flexible agents.

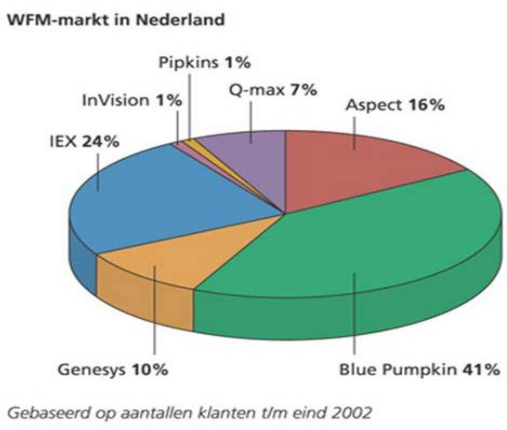
So it seems that a bit of flexibility goes a long way, which might be used as an argument to simplify the skill-based routing modeling.
Normally the skill-based routing situation is evaluated with simulation. For industry-standard overflow queues, a hyperexponential approximation can be used instead of simulation [23]. With overflow queues, the situation with specialist and generalists is meant, were ‘overflowing’ calls go to the generalists queue.
For limited problems, such as where all agents have 2 skills, or with 2 queues, scientific literature is available.

I.1.6 Multi-media / Multi-task scheduling (adding post, mail, chat, etc.)

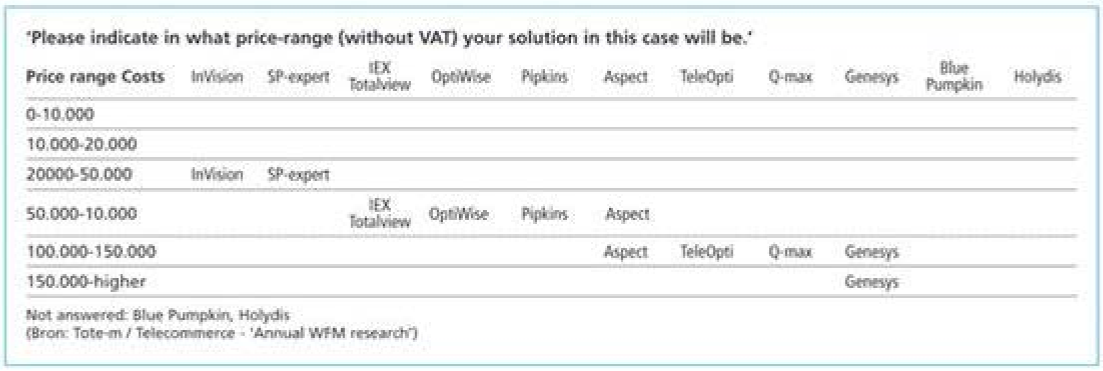
Higher level scheduling than on a routing scale is probably not yet tackled by scientific literature (see [5], p 50). For the evaluation method with preemptive extra media besides phone traffic, evaluation is of course not that hard if the phone is given priority (the spare time is put into a model for the other media).

I.2 Commercial applications

This section looks at some commercial applications for Workforce Management, especially on the Dutch market. For a larger list of company names, see [21].
A recent study on WFM [22] in the Netherlands shows the following distribution of the Dutch market in 2002:



This market share is biased to larger callcenters, not to medium or small callcenters where Anago is currently aiming at. For a callcenter with 160 seats, 200 agents, 2 sites and 3 languages, the cost indication from these companies is:



'Please indicate in what price-range (without VAT) your solution in this case will be.'

Price range Costs	InVision	SP-expert	IEX Totalview	OptiWise	Pipkins	Aspect	TeleOpti	Q-max	Genesys	Blue Pumpkin	Holydis
0-10.000											
10.000-20.000											
20000-50.000	InVision	SP-expert									
50.000-10.000			IEX Totalview	OptiWise	Pipkins	Aspect					
100.000-150.000						Aspect	TeleOpti	Q-max	Genesys		
150.000-higher									Genesys		

Not answered: Blue Pumpkin, Holydis
(Bron: Tote-m / Telecommerce - 'Annual WFM research')

Below are some products from professional companies and some smaller companies that write software for WFM.

I.2.1 Blue Pumpkin (www.bluepumpkin.com)

Blue Pumpkin, founded in 1997, is the market leader for this kind of software, at least on the Dutch market in 2002. In 2004, Frost & Sullivan ranked them first in worldwide share of the workforce management market as measured by license revenues. They schedule agents directly to create schedules, not in agent groups as IEX does. Blue Pumpkin claims *"Director Enterprise's patented skills-based scheduling engine takes into account the unique skill-set of each agent as well as their specific proficiency."*
From one patent[26] we get the following information on the scheduling, which basically is a local search procedure:

```
Loop until no Change made in Schedule or Maximum Iterations is exceeded:
        Loop over each Shift in the Schedule:
                If the Shift is unlocked then
                        Unschedule the Agent for the Shift
                        Loop over each Slot in the Schedule:
                                If Agent can work in the Slot then
                                        Calculate Score with the Agent in the Slot
                                Keep track of the Best Score and the Best Slot
                                If Useful to Schedule the Agent in the Best Slot then
```

```
                              Schedule Agent for the Best Slot
                              Note if Change was made to Schedule
        Loop over each Shift in the Schedule:
                Loop over each Break in the Shift:
                If Break is unlocked then
                        Loop over each Start Time for the Break:
                        Calculate Score with this Start Time for the Break
                        Keep track of the Best Score and Best Start Time
                        Schedule Break at Best Start Time
                        Note if Change was made to Schedule
```

The scoring procedure used is:

```
Loop over each Interval in the Slot
        Calculate the Interval Score
        If Stochastic Scoring is On then
                Scale Interval Score by Gaussian Random Number (mean 1.0, SD 0.1)
        If Interval is in a Break Window then
                Scale Interval Score by Break Length/No. Intervals in Break Window
        Add Interval Score to SlotScore
```

Stochastic Scoring prevents getting stuck in local optima. The interval scoring used is:

```
If Agents Scheduled in Interval<Agent Requirement-Understaffing Limit then Score=10
If Agent Requirement-Understaffing Limit<=Agents Scheduled in Interval  and Agents
        Scheduled in Interval<Agent Requirement then Score=1
If Agent Requirement=Agents Scheduled in Interval then Score=0
If Agent Requirement<Agents Scheduled in Interval<=Agent Requirement+Overstaffing Limit
        then Score=-1
If Agent Requirement+Overstaffing Limit<Agents Scheduled in Interval then Score=-10
```

The required number of agents is calculated using (extended) Erlang C for simple call centers. Simulation is used as one of the other available methods for the optimization engine. Blue Pumpkin employees made some scientific publications about it [18][20].

Blue Pumpkin has another patent on skill-based routing [28], which discusses scheduling. Basically it describes a scheduling system, but it also describes the relation between a scheduling and a simulator component. It describes a method to estimate effective staffing levels for each task without doing simulation at each step. Basically the simulation results are altered and recalculated a bit, in case an agent is added (agent gets workload from current agents) or removed (workload is distributed among existing agents).

Another patent by Blue Pumpkin [29] describes multi-task scheduling. It appears that the evaluation of immediate (phone) and deferred (email) queues is done completely independently. The total evaluation is a sum of these scores. Then some local search is used for the scheduling.
(It is the only description of a technique for this that I found, as there probably is not any scientific literature about it.)

I.2.2 IEX Corporation TotalView (www.iex.com)

IEX is probably ranked second, at least on the Dutch market in 2002.
It appears that first shifts for groups of agents are determined and then agents are allocated to these groups, because IEX has lost a patent suit against Blue Pumpkin on that point [24].

According to their patent, scheduling agents in case of skill-based routing is implemented by using simulation to calculate agent availability and adjusting the schedule according to the results. This is done until an 'optimal' schedule is created. Without skills, this is probably done using the steps forecasting, requirements, shifts, etc. There are 2 evaluation methods: extended Erlang(BErlang-C) and simulation.

I.2.3 Aspect Communications Corporation (Aspect TCS) (www.aspect.com)

Aspect is a larger US player active on the Dutch market. Aspect first calculates the amount of agents needed per hour, and then creates schedules that fit these requirements and finally assigns agents to these schedules. Skill-based routing is therefore handled with agents groups, instead of at agent level. It is interesting that shrinkage (absence, breaks, etc) is settable per hour and not per day as most programs have (so you can account for more shrinkage in the morning, etc.). The algorithms used are not explained, but there's certainly some 'Erlang' involved. Interestingly, they have a document describing different types of multiskill, multichannel call centers [25].

I.2.4 Genesys Telecommunications Laboratories (www.genesyslabs.com)

Genesys is a larger US player active on the Dutch market. They offer a platform that can also do workforce management. Besides a description of what workforce management is, little information is found on their website about the mathematical aspects of the process. They do have some information about the problems with Erlang C, but do not mention what they use themselves.

I.2.5 Teleopti (www.teleopti.com)

Teleopti is a Swedish company providing work force management software on the Dutch market.
From their demo on their website, it appears that first the number of agents per skill per quarter of an hour is determined, probably with some Erlang model. After that an unspecified local search algorithm is being used for scheduling agents.
This means that multiple skills are essentially seen as multiple queues I guess, so the advantages of skill-based routing are not taken.

I.2.6 SP-expert (http://www.sp-expert.de/)

SP-expert is a German company on the area of staffing active on the Dutch market. According to its website, it uses genetic algorithms to do so. It looks like they are planning at agent level, so they are using simultaneous planning. Service levels with skills-based routing are calculated using simulation.
They are working with the Abteilung für Betriebswirtschaftslehre und Produktionswirtschaft of the Technische Universität Clausthal for research. (Or maybe currently with the Institut für Produktionswirtschaft of the Universität Hanover, as Prof. Dr. Stefan Helber works there now.)

I.2.7 InVision (http://www.invision.de/)

According to their website:

"The scheduling methods used in the past were primarily based on intuition and were far too inflexible, resulting in unsatisfactory schedules. SchedulePro uses genetic algorithms and Tabu Search concepts to generate optimal schedules even in very complicated environments."
It looks like the number of agents per shift required is determined first, and then agents are assigned to these shifts later on. Since they also have a simulation component, for skill-based routing probably simulation is used for the evaluation.

I.2.8 Pipkins (http://www.pipkins.com/)
Pipkins is a British company operating on the Dutch market. It uses an extended version of Erlang C ('Merlang'), and does not need simulation to determine the number of agents needed, even in a multiskill environment. The extended Erlang model is claimed to be patented, but the patent could not be found. It therefore could not be determined how the algorithm works exactly. The scheduling takes place using (skill) groups. Interestingly they use the mathematical aspect for marketing, something that almost all others do not.

I.2.9 Q-Max (http://www.q-max.co.uk/)
Q-Max is a British company operating on the Dutch market. The algorithms/frameworks used could not be found, but most probably from the forecasted demand, shifts with certain skill(s) are generated. During one time-period, only one skill seems active.

The following are just some examples that workforce management can also be done a lot simpler, without skill-based routing or other problems. These simple programs could be used for validation/verification.

I.2.10 Portage Communications Demo (http://www.portagecommunications.com/)
Portage makes a WFM program intended for small to medium businesses (price class: $980.00 to $11,900.00). The website has a downloadable demo of the program, which could be used for some verification/validation. It uses Erlang or simulation for determining the staff needed (with user inputted number of agents in case of simulation). How the scheduling of agents works, is not described. As it is really fast in their demo and all agents have ranks, probably a very simple greedy algorithm is used. The program is not able to do various what-if scenarios or database connections (except spreadsheets). The simulation somehow uses about the same computation time if there are 0 calls and the case with 1000 calls. The simulation only simulates a day once, which makes it less useful.

I.2.11 HTL Telemanagement Hills Turbo Tables (http://www.htlt.com/)
This is a simple Excel add-in for Erlang calculations for smaller callcenters for $5,995. There is no simulator or scheduler available. Unfortunately, their demo could not detect Excel properly (perhaps due to a lack of administrator rights), so it could not be tested.

I.2.12 Agenses (http://www.agenses.nl/)
Agenses, started in 2003, is a small Dutch company offering Work Force Management Software (only forecasting and scheduling). The idea is that there is a central website where the work is done. Agents give their preferences on that website, and the planning and forecasting is done there. Forecasting of agents needed is generated with (extended) Erlang C and the scheduling of

the agents is done using some unspecified AI algorithm. They also offer a free Erlang-C calculator on their site and offer a 2 months free trial (as all agents will be involved, it will of course be difficult not to go through).

I.2.13 Kooltoolz cc-Modeler Pro(http://www.kooltoolz.com/)

Kooltoolz is Workforce Management Software as simple as it gets (price: $ 69). Based on shifts, the expected service level for each interval is calculated using the (extended) Erlang formula.

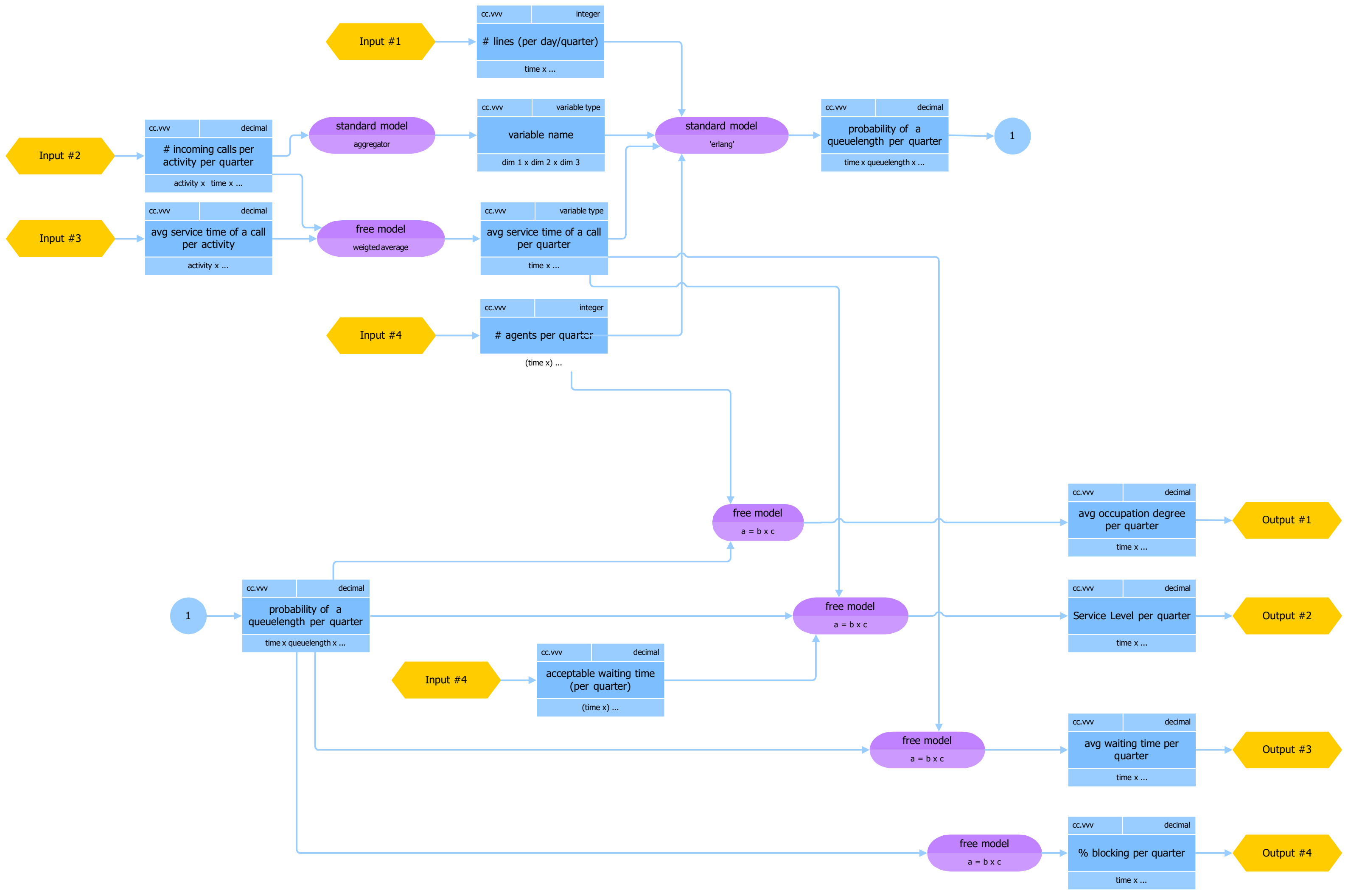
I.2.14 Demo's from larger applications

The following demos could be used for validation/verification:

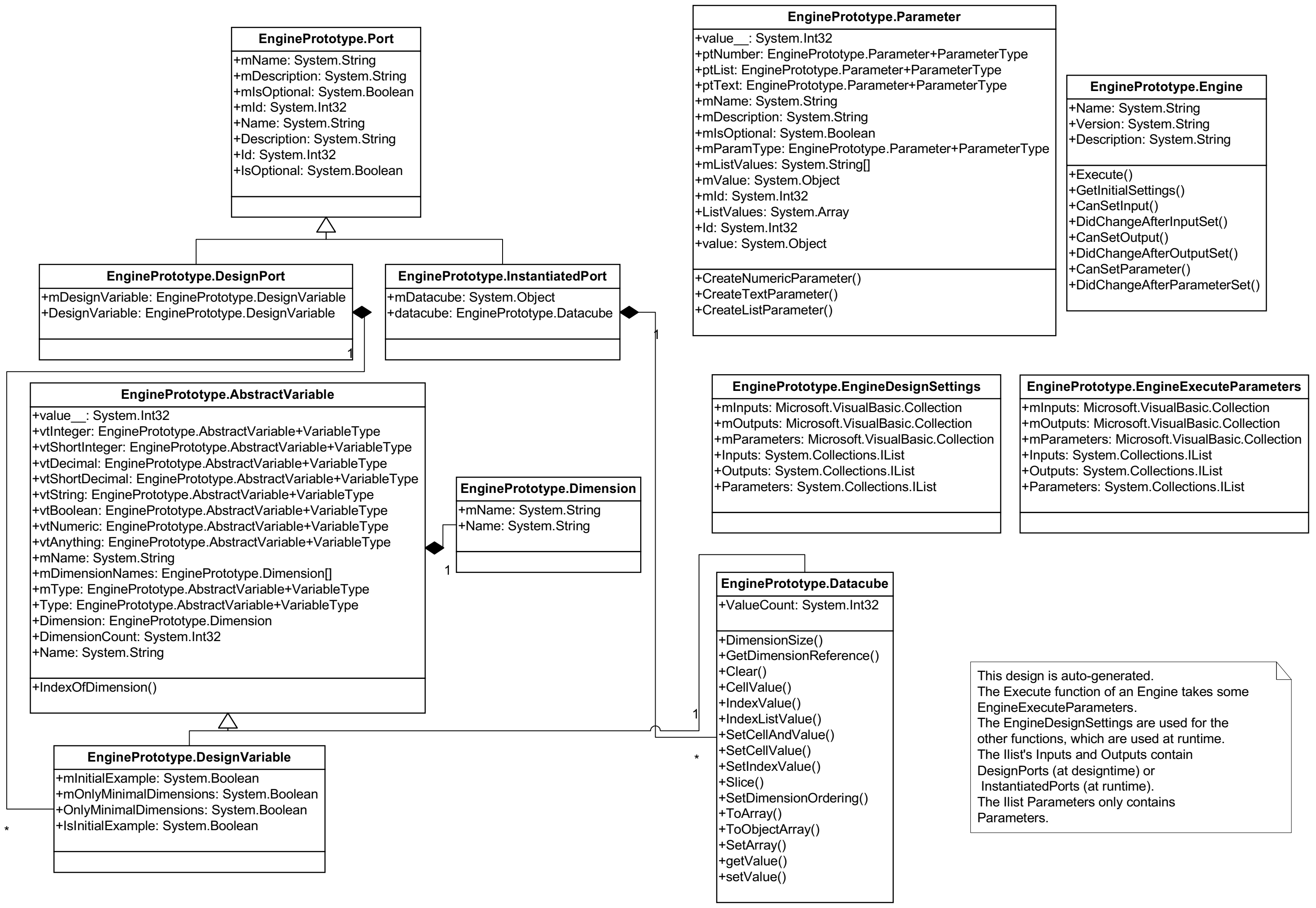
Pipkins Merlang Consulatant Demo
This is a demo of their calculation component.

Aspect Contact Center Calculator
This is a basic Erlang calculator from Aspect.

II ESTIMATING PERFORMANCE SCHEMA



III CLASS DIAGRAM OF THE ENGINE IN ANAGO


EnginePrototype.Port
+mName: System.String
+mDescription: System.String
+mIsOptional: System.Boolean
+mId: System.Int32
+Name: System.String
+Description: System.String
+Id: System.Int32
+IsOptional: System.Boolean
EnginePrototype.DesignPort
+mDesignVariable: EnginePrototype.DesignVariable
+DesignVariable: EnginePrototype.DesignVariable
EnginePrototype.InstantiatedPort
+mDatacube: System.Object
+datacube: EnginePrototype.Datacube
EnginePrototype.Parameter
+value__: System.Int32
+ptNumber: EnginePrototype.Parameter+ParameterType
+ptList: EnginePrototype.Parameter+ParameterType
+ptText: EnginePrototype.Parameter+ParameterType
+mName: System.String
+mDescription: System.String
+mIsOptional: System.Boolean
+mParamType: EnginePrototype.Parameter+ParameterType
+mListValues: System.String[]
+mValue: System.Object
+mId: System.Int32
+ListValues: System.Array
+Id: System.Int32
+value: System.Object
+CreateNumericParameter()
+CreateTextParameter()
+CreateListParameter()
EnginePrototype.Engine
+Name: System.String
+Version: System.String
+Description: System.String
+Execute()
+GetInitialSettings()
+CanSetInput()
+DidChangeAfterInputSet()
+CanSetOutput()
+DidChangeAfterOutputSet()
+CanSetParameter()
+DidChangeAfterParameterSet()
EnginePrototype.AbstractVariable
+value__: System.Int32
+vtInteger: EnginePrototype.AbstractVariable+VariableType
+vtShortInteger: EnginePrototype.AbstractVariable+VariableType
+vtDecimal: EnginePrototype.AbstractVariable+VariableType
+vtShortDecimal: EnginePrototype.AbstractVariable+VariableType
+vtString: EnginePrototype.AbstractVariable+VariableType
+vtBoolean: EnginePrototype.AbstractVariable+VariableType
+vtNumeric: EnginePrototype.AbstractVariable+VariableType
+vtAnything: EnginePrototype.AbstractVariable+VariableType
+mName: System.String
+mDimensionNames: EnginePrototype.Dimension[]
+mType: EnginePrototype.AbstractVariable+VariableType
+Type: EnginePrototype.AbstractVariable+VariableType
+Dimension: EnginePrototype.Dimension
+DimensionCount: System.Int32
+Name: System.String
+IndexOfDimension()
EnginePrototype.Dimension
+mName: System.String
+Name: System.String
EnginePrototype.EngineDesignSettings
+mInputs: Microsoft.VisualBasic.Collection
+mOutputs: Microsoft.VisualBasic.Collection
+mParameters: Microsoft.VisualBasic.Collection
+Inputs: System.Collections.IList
+Outputs: System.Collections.IList
+Parameters: System.Collections.IList
EnginePrototype.EngineExecuteParameters
+mInputs: Microsoft.VisualBasic.Collection
+mOutputs: Microsoft.VisualBasic.Collection
+mParameters: Microsoft.VisualBasic.Collection
+Inputs: System.Collections.IList
+Outputs: System.Collections.IList
+Parameters: System.Collections.IList
EnginePrototype.Datacube
+ValueCount: System.Int32
+DimensionSize()
+GetDimensionReference()
+Clear()
+CellValue()
+IndexValue()
+IndexListValue()
+SetCellAndValue()
+SetCellValue()
+SetIndexValue()
+Slice()
+SetDimensionOrdering()
+ToArray()
+ToObjectArray()
+SetArray()
+getValue()
+setValue()
EnginePrototype.DesignVariable
+mInitialExample: System.Boolean
+mOnlyMinimalDimensions: System.Boolean
+OnlyMinimalDimensions: System.Boolean
+IsInitialExample: System.Boolean
1
1
1
1
*
*

This design is auto-generated.
The Execute function of an Engine takes some EngineExecuteParameters.
The EngineDesignSettings are used for the other functions, which are used at runtime.
The Ilist's Inputs and Outputs contain DesignPorts (at designtime) or InstantiatedPorts (at runtime).
The Ilist Parameters only contains Parameters.

IV INPUTS, OUTPUTS AND PARAMETERS OF THE MODELS

IV.1 Inputs overview

PerformanceEstimator	AgentCalculator	optional	id	name	type	dimension1	dimension2	dimension3
x			0	Agents available	int	agent	period	
x	x		1	Acceptable waiting times	double	call		
x	x		2	Lines	int	-		
x	x	x	3	Skills per agent group	int	agent	call	
x	x	x	4	Threshold useful time	int	agent		
x	x	x	5	Threshold for outbound calls	int	agent		
x	x	x	6	Customer priority	double	call		
x	x		1000	Inter-Arrival mean	double	call	period	
x	x		2000	Service mean	double	call	period	*
x	x	x	3000	Abandonment mean	double	call	period	*
x	x	x	4000	Outbound mean	double	agent	period	*
	x		10	Shift times	boolean	shift	period	
	x	x	11	Shift costs	double	agent	shift	
	x	x	12	Minimum agents	int	agent	shift	
	x	x	13	Maximum agents	int	agent	shift	
	x	x	14	Wanted Service Level	double	call	period	
	x		15	Wanted Average SL	double	call		
	x	x	20	Acceptable period delay for other work	double	other		
	x	x	21	Can do other work	boolean	other	agent	
	x	x	22	Other work	double	other	period	
	x	x	23	Other work wanted SL	double	other		

*=In case of an empirical distribution (determined by the Parameters), an extra dimension is used, that can have any unused name.

IV.2 Outputs overview

PerformanceEstimator	AgentCalculator	optional	id	name	type	dimension1	dimension2
x	x	x*	0	Service Level	double	call	period
x	x	x	1	Waiting time	double	call	period
x	x	x	2	Blocking	double	call	period
x	x	x	3	Agent utilization	double	agent	period
x	x	x	4	Outbound calls	double	agent	period
x	x	x	5	Useful time left	double	agent	period
x	x	x	6	Abandonment	double	call	period
	x		10	Shifts used	int	agent	shift
	x	x	11	Agents available	int	agent	period
	x	x	12	Other work SL	double	other	

*=At AgentCalculator

IV.3 Parameters overview

PerformanceEstimator	AgentCalculator	id	name	type	values	default
x	x	0	(Final) estimation type	List	Simulation, Erlang	Simulation
x	x	2	Service distribution	List	Exponential, Lognormal, Empirical, etc.	Exponential
x	x	5	Maximum runs	Integer	Positive	25
x	x	6	Period Length	Integer	Positive	15
x	x	7	Linked periods	List	True, False	False
x	x	8	Abandonment distribution	List	Exponential, Lognormal, Empirical, etc.	Exponential
x	x	9	Outbound service distribution	Integer	Exponential, Lognormal, Empirical, etc.	Exponential
	x	10	EA Generations	Integer	Positive	0
	x	12	EA Population size	Integer	Positive	60
	x	13	EA Selection size	Integer	Positive	10
	x	14	SA Evaluations	Integer	Positive	0
	x	15	SA Temperature	Double	Positive	5
	x	16	Local Search	List	True, False	True
	x	17	Shift Transfers	List	True, False	True

IV.4 Descriptions of all inputs and outputs

IV.4.1 Agents available

An integer for each agent group and each period that gives the corresponding number of agents that is full-time available. Only a whole number of agents that is available is allowed. So, if during a period somebody goes to the toilet, the performance calculation will be slightly incorrect (because it is not accounted for and you cannot have 'half an agent').

IV.4.2 Acceptable waiting times

This variable is used to calculate the service level. The service level is something like x% of the customers is helped within [acceptable waiting time] seconds. It is for example 20 seconds (20/60=0.333 time-units, if everything is in minutes). For each call type, a different time can be used.

IV.4.3 Lines

There cannot be more customers in the system than this number. Customers, who could not enter the system because of the lines constraint, are reported in the blocking fraction.

IV.4.4 Skills per agents group

This variable is only useful if there are multiple skills and multiple agent groups. It determines which skills each agent group has. (For example it determines that agent group 1 speaks Spanish and English, while agent group 2 speaks English and French). A non-zero integer specifies that a skill is available. Currently, each other integer has the same effect (an agent group has the skill). In the future, other integers might be used to express a certain routing policy. In the current implementation however, the first suitable and available agent group gets an incoming call. If the Skills per agent group are not set, every agent group has all skills.

IV.4.5 Threshold for useful time

This threshold determines the useful time left. The useful time left is the time 'produced' by agents that are free in excess of this threshold. The threshold is settable per agent group. The default is 0. In case the threshold is 0, there is a relation between the useful time left and the agent utilization (if you know one of them, you can calculate the other one).

IV.4.6 Threshold for outbound calls

This threshold determines when an outbound call is done. An outbound call is done if there are agents free in excess of this threshold. The threshold is settable per agent group. If this threshold is not specified, no outbound calls are done. If you set this threshold, but do not set 'outbound call mean', the program might crash.

IV.4.7 Customer priority

Normally, if customers are in the queue, they all have the same priority. A First In, First Out (FIFO) policy is assumed. If this input is given, the priority determines the number of time-units a customer has extra priority. For example, a customer with priority 1 means that 1 time-unit is added to the waiting time for customers of that type. So in this example, the priority customer

will be helped earlier than a normal customer who already waits 0.3 time-units. He will not be helped earlier than a normal customer who already waits 1.1 time-units.
If you set the priority high enough (for example to the length of a day), it will be an absolute priority, because nobody normally waits that long.

IV.4.8 Inter-arrival mean
The mean of the distribution used for the inter-arrival times. The inter-arrivals times are always exponentially distributed, to prevent choosing the wrong distribution. This distribution determines the expected number of calls per time period. 1/the inter-arrival mean=the expected number of calls per time period. A negative inter-arrival mean might crash the component, while a value of 0 or infinity is seen as no arrivals.

IV.4.9 Service time mean
The mean of the distribution used for determining the service time, which is the duration of a call. Based on the parameters, it could be that not only the mean is settable. A negative service time mean will result in useless outputs.

IV.4.10 Abandonment mean
The mean of the distribution used for determining the abandonment time, the patience of a customer. Based on the parameters, it could be that not only the mean is settable. If a call spends more time in the queue than this patience, the customer abandons. Determining the abandonment distribution can be difficult, because the data is censored. You only know the patience for customers that did abandon, not for calls that were handled. For more information, see [5], 6.3.3.

IV.4.11 Outbound mean
The mean of the distribution used for determining the service time for outbound calls. Based on the parameters, it could be that not only the mean is settable.

IV.4.12 Shift times
Determines when a shift takes place. It is a Boolean for each period and each shift. When true, the shift does take place during that time period, when false it does not.

IV.4.13 Shift costs
Determines the costs that are made when a shift is scheduled, for each time the shift is scheduled. If no costs are given, the costs of a shift are assumed to be equal to the number of periods the shift covers.

IV.4.14 Minimum/maximum agents
The scheduling algorithm makes sure that the number of shifts used for each shift/agent group combination is in the interval [minimum; maximum]. (If minimum is not set, it is 0. If maximum is not set, it is infinite.)

IV.4.15 Wanted service level

The scheduling algorithm makes sure that the wanted service level, that is a service level per period and per call type, is met for every period and every call type. (Only if this is possible.) By default, this minimum required service level is 0. The service level is a fraction between 0 and 1.

IV.4.16 Wanted average service level

The scheduling algorithm makes sure that the average service level, that is a service level per call type averaged over the whole day, is met for every call type. (Only if this is possible.) The service level is a fraction between 0 and 1. 0.8 (80%) is a normal value.

IV.4.17 Service Level

Returns the service level (SL) per call type and per period. The service level is a fraction between 0 and 1. The average service level over a day can be determined by using a weighted average with the expected number of calls.
Blocked customers are not counted in the service level statistics. If the estimation is Erlang based, the service level is calculated as seen by customers who never abandon themselves. If simulation is used, customers who abandon before the acceptable waiting time are not counted. Customers who abandon after the acceptable waiting time are counted as 'failed SL'.

IV.4.18 Waiting time

Returns the expected waiting time in time units per call type and per period. The Erlang model returns the expected waiting time for customers who never abandon, while the simulation model reports the (slightly different) waiting time for 'real' customers (which are not counted if they abandon when they have to wait long, so the service level will be slightly better).

IV.4.19 Blocking

Returns the blocking fraction per call type and per period. The blocking is the fraction of customers that could not enter the system because of the Lines variable.

IV.4.20 Agent utilization

Returns the fraction of the time that agents are busy (per agent group and period). In case of other work (at AgentCalculator), this agent utilization is without taking the other work into account.

IV.4.21 Outbound calls

Returns the expected number of outbound calls made per agent group and per period. Returns only zeros when no threshold was set.

IV.4.22 Useful time left

Returns how much useful time occurred, in time-units. This output is closely related to the threshold. This value determines how much other work can be done. The amount of other work is not subtracted from the useful time left at AgentCalculator (so specifying other work has no effect on this output).

IV.4.23 Abandonment
Returns the fraction of customers that is expected to abandon per call type and per period. Returns only zeros when no abandonment was set.

IV.4.24 Shifts Used
Returns the recommended shifts, to satisfy the constraints while keeping the costs low. This is the main output of the scheduling.

IV.4.25 Can do other work
The variable is the same as Skills per agent group, but then for the other work. For each type of other work and agent group, true means that the agent group can do the other work. If this variable is not given, every agent group can do all types of other work.

IV.4.26 Other work
Specifies the amount of other work that arrives, per period and per type of other work. The amount is in time-units. The total amount of useful time left will at least equal the amount of other work after scheduling, as it is assumed that other work needs to be handled the same day.

IV.4.27 Acceptable period delay for other work
Specifies the number of periods that other work can be delayed to meet the service level. For example 0 means that the other work needs to be done in the same time period (quarter of an hour), while a very high number means that it needs to be done the same day.

IV.4.28 Other work wanted SL
Specifies the service level wanted for the other work, per type of other work. The scheduling tries to make sure that the service level will be met.

IV.4.29 Other work SL
Returns the expected service level for the other work, per type of other work. The service level for the other work is determined using the 'useful time left'.

IV.5 Descriptions of all parameters

IV.5.1 Distribution parameters (service, abandonment, outbound)
These parameters select the distribution to be used. The default is an exponential distribution. Based on the selection of the distribution, the number of inputs might change. The inter-arrival distribution is not settable in order to prevent a bad decision. (In case of independent arrivals, the inter-arrival times are always exponentially distributed.)

IV.5.2 Evolutionary Algorithm parameters (EA Generations, Population Size, Selection Size)
These parameters determine the behavior of the evolutionary algorithm during scheduling. The default is 0 generations, which means that the evolutionary algorithm is not used. The evolutionary algorithm is described in 7.4.6. Using the evolutionary algorithm is currently not recommended, as simulated annealing might be a better choice.

IV.5.3 Simulated Annealing parameters (SA Evaluations, SA Temperature)

These parameters determine the behavior of the simulated annealing algorithm during scheduling. The default is 0 evaluations, which means that simulated annealing is not used. The algorithm is described in 7.4.7. Currently, simulated annealing probably is a better idea than evolutionary algorithms. For example, you can do 1000 evaluations with temperature 5, which takes a few minutes. The choice of the temperature should be related to the Shift costs. (If you specify high shift costs, you should use a higher temperature because the fitness function values will be higher, see 7.2.)

IV.5.4 Local search parameters (Local Search, Shift Transfers)

These parameters determine the behavior of the local search during scheduling. 'Local Search' specifies whether local search should be used at all. 'Shift Transfers' determines whether shift exchanges are used for determining the neighborhood (because that makes the neighborhood quite large if there is a large number of shifts). Setting these variables to false will generate faster schedules that might be worse. I recommend setting these values to true, until it takes too much time.

IV.5.5 (Final) estimation type

Determines which estimator should be used, simulation or Erlang. If the type is Simulation and the component is the AgentCalculator, internally during the scheduling an Erlang estimator might be used first (hence 'final' is used). Simulation takes more time, but is more accurate and therefore recommended.

IV.5.6 Linked Periods

Determines whether uniformization is used in case an Erlang estimator is selected. Because of the bad results in the case study (6.2), setting linked periods to true is not recommended. (It probably takes more time than it is worth; probably simulation is a better choice.)

IV.5.7 Maximum runs

Determines the number of runs used by simulation (in case of simulation). The (recommended) default is 25. A higher number takes more time, but is also more accurate.

IV.5.8 Period Length

Determines the duration of a period. For example, in case time-units are minutes, and the time periods are quarters of an hour, the value should be 15 (the default).

V A SIMULATION FRAMEWORK

V.1 Introduction

V.1.1 Existing frameworks

Before writing a framework to be used for simulation, some existing packages were inspected. Most of the packages found were available for educational licensing only. Others were only used in-house. Only one package [37], was available for the .NET framework. It however appeared to be used mainly in-house only and seemed way too complex.
The non-.NET packages would have to be re-written anyway. For a list of existing frameworks, see [35] (which has a link to [36]).

Another possible problem with the existing packages was that they seemed too complicated, and needed time and understanding. Therefore, it was decided to create a new and simpler framework in VB.NET, while of course copying good aspects from the existing libraries.

V.1.2 A new framework

The newly developed framework is a VB.NET package, designed for speed, understandability and easiness.

The base of this framework is that in the simulation initial events are scheduled. The Event scheduler will get the run command, which executes events. So in pseudo-code, this is:

```
Schedule initial events
While events available
        Execute first happening event (CurrentEvent.Execute())
End While
```

If an event is happening, of course new events can be scheduled.
There are only 4 aspects available in the framework:

1. Event scheduler (PES, Pending Event Set, or FEL, Future Event List)
2. Random variate classes
3. Random generator classes
4. Statistics classes

A more detailed explanation of these aspects follows:

V.2 The event scheduler (PES, Pending Event Set, or FEL, Future Event List)

Almost all simulations need an event scheduler. The event scheduler takes care the correct ordering of events and, in this framework, of the running of the simulation.

Research has shown that under some conditions, about 40% of simulation execution time is due to the FEL structure [38]. Because of the high impact the event scheduler has on the performance, multiple implementations have been made. For callcenters with 100 agents under high load, the simulation with the splay tree was twice as fast as the simple linked list. The ladder queue is only interesting if there are 1000 agents or more, because it takes a little more time then the splay tree otherwise (which could be because of some implementation overhead).

V.2.1 A linked list (O(N) complexity)

This is the most basic event list found in most simulation frameworks. A description can be found in any basic data structures book and this was the initial event list.

V.2.2 A binary tree (O(log N) to O(N) complexity)

The binary tree is implemented, because it was used when I first implemented a bad version of a splay tree, which added nodes and splayed them separately later on. A description can be found in any basic data structures book.

V.2.3 A splay tree (O(log N) complexity, stable)

The splay tree has been motivated by the results of [39] and [40] and appears to be the best in this situation as expected from these papers. A description can be found in any basic data structures book.

V.2.4 A ladder queue (ladderQ, O(1) complexity)

This is a variation of the more widely known calendar queue, see [41] for a description. This is the only queue I found that should be O(1) under all conditions. It was the best queue found by researchers active in this field [38]. In the paper, the ladderQ pays of from a queue length of 100 and is slightly faster then the splay tree. My own implementation appears to be faster if the queue becomes greater than 1000 items or so, which will not be the case in most callcenters. Due to its internal structure, LadderQ might have some problems if there are single events scheduled in the far future, like an EndOfTheSimulation event.

V.3 The random generators

It appeared (see 4.5.1) that the 'uniform' random numbers provided by the Microsoft .NET System.Random class were not exactly random. It also has no support for random integers over their whole range, which are used by the Ziggurat method [32]. Therefore, alternatives have been implemented that implement IntegerSupportingRandom. IntegerSupportingRandom has a method (NextInteger) for providing random integers.

V.3.1 System.Random

Most Variates still support System.Random. Its use is however not recommended, as it produces numbers that are 'not random enough'. It appears that System.Random was written based on Ran3 of Numerical Recipes [30] [43]. Due to some minor changes (probably setting MBIG to Int32.MaxValue), it does not seem to behave random enough.

V.3.2 SHR3CONG

SHR3CONG is SHR3+CONG from the random SHR3 and CONG functions provided by George Marsaglia [33]. SHR3 was chosen because of its speed and good performance in tests and support for the Ziggurat method [32]. Because SHR3 does not pass all tests and Normal variates resulting from it may not be entirely random enough [34], SHR3 was extended with CONG.

V.3.3 LFIB4

Another fast random generator by George Marsaglia [33], that passes all tests in his DIEHARD test suite.

V.3.4 MCW

Another fast random generator by George Marsaglia [33], that passes all tests in his DIEHARD test suite.

V.3.5 Ran2

Ran2 is a well-known random generator from Numerical Recipes [30]. It is well tested and to the best of my and their knowledge did not fail a non-trivial test yet.

V.4 The random variates

A number of random distributions are available, which are all derived from the base class Variate. In the simulation, Variate.getNext() can be used to obtain the next random number from a distribution, that even does not need to be known. There is a base class Variate that can be used to instantiate the Variates, with the same .NET Random object.
Currently available distributions are:

V.4.1 Deterministic

Represents a deterministic variate, which simply always returns the average.

V.4.2 Exponential

Represents an exponential variate. Initially, the samples were generated using the inverse distribution function, so –log(U) was used, with U an uniform variable.
Later the Ziggurat method was implemented[32], which appears to be about twice as fast. A faster method could not be found.

V.4.3 (Standard) normal

Represents a standard normal variate. The implementation used is the Ziggurat method [32], which is probably the fastest normal variate generator currently available. One paper suggests that the Wallace method can be 3 times faster, but that is not a fair comparison. That difference is only possible if the method for generating uniform randoms is not so fast, because the Wallace methods generates random normals directly without use of uniform randoms (except for initialization).

V.4.4 Gamma

The gamma variate class is based on Marsaglia's simple method for generating gamma variates [30]. It is probably the fastest method currently available.

V.4.5 Lognormal

A lognormal variate is obtained by using exp(U), with U an normal variate.

V.5 Statistics classes

The statistic classes provide some means to calculate things like means and standard deviates over multiple periods and runs easily if needed. The following classes are available:

V.5.1 SimpleStatistic

For calculating and keeping a count of numbers and an average and a standard deviation over these numbers.

V.5.2 AllValuesStatistic

The same as SimpleStatistic, but records all values in memory for example for debugging.

V.5.3 AverageStatistic

For calculating and keeping a count of numbers and an average over these numbers.

V.5.4 ExtraAccurateAverageStatistic

This statistic does the same as AverageStatistic, but reduces the effect of round-off errors caused by using double precision variables. For example, if you add a small number to a large number (as in the current total), then it could well be that round off error causes the result to be the same as before adding the small number. However, a big lot of small number can be significant.

The method used to prevent round off error is a bucketing method. If a number is to be added, it is added to a bucket containing all numbers of about the same size. If after this adding the total value in a bucket is in the same size-range as the next largest bucket, the buckets value is added to the next largest bucket and the bucket is emptied. (To be exact only an amount that is representable is shifted to the next bucket.) This method reduces round off error to a situation where it almost does not exist.

V.5.5 SummedStatistic

For calculating and keeping a count of numbers and a sum over these numbers.

V.5.6 BooleanStatistic

For calculating and keeping a count of yes and no's. This class is not entirely necessary, as the same result can be accomplished with an AverageStatistic and adding only 1's and 0's.

V.5.7 AverageOverTimeStatistic

For calculating and keeping an average over the time. This statistic is used when you want to calculate the average over the time, if you know the value it takes from time a, b, c and d.

V.5.8 ValueOverTimeStatistic

The same as the result of the ValueOverTimeStatistic, but then multiplied by the time.

V.5.9 MultiRunStatistic

For using a statistic for a simulation with more than one run, MultiRunStatistic is provided. It is able to record a statistic (one of the previous ones) with a meta-statistic over multiple runs.

V.5.10 MultiRunStatisticHolder

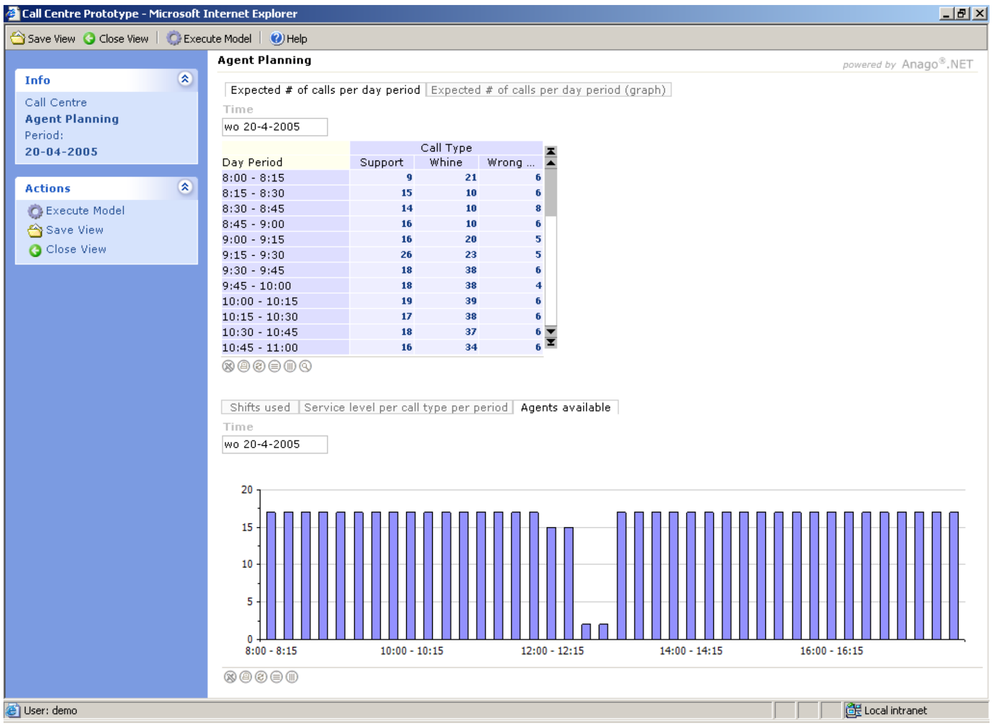
This class allows easy instantiation of MultiRunStatistics. It takes care of communicating the event of the end of a run to the MultiRunStatistics and is able to instantiate an array of MultiRunStatistics at once.

VI ROUNDING TEST RESULTS

Date	real			Simulation (normal rounding)			Rounded to below			Rounded to above		
	ext	not	int	ext	not	int	ext	not	int	ext	not	int
3-1	72%	72%	70%	71%	64%	67%	61%	51%	54%	83%	77%	78%
4-1	87%	88%	82%	88%	88%	86%	82%	84%	81%	92%	93%	91%
5-1	97%	95%	91%	94%	95%	93%	88%	90%	88%	96%	97%	95%
6-1	87%	81%	74%	86%	84%	83%	77%	73%	72%	91%	88%	89%
7-1	79%	76%	76%	83%	82%	79%	70%	69%	68%	88%	88%	88%
10-1	88%	91%	83%	89%	91%	89%	82%	83%	81%	92%	93%	91%
11-1	75%	87%	72%	83%	84%	82%	70%	71%	67%	87%	87%	84%
12-1	64%	84%	60%	81%	79%	77%	70%	68%	65%	88%	88%	86%
13-1	58%	74%	55%	77%	79%	76%	69%	70%	69%	86%	86%	86%
14-1	55%	76%	65%	76%	74%	74%	63%	59%	62%	82%	80%	82%
17-1	73%	83%	85%	80%	83%	79%	67%	74%	67%	84%	87%	84%
18-1	84%	85%	84%	90%	88%	88%	83%	81%	80%	94%	92%	92%
19-1	65%	82%	80%	81%	83%	84%	68%	74%	74%	85%	88%	88%
20-1	54%	59%	52%	55%	47%	49%	39%	33%	38%	63%	56%	61%
21-1	63%	76%	71%	73%	74%	75%	61%	62%	63%	81%	83%	83%
24-1	46%	64%	58%	69%	71%	70%	54%	58%	57%	75%	79%	78%
25-1	88%	92%	85%	88%	88%	85%	82%	83%	81%	92%	93%	93%
26-1	81%	89%	94%	88%	88%	86%	81%	80%	78%	93%	92%	92%
27-1	66%	68%	54%	72%	70%	67%	60%	57%	55%	81%	79%	78%
28-1	79%	79%	78%	79%	82%	76%	66%	71%	62%	84%	87%	81%
31-1	60%	67%	66%	62%	62%	65%	50%	50%	53%	74%	74%	75%
1-2	69%	75%	72%	75%	74%	77%	65%	66%	67%	83%	83%	84%
2-2	73%	75%	73%	77%	72%	72%	63%	57%	58%	84%	80%	81%

These are results from a test on the impact of rounding (see 6.2). It seems that the real values are (almost) always in the interval [rounded to below, rounded to above].

VII SOME SCREENSHOTS OF ANAGO

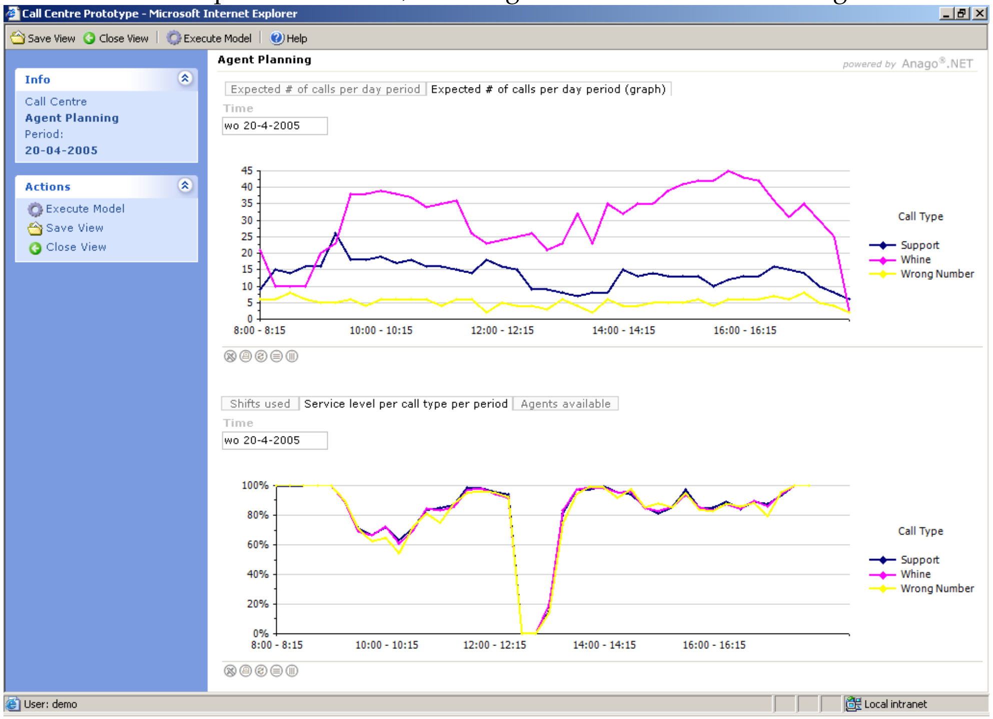


wo 20-4-2005

	Call Type		
Day Period	Support	Whine	Wrong ...
8:00 - 8:15	9	21	6
8:15 - 8:30	15	10	6
8:30 - 8:45	14	10	8
8:45 - 9:00	16	10	6
9:00 - 9:15	16	20	5
9:15 - 9:30	26	23	5
9:30 - 9:45	18	38	6
9:45 - 10:00	18	38	4
10:00 - 10:15	19	39	6
10:15 - 10:30	17	38	6
10:30 - 10:45	18	37	6
10:45 - 11:00	16	34	6

wo 20-4-2005

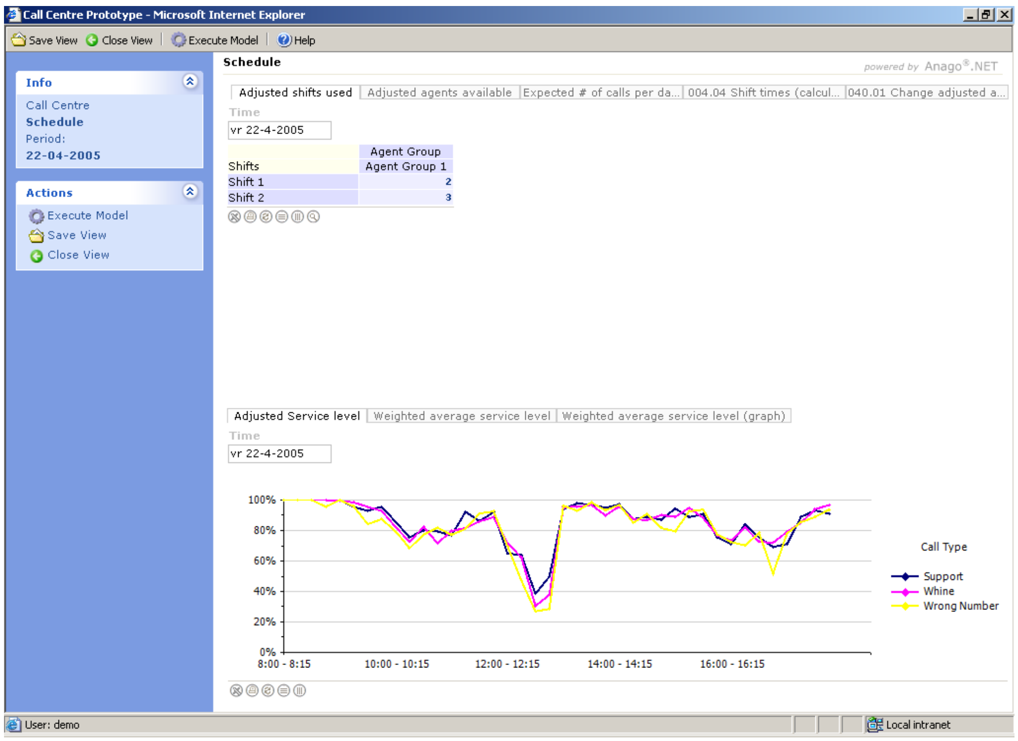
If we know the expected situation, we can get estimate the shifts and agents needed.



wo 20-4-2005

wo 20-4-2005

Here you can see the expected service level if you use that shifts. Note that during lunchtime, it seems better to at least meet the service level for one of the two lunchtimes.



Call Centre Prototype - Microsoft Internet Explorer

Save View | Close View | Execute Model | Help

Info
Call Centre
Schedule
Period:
22-04-2005

Actions
Execute Model
Save View
Close View

Schedule — powered by Anago®.NET

Adjusted shifts used | Adjusted agents available | Expected # of calls per da... | 004.04 Shift times (calcul... | 040.01 Change adjusted a...

Time
vr 22-4-2005

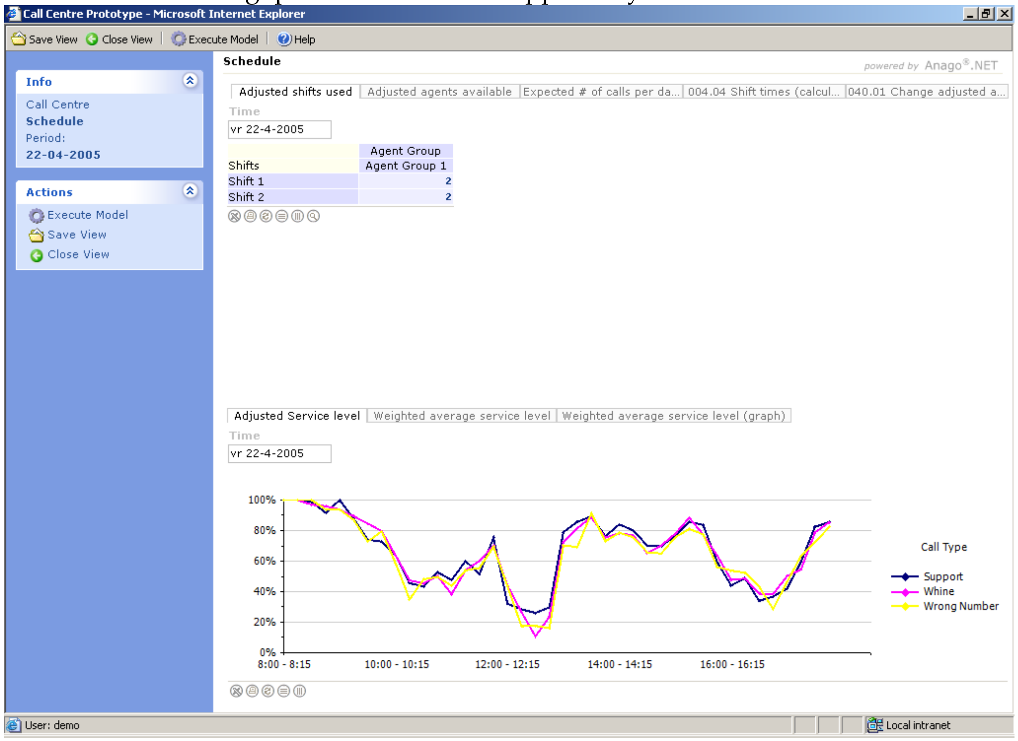
	Agent Group
Shifts	Agent Group 1
Shift 1	2
Shift 2	3

Adjusted Service level | Weighted average service level | Weighted average service level (graph)

Time
vr 22-4-2005

User: demo | Local intranet

One of the interesting questions is: What happens if you use one shift less?



Call Centre Prototype - Microsoft Internet Explorer

Save View | Close View | Execute Model | Help

Info
Call Centre
Schedule
Period:
22-04-2005

Actions
Execute Model
Save View
Close View

Schedule — powered by Anago®.NET

Adjusted shifts used | Adjusted agents available | Expected # of calls per da... | 004.04 Shift times (calcul... | 040.01 Change adjusted a...

Time
vr 22-4-2005

	Agent Group
Shifts	Agent Group 1
Shift 1	2
Shift 2	2

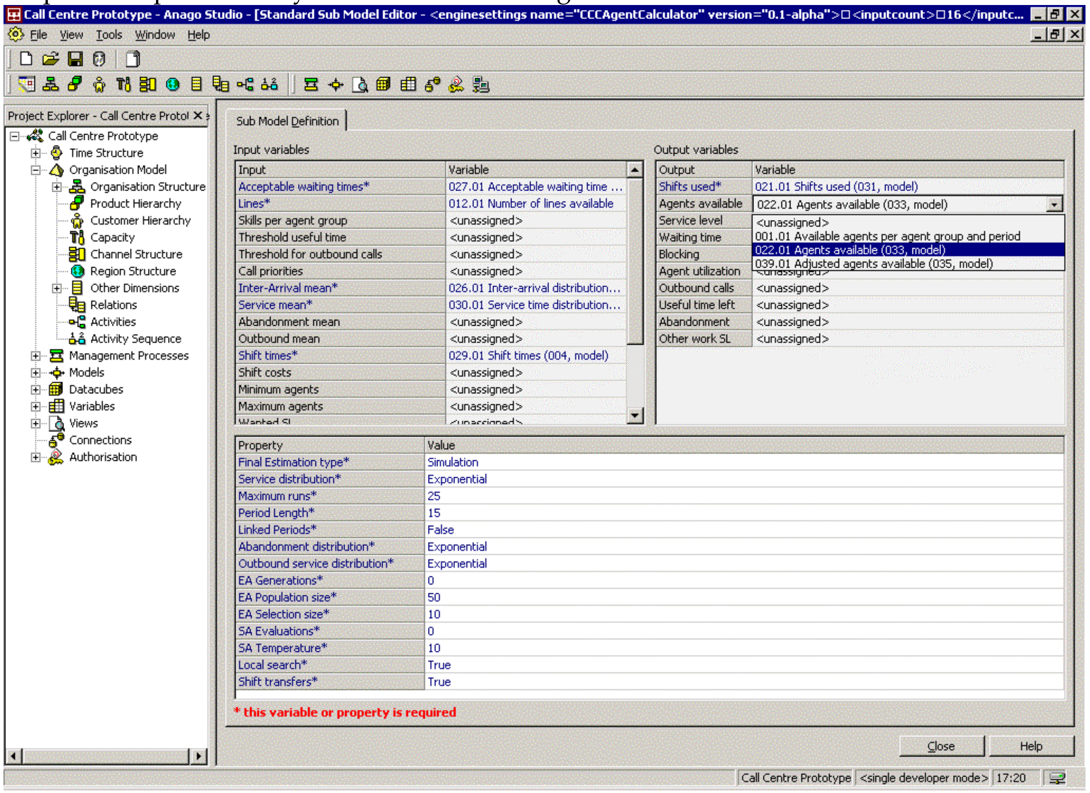
Adjusted Service level | Weighted average service level | Weighted average service level (graph)

Time
vr 22-4-2005

User: demo | Local intranet

Here you can see the results, the expected service level is quite a bit lower.

At design time, Anago Studio can detect which variables are suitable. Only valid options are presented to the users, so you do not get to many choices. Here is a screenshot of all inputs, outputs and parameters you can choose at design time:


Call Centre Prototype - Anago Studio - [Standard Sub Model Editor - <enginesettings name="CCCAgentCalculator" version="0.1-alpha">□<inputcount>□16</inputc...

Input variables

Input	Variable
Acceptable waiting times*	027.01 Acceptable waiting time ...
Lines*	012.01 Number of lines available
Skills per agent group	<unassigned>
Threshold useful time	<unassigned>
Threshold for outbound calls	<unassigned>
Call priorities	<unassigned>
Inter-Arrival mean*	026.01 Inter-arrival distribution...
Service mean*	030.01 Service time distribution...
Abandonment mean	<unassigned>
Outbound mean	<unassigned>
Shift times*	029.01 Shift times (004, model)
Shift costs	<unassigned>
Minimum agents	<unassigned>
Maximum agents	<unassigned>
Wanted SL	<unassigned>

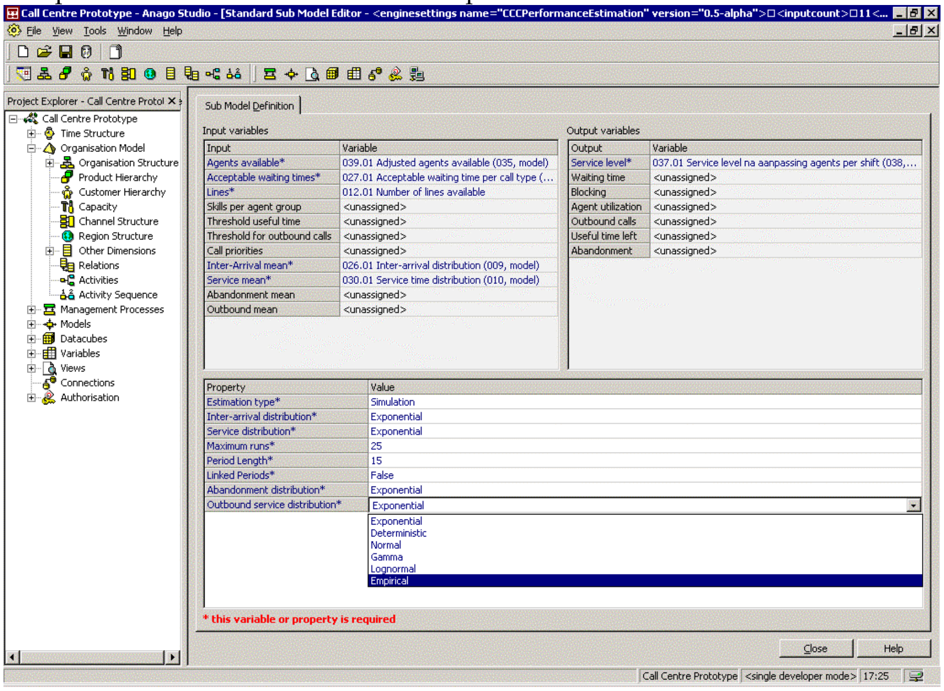
Output variables

Output	Variable
Shifts used*	021.01 Shifts used (031, model)
Agents available	022.01 Agents available (033, model)
Service level	<unassigned>
Waiting time	001.01 Available agents per agent group and period
Blocking	022.01 Agents available (033, model)
Agent utilization	039.01 Adjusted agents available (035, model)
Outbound calls	<unassigned>
Useful time left	<unassigned>
Abandonment	<unassigned>
Other work SL	<unassigned>

Property	Value
Final Estimation type*	Simulation
Service distribution*	Exponential
Maximum runs*	25
Period Length*	15
Linked Periods*	False
Abandonment distribution*	Exponential
Outbound service distribution*	Exponential
EA Generations*	0
EA Population size*	50
EA Selection size*	10
SA Evaluations*	0
SA Temperature*	10
Local search*	True
Shift transfers*	True

* this variable or property is required

The performance estimator has fewer options:


Call Centre Prototype - Anago Studio - [Standard Sub Model Editor - <enginesettings name="CCCPerformanceEstimation" version="0.5-alpha">□<inputcount>□11<...

Input variables

Input	Variable
Agents available*	039.01 Adjusted agents available (035, model)
Acceptable waiting times*	027.01 Acceptable waiting time per call type (...
Lines*	012.01 Number of lines available
Skills per agent group	<unassigned>
Threshold useful time	<unassigned>
Threshold for outbound calls	<unassigned>
Call priorities	<unassigned>
Inter-Arrival mean*	026.01 Inter-arrival distribution (009, model)
Service mean*	030.01 Service time distribution (010, model)
Abandonment mean	<unassigned>
Outbound mean	<unassigned>

Output variables

Output	Variable
Service level*	037.01 Service level na aanpassing agents per shift (038,...
Waiting time	<unassigned>
Blocking	<unassigned>
Agent utilization	<unassigned>
Outbound calls	<unassigned>
Useful time left	<unassigned>
Abandonment	<unassigned>

Property	Value
Estimation type*	Simulation
Inter-arrival distribution*	Exponential
Service distribution*	Exponential
Maximum runs*	25
Period Length*	15
Linked Periods*	False
Abandonment distribution*	Exponential
Outbound service distribution*	Exponential

* this variable or property is required