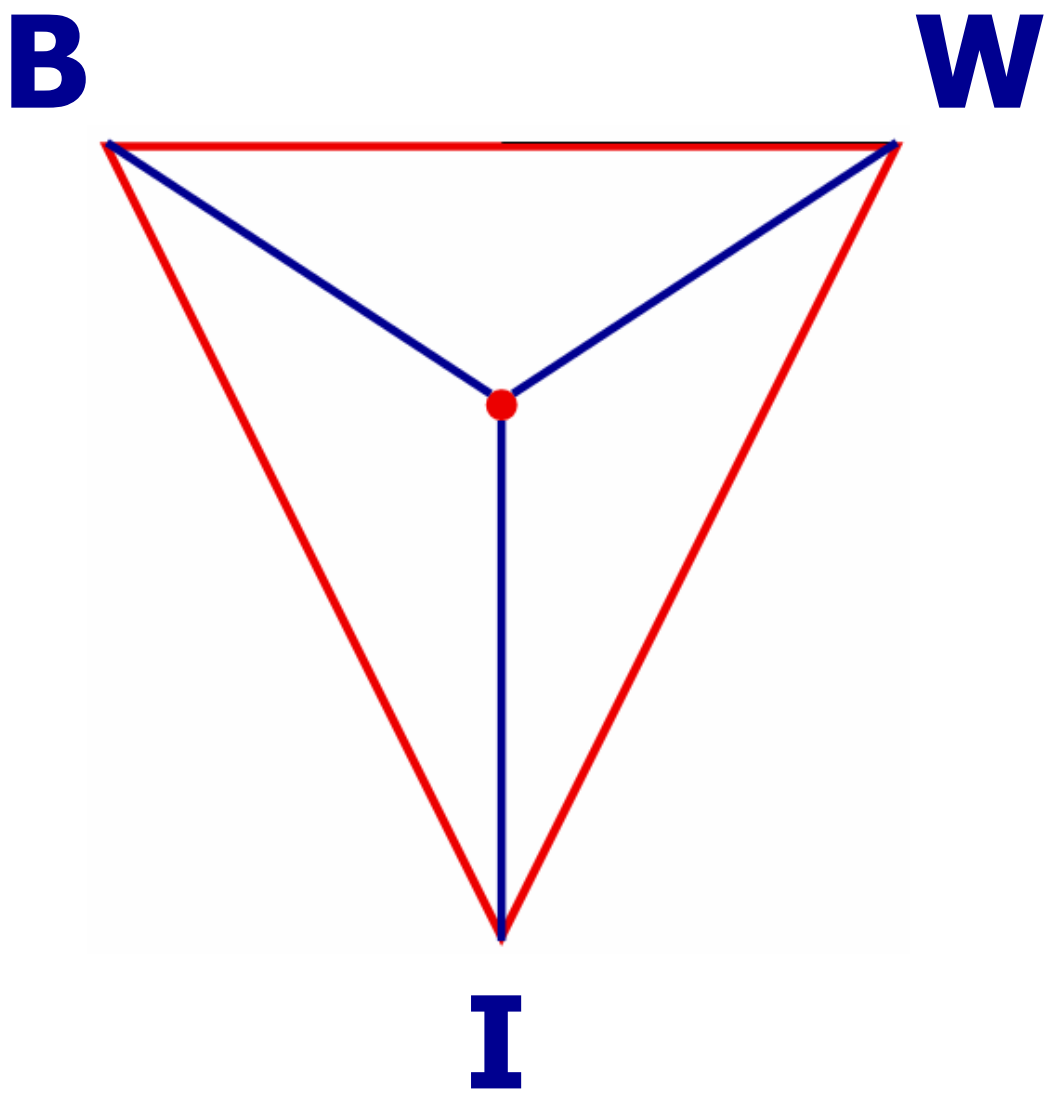
Using XML in Call Centers

BWI-Werkstuk



By Theo Peek
Supervisor: Prof. Dr. Ger Koole

Author: Theo Peek, supervised by Ger Koole.
You can contact me by sending an e-mail to tdpeek@few.vu.nl. Latest news, updates and examples can be found on my website: http://www.few.vu.nl/~tdpeek

Preface

Over the last two years, I have used the knowledge that I gained from my studies in Business Mathematics and Computer Science (in Dutch BWI for Bedrijfswiskunde en Informatica) and put it to practice in call centers on two occasions. The first was a business case that formed part of BWI; the second was my part-time job at the Management Information department of a large call center. Working with call center log data on those two occasions inspired me to choose "Using XML in Call Centers" as the subject of this research report.

The research report (BWI-werkstuk) forms a standard part of the BWI Masters degree. I started writing it in September 2002 and finished it in February 2003. Naturally, I owe many thanks to the people who helped me during my writing.

First of all I would like to mention my supervisor Prof. Dr. Ger Koole, who is the leading Dutch call center researcher. I am honored to have worked with him and I would like to thank him very much for his criticism and inspiration. Second reader Prof. Dr. Frank van Harmelen was the first to be enthusiastic about my idea, for which I owe him many thanks. Drs. Jeen Broekstra and Drs. Michel Klein helped me by giving me several XML tips and corrections; thank you both. Special thanks go to Drs. Maaike van den Haak, who never grew tired of telling me that data are plural. I would like to thank her for the many hours she put into correcting my English.

Theo Peek
February 2003

Management summary

In a call center, management reports on prior performance are vital. Based on the information in these reports, scheduling decisions for the next time period are made. Those decisions in turn influence the performance and customer service level in that period.

Management reports are based on call center log data, which contain a lot of information about each call arriving at the call center. In my experience, these log data are very complex and hard to understand, particularly since these data are stored in tables. Tables are not capable of capturing the sophisticated process of a call in a call center, because of their simple and rigid structure.

I have invented a better way of logging calls than using traditional tables. XML (eXtensible Markup Language) is a relatively new technology, which will certainly become more common in the near future. Using XML offers clear advantages over tables:

- The sophisticated process of a call can be better modeled. This in turn leads to the following advantages:
  - Data are more detailed;
  - Data are easier to understand;
- XML offers an excellent validation mechanism called XML Schema. A lot of computer programs can use this validation to read data, which means that XML data can be processed in other ways that offer more possibilities than tables.

Using XML for call logging leads to call log data that are easier to understand. As a result, call centers might be able to downsize their Management Information department or create richer management reports as a result of the more detailed log data.

As a compromise, it is possible to convert the currently used tables into XML and test them. The data cannot be as detailed as described before, because data are stored less detailedly in tables, but this does offer the opportunity to test and experience the advantages that XML offers, without losing the proven safety of database technology.

Index

1 Introduction

1.1 Problem statement

In this report, I want to answer the following question:

- Is XML a useful data structure for call center log data?

This leads to the following sub-questions:

- What is a call center?
- Why is the current data storage not feasible?
- What is XML?
- How can I use XML to store call center log data?

1.2 Structure of this report

The main question and the resulting sub-questions are dealt with in separate chapters:
Chapter 2: What is a Call Center?
Chapter 3: Why are tables not a feasible data structure?
Chapter 4: What is XML?
Chapter 5: How can I use XML to log my calls?

In Chapter 6, I will present my conclusions and in Chapter 7 I will present suggestions for further research.

1.3 For whom is this report?

This report is aimed at anyone with an interest in call centers or XML. Call center managers and researchers will probably know all about call centers, but may not be aware of the advantages of XML. I recommend these readers to scan Chapter 2 and read Chapters 3 through 7. While the mathematics and XML programming in this report might be a bit difficult to understand, the main problem and the solution should be generally comprehensible.

XML experts will not need a quick XML course, but may be interested in the application of XML to call centers. I recommend these readers to read Chapters 2, 3, 5, 6 and 7 and scan Chapter 4. Again, mathematics can be complex, but readers will certainly be able to understand the problem and solution I present.

1.4 Website

The XML examples I use in this report can all be downloaded from my website: http://www.few.vu.nl/~tdpeek.

2 What is a Call Center?

2.1 Introduction

In the modern information-centered society, a lot of organizations have trouble dealing with the ever-growing flow of telephone calls. Banks do not want their expensive financial experts listening to long-winded customers, while customer service is still essential. A call center could be the solution to these problems. Customers can call a central (possibly toll-free) number for all their questions, and be directed to the call center, where a number of telephone experts are ready to assist them.

In this way, only the agents in the call center deal with customers directly, freeing more time for other employees of the organization, who can focus on their specific expertise. Agents can get special training, which will turn them into telephonic customer service professionals.

In this chapter, I will describe the process of a call in a basic call center as well as in a more advanced call center. Then I will introduce a mathematical model used to optimize the processes in call centers and the data analysis needed for this model.

2.2 A basic call center

2.2.1 The process of a call

A basic call center is typically small, usually with fewer than ten agents and without any complex routing. The agents are equally trained and qualified and all customers require a similar service. The process of a call can then be divided into two phases. An activity diagram of the first phase can be found in Figure 2.1. Please note that this is not a valid UML activity diagram; I have made some departures for the sake of readability. For more information about UML, I recommend [3]. Let us walk through this diagram step by step.

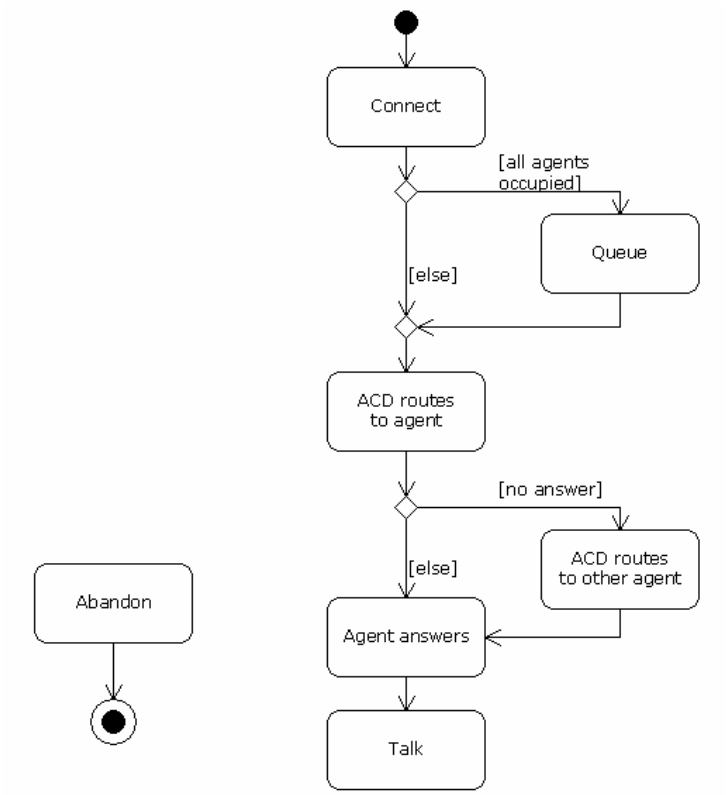


Figure 2.1: First phase of the process of a call in a basic call center

The process starts with Connect, when the call is connected to the call center. From this point onwards, the customer can abandon at any time. Because this can happen at any activity state in the process, I have omitted the connectors to the Abandon state for a clear organization. If all agents are busy at the time, the call is queued. When an agent is available, the Automatic Call Distributor (ACD) routes the call to that agent. The agent should answer the call, but if he or she does not, the ACD routes the call to another agent. In a well-managed call center, this should of course never happen. After the agent answers the call, the second phase begins.

The activity diagram of the second phase can be found in Figure 2.2.

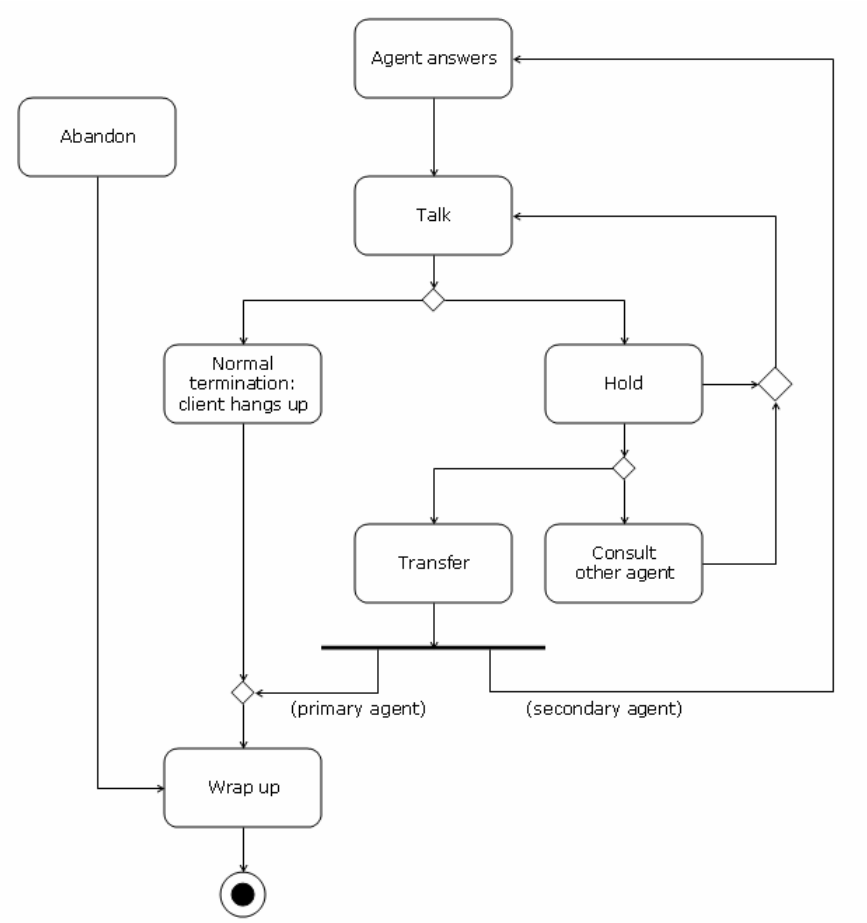


Figure 2.2: Second phase of the process of a call in a basic call center

In the second phase, the actual conversation takes place. When the call ends in this phase, wrap-up time follows the abandonment or termination, allowing the agent to make notes about the call. After the activity Talk, two activities can follow: termination, if the customer is satisfied with the service, or Hold. During Hold, the agent can, for instance, quickly look up some information. In that case, after Hold, the agent retrieves the call. The agent can also choose to consult another agent by initiating a second, internal call, which is usually short. When the agent knows enough to help the customer, the agent drops the internal call and retrieves the customer call. When the problem is too difficult to handle for the agent, he or she can choose to transfer the customer call to another agent. The primary agent then performs wrap-up for the call, which ends the call for that agent. The secondary agent answers the customer call. The call follows the original activity diagram from that point onwards, but now for the second agent.

2.2.2 The challenge

The basic call center is ideal to describe the challenges call center managers are faced with. A manager cannot influence the number of calls that a call center has to deal with. By training agents, he or she might be able to influence the average service time of calls, defined as the total time an agent spends on a call (talk time + hold time + wrap-up time). But the only thing a manager has total control over is the number of agents he or she hires. If a manager hires fewer agents, he or she will save salary and training expenses, but this, of course, increases the risk that a customer will find all agents busy. This then results in a longer average waiting time and an inferior customer service.

On the other hand, when there are fewer agents, their productivity is quite high. The more agents are hired, the more they are available, waiting for calls. Agents can be in three states: available, busy or idle. When an agent is talking, on hold or doing wrap-up, he or she is busy. When he or she is taking a break, in a meeting or doing other work, the agent is said to be idle. Idle cannot be avoided in many situations, so productivity can be defined as [busy time] / ([busy time] + [available time]).

To measure customer service, a manager could use the average waiting time, but a more common quantity is the service level. With a threshold of x seconds (typically, $x = 20$), the service level is defined as the percentage of calls answered within x seconds. As a target, a percentage of 80 is often used, so managers try to hire as few agents as possible while maintaining a service level of 80%. More details on this follow in Paragraph 2.5.

2.3 Skill groups

As a call center gets larger, it becomes more and more difficult to equip all agents with the same knowledge and skills. A large call center typically provides customers with different services and therefore it is only natural that agents have different fields of expertise. A way of dealing with differently trained agents is to assign them to skill groups.

Consider an international call center, which serves customers from Spain and France. Some of the customers speak French, while others require a Spanish-speaking agent. The call center would do wise to divide its agents into two skill groups: Spanish-speaking and French-speaking. Some agents, who speak both, are assigned to both skill groups.

The problem that then arises is how to tell which customer requires which skill. There are four ways to find out which service a customer requires:

- By hiring a first-line group of agents who determine the service the customer requires and who then route the customer accordingly;
- By identifying the number the customer calls from (ANI);
- By identifying the number the customer dialed (DNIS);
- By letting customers choose from menu-items (IVR).

Hiring a first-line group of agents would not be wise in this example, but could be useful in others. Consider, for instance, a bank, whose customers can call with questions about their mortgage or their stocks. Using an extra line of agents does not change the process described in Paragraph 2.2.

Identifying customers by Automatic Number Identification (ANI) is a possibility that would work well in an international call center, provided that customers have enabled ANI, since each country uses a different international access code. In other situations, such as our bank example, this method is not feasible. A bank does not know the phone number of first-time callers (potential customers) and will most certainly have customers with both stocks and a mortgage. Furthermore, if customers call from work or their cell phones, the number is not always recognized. However, if ANI is used, this does change the process of Paragraph 2.2, as will be discussed later on in this paragraph.

Identifying customers by Dialed Number Identification Service (DNIS) is a more practical idea, especially in a multi-lingual call center. Customers can dial a different number for each service they require. However, as the number of different services becomes larger, this does get more complex. Companies cannot expect their customers to learn and use too many different numbers. As with

ANI, identification by DNIS changes the process of Paragraph 2.2; these changes will also be discussed later on in this paragraph.

In almost all situations, the use of an Interactive Voice Response system (IVR) is practical. Customers can choose from a number of menu choices ("Para español, pulse uno – For English, press two – Pour la version francaise, choisir le trois") or even successive menu choices. Moreover, an IVR can make agent service redundant, for instance in the case of an account balance line. The use of an IVR changes the process of a call, as shown in Figure 2.3.

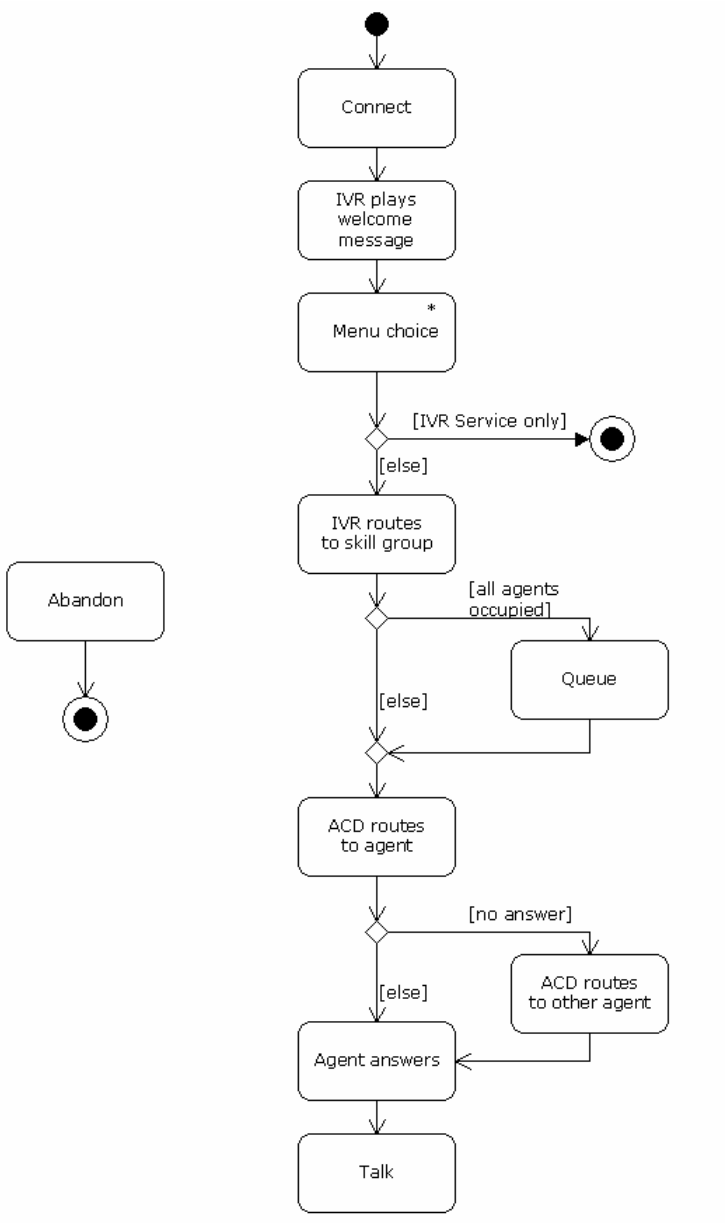


Figure 2.3: First phase of the process of a call in a call center using skill groups

After Connect, the IVR plays a welcome message and the first set of menu choices. The customer then chooses one of them. A customer can make successive menu choices, denoted by *. If a customer is served by the IVR only, i.e. without having spoken to an agent, the call now ends. If the customer wishes to speak to an agent, then the IVR assigns the call to a certain skill group, based on the menu choices. This could also be done by the ACD, depending on the technical

configuration of the call center. If all agents in the skill group are engaged, the call is queued, waiting for an agent to become available. From this point onwards, the process is similar to the process described in Paragraph 2.2. This activity diagram is easily adapted to identification by ANI or DNIS.

2.4 Further enhancements

In order to increase productivity or service level, a few other enhancements can be made. These include call blending, virtual call center and multi-media. Enhancements like this are beyond the scope of this report, but the solutions offered in Chapter 5 can easily be adapted to the situations that these enhancements present. For more information, see [1].

2.5 Mathematical models

The models presented in this paragraph illustrate the need for data collection and analysis.

2.5.1 Basic mathematics

To optimize the process, for instance by determining the optimal number of agents, mathematical models can be used. For this purpose, we have to consider a call center as a queueing system. An example of a queueing system can be found in Figure 2.4.

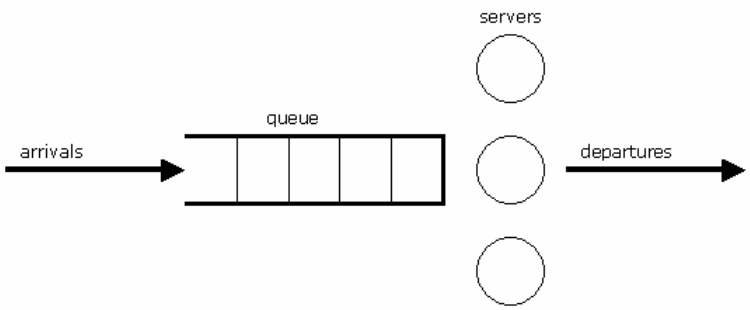


Figure 2.4: A queueing system

In a queueing system, there are arrivals, a queue, a number of servers and departures. Calls to a call center can be seen as arrivals in the same way as customers arrive at a store or products arrive at a sealing machine. If all agents, shop assistants or the sealing machine are busy, the call, customer or product is placed in a queue before being served. There can be one ore more servers: one, in the case of the sealing machine, and more than one in the case of shop assistants or agents. As the servers do their job, the queue size decreases. One by one, all calls, customers or products are being served. After service, there is a departure: a customer hangs up or leaves the store or the product is neatly sealed.

To apply mathematics to this system, we have to measure the named quantities. We have to know how many arrivals there are, how long the service time is and what the number of servers is. We measure the arrivals as a rate or intensity: the arrival rate λ, defined as the number of arrivals per time unit (e.g. second, minute or day). From an existing queueing system, λ can be calculated by measuring every inter-arrival time: the time between two successive arrivals. Then $\lambda = 1 /$ [the mean inter-arrival time].

If we plot these inter-arrival times in a histogram, we get an empirical distribution of inter-arrival times. If a mathematical probability distribution fits to the empirical distribution, we model the arrivals using the mathematical probability distribution. Often, the exponential distribution is used,

because of its practical properties: it is for instance memoryless. This distribution looks like Figure 2.5. With inter-arrival times exponentially distributed, the process of arrivals is called a Poisson process.

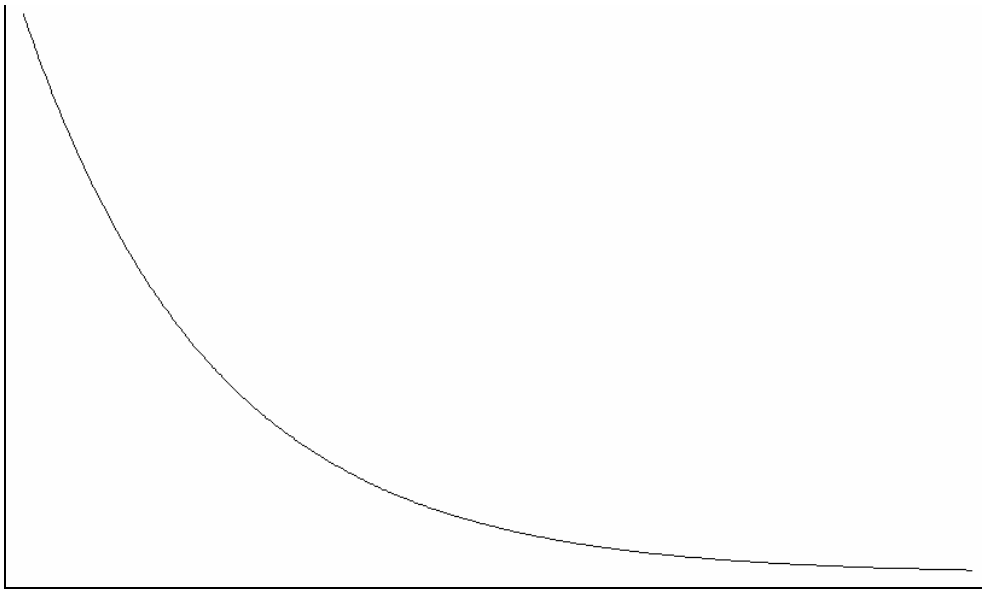


Figure 2.5: The exponential distribution

To measure the service times, we calculate the mean service time S from all available service times. In further calculations, we often need a rate (like the arrival rate), so we define the service rate $\mu = 1 / S$. Like the inter-arrival times, we can also fit a probability distribution to the service times, and again, the exponential distribution is often used.

Having modeled the queueing system with exponential inter-arrival times and service times, we get an $M/M/k$ model (also referred to as Erlang C model) with parameters λ and μ. The first M (for Markovian) says: the arrival process is a Poisson process with parameter λ. The second M (again for Markovian) says: the service times are exponentially distributed with parameter μ. k is the number of servers in the system.

Using this model, k can become as large as we wish. In reality, the number of telephone lines bound the number of agents in a system. If a call center only hires 100 trunk lines from its telephone company, engaging 120 agents would be useless, as only 100 customers at a time can be connected to the call center. Since the $M/M/k$ model assumes an infinite system capacity, the $M/M/k/K$ model (also referred to as Erlang B model), with K the system capacity or number of trunk lines, describes reality better. If the total number of customers in the system reaches K, arriving calls are blocked until a customer leaves the system. [2] even describes situations in which it is advantageous to block calls when there is enough capacity. In reality, however, most call centers equip themselves with an ample number of trunks, because of the relatively low costs of an extra trunk line. In view of the main problem statement, this report does not deal with a blocking situation. Therefore, the $M/M/k$ model will be used from this point onwards.

Another shortcoming of the M/M/k model is that abandonments do not occur. This is a result of the infinite-patience assumption of the M/M/k model. In reality, customers have a finite patience. Research even shows that *general* patience can be analyzed [1]. Therefore, the M/M/k+M model

(also referred to as Erlang A model) can be used, where +*M* stands for Markovian patience: an exponentially distributed patience.

All three models described assume all durations to be exponentially distributed. These models can be generalized by replacing an *M* with a *G* for general. In [10], for instance, the service times were lognormal distributed, see Figure 2.6. These models can be used to describe reality better, but they are less powerful in calculation. Therefore, they are often used in simulation models.

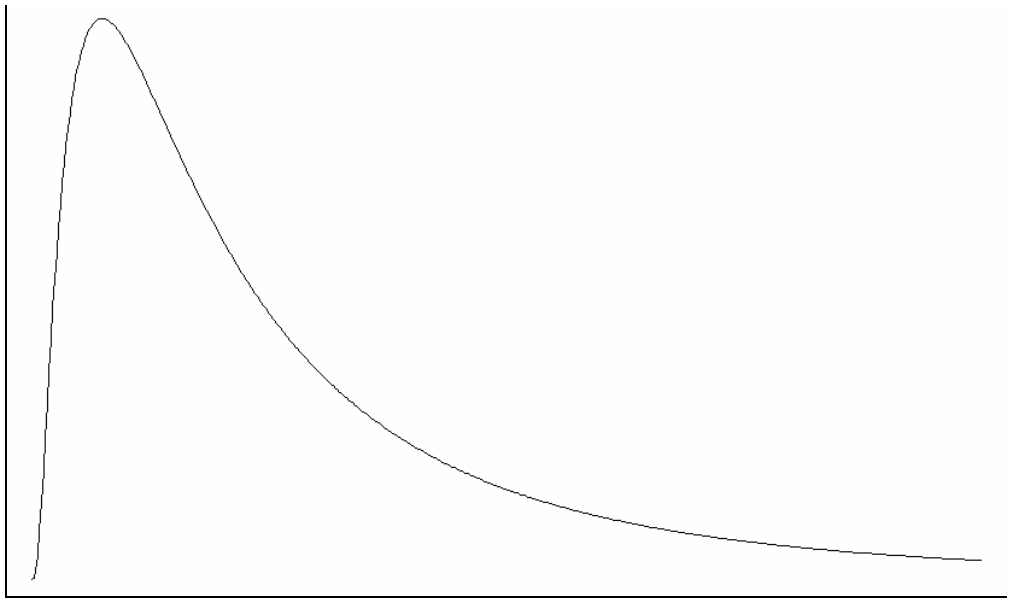


Figure 2.6: The lognormal distribution

2.5.2 Advanced mathematics

When λ, μ and k of the $M/M/k$ model are known, we can predict the service level by the Erlang (C) formula. This formula holds if the system is stable, which means that the agents are able to handle all inbound calls. To verify that, I introduce the load $a = \lambda\mu$. The agents are able to handle all calls if $a < k$, so if $a < k$ the system is stable. The Erlang formula is:

$$SL = 1 - C(k,a)e^{-(s-a)\frac{t}{\mu}},$$

with

$$C(k,a) = \frac{a^k}{(k-1)!(k-a)}\left(\sum_{j=0}^{k-1}\frac{a^j}{j!} + \frac{a^k}{(k-1)!(k-a)}\right)^{-1}$$

the probability that an arbitrary customer finds all agents busy. This probability is often referred to as the probability of delay.

We can use the Erlang formula to gain all sorts of insight. We could, for instance, be interested in the effect of changes to the parameters on our call center. For example:

- How much lower would our service level be if the arrival rate increases by 10%?
- If we can reduce service times by ten seconds, what will the effect on our service level be?
- If we hire one more agent, will our service level finally reach 80%?

2.6 Data collection and analysis

The answers to these questions can be predicted with the Erlang formula, as long as we have data to extrapolate. For example, if we wish to answer the first question, we must assume that our service times and number of agents remain the same. So we analyze historical data to determine the distribution and mean of the service times and the current arrival process. We should of course also know the current number of agents, but that is likely to be the case. Call center managers will also use data to check their agents' productivity.

Here we see the importance of data collection. This is the first step towards management information: keeping track of everything that happens. There is a huge difference between data and information: data become information, if the figure presented is relevant to the recipient. A call center manager is not interested in the fact that there was a call at 9:34:57 a.m. if there are hundreds of calls every day. In that case, the fact that there was a call at 9:34:57 a.m. is an example of data. However, the fact that on 17th October 2002 75% of all calls were answered within 20 seconds *is* relevant to a manager and therefore information. Creating management reports like the one in Figure 2.7 from a database is equivalent to getting information from data. This information can then be analyzed for various purposes, for instance in determining the parameters for the Erlang formula.

At the bottom page 20, a typical weekly call center management report is shown. It is taken from an actual call center, but for confidentiality reasons, I have altered some of the information. However, it is very representative. The report can apply to the whole call center, or to a single skill group. Let us assume that it applies to a skill group. For each half hour, the following figures are shown:

- Logged on time in hours, minutes and seconds;
- Percentage of time that the agents in the skill group spent on:
  - Inbound calls;
  - Outbound calls;
  - Wrap-up;
  - Idle;
  - Available.
- The absolute number of inbound calls that the agents in the skill group handled;
- The absolute number of outbound calls that the agents in the skill group handled;
- The absolute number of abandoned calls that were routed to the skill group;
- The average number of inbound calls that the agents in the skill group handled in one logged on hour;
- The average number of outbound calls that the agents in the skill group handled in one logged on hour;
- The average duration of:
  - Inbound calls;
  - Outbound calls;
  - Wrap-up;
  - Answering;
- One Touch Ratio: the percentage of calls that agents handled without transferring them to other agents;
- Service level: the percentage of calls that were answered within 20 seconds;
- Productive Time Ratio: Percentage of time spent on inbound plus the percentage of time spent on outbound.

At the bottom of the report, a summary row states the averages or totals for the entire day.

The number of inbound calls is used to determine the arrival rate. In this report, the skill group received 105 calls during the half hour starting at 11:30 a.m. For that half hour, the arrival rate λ would be one of the following:

- 105 per half hour;
- 210 per hour;
- 3.5 per minute;
- 0.058 per second.

It is not relevant which one we choose, as long as we use the same time-unit (hour, half hour, minute or second) throughout the whole model. For this report, I have decided to use minutes, so $\lambda = 3.5$.

The average service time is equal to the average length of inbound calls plus wrap-up. For this skill group in this half hour, it is 135.5 + 51.7 = 187.2 seconds ≈ 3.12 minutes. That means that the service rate $\mu = 1 / 3.12 \approx 0.32$.

The number of agents k can be found in the column Logged On Time: 12:01:02, which means a little more than 24 agents were logged on during this half hour.

Figure 2.7: A typical call center management report (part I)

		Logged on time			Distribution of time				
Date	Half hour	Hours	Mins	Secs	Inbound	Outbound	Wrap-up	Idle	Avail
23-9-2002	8:00:00	1	36	29	16,40%	0,00%	6,30%	0,40%	77,00%
23-9-2002	8:30:00	3	59	58	18,90%	0,00%	7,20%	10,50%	58,30%
23-9-2002	9:00:00	9	46	9	46,30%	0,80%	15,10%	5,70%	27,90%
23-9-2002	9:30:00	11	40	56	43,80%	0,80%	14,90%	13,90%	30,90%
23-9-2002	10:00:00	12	26	19	26,20%	2,40%	12,40%	21,60%	34,50%
23-9-2002	10:30:00	12	29	23	33,40%	2,10%	14,20%	21,00%	30,30%
23-9-2002	11:00:00	12	0	24	32,90%	0,90%	16,10%	16,70%	30,40%
23-9-2002	11:30:00	12	1	2	32,90%	0,50%	12,60%	22,20%	32,90%
23-9-2002	12:00:00	12	30	0	23,40%	1,20%	7,70%	36,60%	34,10%
23-9-2002	12:30:00	12	1	56	20,50%	0,40%	9,60%	45,00%	24,50%
23-9-2002	13:00:00	13	29	22	54,70%	0,50%	19,60%	11,00%	11,00%
23-9-2002	13:30:00	13	24	14	55,30%	1,90%	19,30%	16,00%	9,40%
23-9-2002	14:00:00	13	0	28	40,00%	1,80%	13,20%	23,00%	20,60%
23-9-2002	14:30:00	11	32	51	47,40%	0,60%	13,90%	18,40%	20,10%
23-9-2002	15:00:00	10	58	6	44,10%	2,50%	16,90%	17,90%	19,40%
23-9-2002	15:30:00	9	4	17	45,90%	3,40%	19,80%	21,30%	8,60%
23-9-2002	16:00:00	7	15	22	54,60%	1,90%	19,50%	18,40%	9,80%
23-9-2002	16:30:00	5	49	45	38,40%	4,40%	19,70%	28,50%	5,80%
23-9-2002	17:00:00	3	41	16	44,70%	2,30%	25,20%	23,80%	8,90%
23-9-2002	17:30:00	1	45	21	57,20%	0,00%	14,20%	34,70%	11,80%
23-9-2002	18:00:00	1	29	28	28,90%	6,50%	21,80%	12,90%	17,50%
23-9-2002	18:30:00	1	0	13	35,50%	0,00%	10,70%	39,30%	11,30%
23-9-2002	19:00:00	1	0	0	37,90%	9,80%	30,60%	2,80%	3,30%
Summary		2.796	56	54	32,3%	4,6%	19,3%	22,8%	20,4%

Figure 2.7: A typical call center management report (part II)

Number of calls					Average length in secs				One Touch Ratio	Service Level	Productive Time Ratio
Absolute			Per sign-on hour								
Inbound	Outbound	Abandon	Inbound	Outbound	Inbound	Outbound	Wrap-up	Answer			
10	1	0	6,2	0,6	94,7	236	36,3	7,5	90,00%	100,00%	16,40%
24	3	1	6	0,8	113,1	44,3	43	11,79	83,30%	91,27%	18,90%
116	4	2	11,9	0,4	140,3	163,5	45,7	8,69	88,80%	95,62%	47,00%
112	8	8	9,6	0,7	164,5	168,3	55,8	7,78	84,80%	80,04%	44,60%
95	15	2	7,6	1,2	123,3	151,7	58,6	9,32	83,20%	87,24%	28,60%
102	15	5	8,2	1,2	147,1	122,3	62,5	8,08	84,30%	79,62%	35,50%
103	6	0	8,6	0,5	138	73	67,5	9,05	78,60%	95,68%	33,70%
105	7	0	8,7	0,6	135,5	92,3	51,7	9,05	86,70%	94,87%	33,30%
76	9	1	6,1	0,7	138,6	106,3	45,3	9,21	67,10%	97,05%	24,60%
76	22	2	6,3	1,8	116,8	43,5	54,5	8,05	82,90%	82,25%	20,90%
191	15	3	14,2	1,1	139,1	46,3	49,9	7,41	72,80%	88,87%	55,20%
176	30	4	13,1	2,2	151,5	56,9	53	8,05	68,80%	86,81%	57,10%
146	14	2	11,2	1,1	128,3	141,1	42,3	8,36	71,20%	82,56%	41,80%
150	12	5	13	1	131,5	63	38,6	8,28	71,30%	78,54%	48,00%
115	20	10	10,5	1,8	151,5	124,7	58	9	76,50%	65,58%	46,60%
113	11	11	12,5	1,2	132,6	166,1	57,1	7,96	74,30%	62,35%	49,20%
103	9	6	14,2	1,2	138,6	87,6	49,4	8,71	74,80%	74,34%	56,50%
65	8	5	11,2	1,4	124,1	225,5	63,4	7,15	64,60%	72,61%	42,90%
30	4	2	8,1	1,1	197,7	141,8	111,4	6,93	93,30%	86,21%	46,90%
22	0	0	12,5	0	164,3	0	40,8	5,91	81,80%	95,98%	57,20%
17	3	3	11,4	2	91,4	156,3	68,8	10,88	76,50%	92,56%	35,50%
8	0	0	8	0	160,3	0	48,4	10,38	100,00%	99,01%	35,50%
5	1	1	5	1	273,2	471	220,6	9,8	80,00%	80,00%	47,80%
21.303	**5.520**	**73**	**7,6**	**2**	**152,6**	**160**	**91,3**	**8,68**	**78,10%**	**81,24%**	**36,90%**

3 Why are tables not a feasible data structure?

3.1 What are call center log data?

Call center log data are commonly stored in tables of relational databases. That leads to raw data, which look like Figure 3.1. The rows do not fit on a page, so they are divided into three parts shown underneath each other. Please note that this is a fictitious example, which I have designed based on my experience with various ACDs.

	CallID	SequenceNumb	DateTime	ANI	DNIS	CallType	Disposition	Peripheral
	839753	1	6-10-2002 17:32:01	207774444	8000800	1	29	4001
▶	839752	1	6-10-2002 17:31:45	204447777	8000800	1	12	4001
	839752	2	6-10-2002 17:34:33	0	0	4	21	4001
*		0				0	0	0

DelayTime	QueueTime	Duration	HoldTime	AbandonTime	TalkTime	WrapUpTime	AgentID
5	21	26	0	26	0	0	
12	13	1232	12	0	1200	8	1234
0	0	455	45	0	400	120	1589
0	0	0	0	0	0	0	0

SkillGroupID	ServiceID	TeamID	Variable1	Variable2	Variable3	Variable4
5678	9012	3456	#1#4	1589		
5679	9012	3689			1234	
0	0	0				

Figure 3.1: Call center log data

It is quite difficult to determine what has actually happened here. Row 1 is an abandoned call (Disposition 29 means abandoned) from number 020 777 4444 to 800 0800. The caller abandoned after 26 seconds.

Rows 2 and 3 have the same CallID, which means that they belong together. Disposition 12 (row 2) means transferred, so the primary agent (AgentID 1234) transferred the call to a secondary agent (AgentID 1589) after 1212 seconds (TalkTime + HoldTime). The conversation from then on is summarized in row 3. CallType 4 means internal call and Disposition 21 means normal termination. So after 445 seconds with the secondary agent, the caller hung up.

What I want to illustrate with this example is that the current storage of call center log data is very complicated. If you have never seen call center log data before, you will probably not be able to understand them. Please note that if the rows were sorted by DateTime, as is often the case, the rows belonging together would be separated by the row of the abandoned call.

3.2 What is a data structure?

A data structure is a way to store data. The most common data structure is a relational database consisting of several tables, as shown in the previous paragraph. Other data structures include tree structures and list structures, displayed in Figure 3.2 and Figure 3.3.

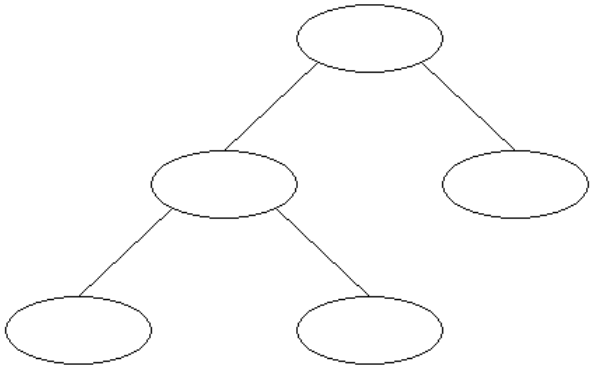


Figure 3.2:Tree structure

Figure 3.3: List structure

The reason why tables are not the best data structure for call center log data is that they do not describe reality very well. Tables are useful when there are several occasions (in rows) on which the quantities to store (in columns) are (almost) identical. If we look at Figure 2.1 through Figure 2.3, we see that there are numerous situations in which we need to store different quantities. For instance, when a customer abandons before reaching an agent, there is no need to store Talk Time or Hold Time.

A tree structure can describe the process better, as will be shown in Chapter 5, because a tree structure is richer than a table structure, which means that it is easy to convert tables to trees, but hard or impossible to convert trees to tables. The tree structure which XML uses is very flexible, which makes it ideal to describe the process.

The reason why tables are used as a data structure is that the technology to distill information from data is mature and reliable, though difficult. The Structured Query Language (SQL) is often used for this purpose. At the moment, tree-structured XML is full-grown and ready to take over to become the standard in data storage. Database manufacturers have XML database management systems (DBMSs) waiting on their shelves for application.

3.3 Why look at log data?

If log data are stored in relational databases, SQL can be used to retrieve information from them. An SQL query does not care if the data are complex: if the query is programmed correctly, the right information is retrieved. So why is it a problem that log data are difficult to understand for people? The answer is that they have to check whether the data are stored correctly and verify the SQL query before they can rely on technology to create reports for them. Moreover, even if data are stored correctly and the SQL is verified, the occasional need to check the log data for a particular call still remains. My personal experience with call centers strongly supports this; it was the reason to write this report in the first place.

4 What is XML?

This chapter shows how XML can improve call center data storage. It is not meant as a complete XML course, but even if you do not fully understand this chapter, you will still be able to see why XML is a better call center log data structure than a set of tables. For a complete XML course, please consult [5] or [6].

4.1 Introduction

XML is short for eXtensible Markup Language. In a markup language the data are between tags. Extensible means that you can invent tags yourself. Consider the following example:

```
<name>Theo</name>
```

The data (`Theo`) are between tags: a start tag (`<name>`) and an end tag (`</name>`). Together they form an element. Elements are structured in a tree structure (see also Figure 4.1):

```
<person>
    <name>
        <first_name>Theo</first_name>
        <last_name>Peek</last_name>
    </name>
    <address>
    …
    </address>
</person>
```

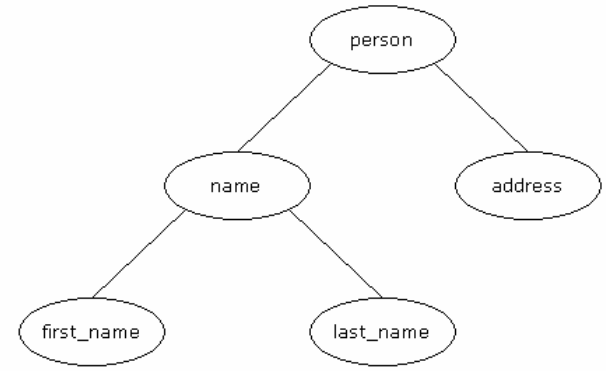


Figure 4.1: The tree structure of XML

`name` is called a child element of `person`. Although you might see a resemblance with HTML, the HyperText Markup Language, which is used to create web pages, these two languages are definitely not the same. Both HTML and XML are sub-versions of SGML (Standard Generalized Markup Language) and both use tags to structure data. However,

- HTML uses a predefined set of tags whereas XML is extensible;
- XML is more strictly structured than HTML. In HTML it is, for instance, possible to omit certain end tags, which is not allowed in XML;
- XML can be validated using a definition file (see Paragraph 4.2.3);
- XML was designed to describe data and to focus on what data are; HTML was designed to display data and to focus on what data look like;
- XML is case sensitive.

The World Wide Web Consortium (W3C) developed XML, starting in 1996, and resulting in the first standard (Version 1.0) in 1998.

4.2 A quick guide

4.2.1 Basic XML

An XML document always starts with the XML declaration:

```
<?xml version="1.0"?>
```

This is immediately followed by the root element. An XML document contains exactly one root element (`address_book`):

```
<?xml version="1.0"?>
<address_book>
…
</address_book>
```

The root element can, like all elements, contain:

- Text (simple content);
- Other elements (element content);
- Nothing (empty content).

Although an element can contain both text and other elements (mixed content), this is not recommended as it is more structured to separate simple content and element content. So:

```
<?xml version="1.0"?>
<address_book>
    <person>
        <name>
            <first_name>Theo</first_name>
            <last_name>Peek</last_name>
        </name>
        <address>
            …
        </address>
        <comments/>
    </person>

    <person>
        <name>
            <first_name>Ger</first_name>
            <last_name>Koole</last_name>
        </name>
    </person>
</address_book>
```

`first_name` and `last_name` are the only elements which contain simple content; the rest contain element content. `<comments/>` is an example of empty content. The slash at the end of the tag should be interpreted as: "this is an empty element, so do not expect an end tag."

4.2.2 Attributes and Namespaces

Attributes can be used to give meta-information, i.e. information about the data. This could be done for instance in the `person` element, like this: `<person gender="male">`. `gender` is the attribute with value `male`. The value of an attribute is always enclosed in double quotes. If the value of an attribute contains double quotes, it can also be enclosed in single quotes. Suppose we want to store the following sentence: John said: "Hello!", then we could store it like this: `<sentence contents='John said: "Hello!"' />`.

Very often, attributes can be replaced by extra elements:

…

```
<person>
    <gender>male</gender>
    <first_name>
…
```

It is recommended to use elements instead of attributes whenever possible, since elements can be better validated and processed.

Suppose we assign an ID number to every person and we define the address by a street and a number. Then we will get a number element twice: once in the context of the person and once in the context of the address. To avoid confusion and potential conflicts, we can use namespaces. Every namespace has its own prefix: for instance, we use the *p* namespace prefix for person elements and the *a* namespace prefix for address elements. This would result in the following example:

```
<?xml version="1.0"?>
<address_book>
    <p:person>
        <p:number>1</p:number>
        <p:name>
            <p:first_name>Theo</p:first_name>
            <p:last_name>Peek</p:last_name>
        </p:name>
        <a:address>
            <a:number>15</a:number>
            <a:street>De Boelelaan</a:street>
        </a:address>
        <p:comments/>
    </p:person>

    <p:person>
        <p:number>2</p:number>
        <p:name>
            <p:first_name>Ger</p:first_name>
            <p:last_name>Koole</p:last_name>
        </p:name>
    </p:person>
</address_book>
```

Now, there can be no misunderstanding about the two `number` elements. The XML file shown above is not yet complete. We should declare the namespaces *a* and *p* (added parts are **bold**):

```
<?xml version="1.0"?>
<address_book>
    <p:person xmlns="http://www.few.vu.nl/~tdpeek/Person">
        <p:name>
            <p:first_name>Theo</p:first_name>
            <p:last_name>Peek</p:last_name>
        </p:name>
        <a:address xmlns="http://www.few.vu.nl/~tdpeek/Address">
            <a:number>15</a:number>
            <a:street>De Boelelaan</a:street>
        </a:address>
        <p:comments/>
    </p:person>

    <p:person>
        <p:number>2</p:number>
        <p:name>
            <p:first_name>Ger</p:first_name>
            <p:last_name>Koole</p:last_name>
        </p:name>
    </p:person>
</address_book>
```

As you can see, we declare a namespace at the point where it is first used. In the xmlns (XML NameSpace) attribute we must declare a URI (Uniform Resource Identifier): a string to identify

the namespace uniquely. It is common to use a URL (Uniform Resource Locator), for they are by definition unique. Please note that there is no need for the URL to exist (in this case, they do not!).

4.2.3 Structuring by DTD or XML Schema

Based on the previous paragraph, you could claim that XML is not structured at all: you can invent tag and structure elements entirely to your own liking. Fortunately, we *can* structure and define our XML document by a Document Type Definition or an XML Schema. Both define which elements can be used and how they should be structured. In that way, we can define for instance:

- `address_book` is the root element, which contains `person` elements;
- `first_name` and `last_name` elements can only exist within the `name` element;
- `name` can only exist within the `person` element;
- A `person` element can also contain an `address` element, but this is not required.

XML Schema was developed later than DTD and therefore DTD is more common at this moment. However, XML Schema is more powerful, so we will use XML Schema to validate our XML document. XML Schemas themselves are written in XML and are also referred to as XSDs (XML Schema Documents). The XSD of our address book example would begin like this:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.few.vu.nl/~tdpeek"
xmlns="http://www.few.vu.nl/~tdpeek"
elementFormDefault="qualified">
```

The first line indicates that the XSD is an XML document. It can also be omitted, but in editors it is useful to include it, for formatting purposes. The second line defines that the *xs* namespace refers to the schema of schemas, defined by the W3C. The third defines the namespace that the schema works on. The fourth line specifies the default namespace of the XSD (in this case, the same as the target namespace). The last line specifies that every element used in the XML document must be declared in the XSD.

We then continue the XSD with the definitions of the elements, starting with `address_book`:

```
<xs:element name="address_book">
    <xs:complexType>
        <xs:sequence>
```

These are three start tags, so we will need end tags for them later on. `element` is defined in the *xs* namespace and has an attribute containing the name of the element. `adress_book` is a complex type, as it contains other elements. The elements contained (all `person` elements) appear in a sequence:

```
<xs:element name="person" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
```

Since all elements are `person` elements, the first element in the sequence is the `person` element. With the `minOccurs` and `maxOccurs` attributes we can control the number of times an element can occur in the sequence. If omitted, these attributes are both assumed to be one. In this case, we wish to store more than one person in our `address_book`, so we increase `maxOccurs`. `person` also contains other elements in a sequence:

```
<xs:element name="name">
    <xs:complexType>
        <xs:sequence>
```

And even `name` contains other elements in a sequence:

```
                                <xs:element name="first_name" type="xs:string" />
                                <xs:element name="last_name" type="xs:string" />
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
```

`first_name` and `last_name` are elements contained in `name`. They both have simple content of the string type. Please note that the slash at the end of the `xs:element` tag specifies that it is an empty element. All its information is in attributes. This is all the information there is about the `name` element, so the elements are closed by end tags.

The next element of `person` is the `address` element:

```
                    <xs:element name="address" type="xs:string" minOccurs="0"/>
```

`address` is also of the string type. Because we sometimes only know the `name` of a `person`, we wish to be able to exclude the `address` element, hence `minOccurs="0"`. Now we end all elements that we started:

```
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Again the XSD, but now nonstop:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.few.vu.nl/~tdpeek"
xmlns="http://www.few.vu.nl/~tdpeek"
elementFormDefault="qualified">

<xs:element name="address_book">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="person" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="name">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="first_name" type="xs:string" />
                                    <xs:element name="last_name" type="xs:string" />
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="address" type="xs:string" minOccurs="0" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

4.3 The future of XML

At the moment, XML is novel and therefore not very commonplace. This will certainly change in the near future, witness the fact Microsoft's .NET technologies rely heavily on XML [8], for instance. Even today, all recent office suites support XML as a file format. All large software companies have XML products ready and waiting for customers, and not without reason, as XML

has a large number of applications. Together with XSL, XML can replace HTML and word processor formats to store documents. Together with XML Schema, XML can replace all database and spreadsheet storage formats. Because of the large area XML can be applied in, XML could well be the first step to a standard, data-independent, platform-independent storage language. As a result, communication between computer programs can become much easier.

On 21st January 2003, the XML Schema Working Group of W3C released the first public Working Draft of Requirements for XML Schema 1.1 [7]. In the near future we will see a gradual replacement of DTDs by XML Schemas. The most important reasons for this are [5]:

- XML Schemas support data types, so:
  - It is easier to describe permissible document content;
  - It is easier to validate the correctness of data;
  - It is easier to work with data from a database;
  - It is easier to define data facets (restrictions on data);
  - It is easier to define data patterns (data formats);
  - It is easier to convert data between different data types;
- XML Schema are written in XML, so:
  - You do not have to learn another language;
  - You can use your XML editor to edit your Schema files;
  - You can use your XML parser to parse your Schema files;

XML is an ideal combination of structure, imposed by an XML Schema or DTD, and flexibility or extensibility to ease the modeling of reality. In the area of web services, XML is already a standard. Web services will expand themselves and simultaneously enlarge the use of and familiarity with XML. It is my hope that this report will inspire call center managers to investigate the possibility of introducing XML into their businesses and reap the fruits of its advantages.

5 How can I use XML to log my calls?

In the previous chapter XML was introduced. In this chapter, I will use XML to log calls.

5.1 Simple examples

Please recall the example of Paragraph 3.1. If we had logged the calls in XML, what would they look like? Let us first look at the abandoned call:

```
<?xml version="1.0"?>
<calls xmlns="http://www.few.vu.nl/~tdpeek" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.few.vu.nl/~tdpeek Call.xsd">
    <call>
        <routingPhase>
            <connect>
                <time>17:32:01</time>
                <ani>0207774444</ani>
                <dnis>08000800</dnis>
            </connect>
            <play>
                <time>17:32:01</time>
            </play>
        </routingPhase>
        <endOfCall>
            <abandon>
                <time>17:32:27</time>
            </abandon>
        </endOfCall>
    </call>
</calls>
```

First, we use several attributes in the root element `calls` to link the XML document with an XSD: `Call.xsd`. The `connect` element matches the Connect state of Figure 2.3. The `play` element matches the "IVR plays welcome message" state. The call ends by abandonment at 5:32:27 p.m. As you can see, we can match the activity states of the UML diagrams of Chapter 2 with XML elements, so we get a very clear description of the process as it happened.

Let us take a look at the second call of Figure 3.1. Please note that the actual times may not correspond, the example is merely to show the events that took place.

```
<?xml version="1.0"?>
<calls xmlns="http://www.few.vu.nl/~tdpeek" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.few.vu.nl/~tdpeek Call.xsd">
    <call>
        <routingPhase>
            <connect>
                <time>17:31:45</time>
                <ani>0204447777</ani>
                <dnis>08000800</dnis>
            </connect>
            ...
```

The customer is calling from number 0204447777 and has dialed 08000800. He or she now hears the menu choices:

```
            ...
            <play>
                <time>17:31:45</time>
            </play>
            <menu>
                <time>17:31:48</time>
                <choice>1</choice>
            </menu>
            <menu>
                <time>17:31:57</time>
```

```
            <choice>4</choice>
        </menu>
        ...
```

After choosing menu choices 1 and 4, the customer is now routed to a skill group:

```
        ...
        <routeToSkillGroup>
            <time>17:31:57</time>
            <service>Reservation</service>
            <skillGroup>English speaking</skillGroup>
            <peripheral>Lucent</peripheral>
            <location>Amsterdam</location>
        </routeToSkillGroup>
        ...
```

To categorize calls, we can assign `services`. `peripheral` is used when there are two or more ACDs and `location` when there are two or more sites. Unfortunately, all agents in the English-speaking skill group are busy, so the call is queued:

```
        ...
        <queue>
            <time>17:31:58</time>
            <queueLength>1</queueLength>
        </queue>
        <routeToAgent>
            <time>17:32:08</time>
            <agent>Theo Peek</agent>
        </routeToAgent>
    </routingPhase>
    ...
```

With every decrement of the queue, a `queue` element is added. This way, queue length and perception can be analyzed. When an agent becomes available, the customer is routed to that agent, which concludes the routing phase. The call now enters the agent phase, which matches Figure 2.2:

```
    ...
    <agentPhase>
        <answer>
            <time>17:32:10</time>
        </answer>
        <agentAction>
            <talk>
                <time>17:32:10</time>
            </talk>
        </agentAction>
        <agentAction>
            <hold>
                <time>17:33:50</time>
            </hold>
        </agentAction>
        ...
```

`talk`, `hold` and `consult` (not shown in this example) are structured as instances of `agentAction`, see Figure 5.1. For technical reasons (most call center telephones work that way), a transfer or consult is always preceded by a hold state.

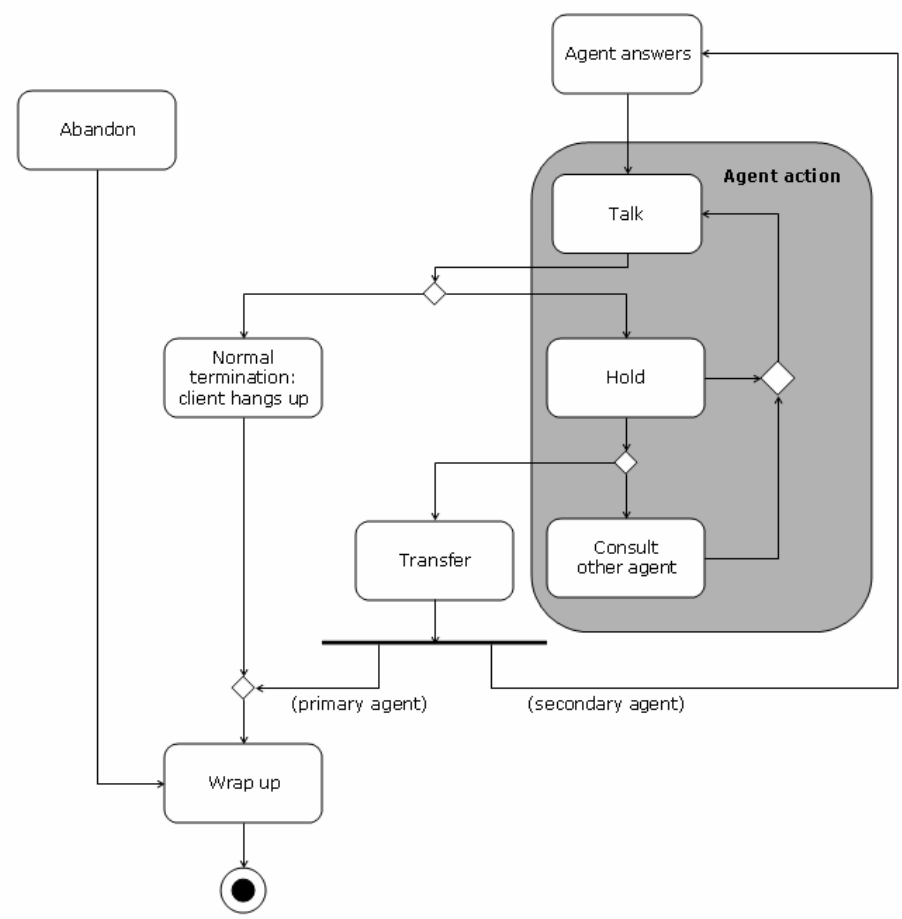


Figure 5.1: The agent phase of the process revisited.

The agent now transfers the call to another agent.

```
...
<transfer>
    <time>17:34:30</time>
    <agent>Ger Koole</agent>
    <nextAgentPhase>
        <answer>
            <time>17:34:33</time>
        </answer>
        <agentAction>
            <talk>
                <time>17:34:33</time>
            </talk>
        </agentAction>
        <wrapUp>
            <time>17:35:00</time>
        </wrapUp>
        <end>
            <time>17:35:45</time>
        </end>
    </nextAgentPhase>
</transfer>
...
```

In the `transfer` element, the element `nextAgentPhase` follows the `time` and `agent` elements. `nextAgentPhase` is equivalent to the `agentPhase` element in which it resides. All data of a call are in one place, as opposed to the table structure, where a call is distributed over two or more records in

case of a transfer. After the end tag of the `transfer` element, data of the initial agent follow: in other words, the `wrapUp` below corresponds with the wrap up of the primary agent:

```
            ...
            <wrapUp>
                <time>17:34:32</time>
            </wrapUp>
            <end>
                <time>17:34:40</time>
            </end>
        </agentPhase>
        ...
```

This concludes the agent data of the call. Now we only need to specify how the call ended (in this case normal termination instead of abandonment) and close all open elements:

```
        ...
        <endOfCall>
            <terminate>
                <time>17:35:45</time>
            </terminate>
        </endOfCall>
    </call>
</calls>
```

5.2 Complete list of events

As we saw in the previous paragraph, XML can be used to clearly describe the process of calls. From Figure 2.1 through Figure 2.3 and Figure 5.1 we convert the action states into XML elements. Some of the elements contain properties, of which `time` is a frequently used example. In the two examples of the previous paragraph, most of the states and their properties have been shown. To complete the model, Table 5.1 shows the complete list of all possible elements and their properties. The table cells of talk, hold and consult are grey, because these elements are grouped together in the agentAction element, as we have seen and will also see later on.

Table 5.1: Complete list of elements and their properties.

Phase	***Event / Element***	***Properties***
routingPhase	connect	time ani dnis
	play	time
	menu	time choice
	routeToSkillGroup	time skillGroup service peripheral location
	queue	time queueLength
	routeToAgent	time agent
agentPhase	answer	time
	talk	time
	hold	time

	consult	time agent
	transfer	time agent
	wrapUp	time
	end	time
endOfCall	terminate	time
	abandon	time
	IVR_only	time

5.3 XML Schema

Now that we have all possible events, we must convert the events into an XML Schema which we can use to validate XML documents. The (tree-like) structure of this XML Schema is shown in Figure 5.2. Please note that this diagram is made using XMLSpy. More information on XMLSpy can be found in Appendix B.1.

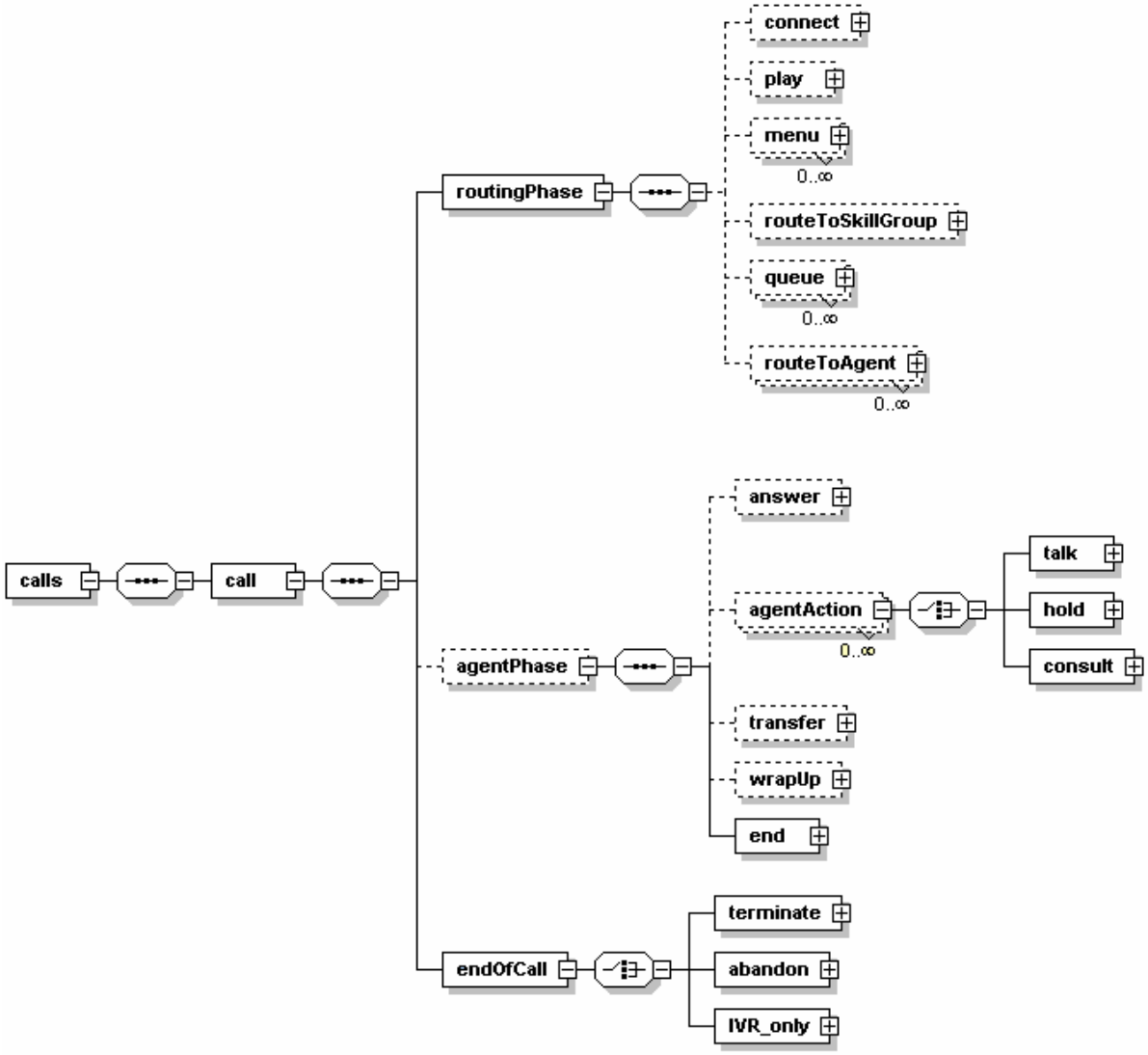


Figure 5.2: Diagram of the XML Schema.

A call consists of a routingPhase, an agentPhase and an endOfCall, which in turn contain the elements listed in Table 5.1. Let us see how things look in XML Schema:

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.few.vu.nl/~tdpeek"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.few.vu.nl/~tdpeek"
elementFormDefault="qualified">
    <xs:element name="calls">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="call">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="routingPhase">
                            ...
```

`calls` contains a sequence of `call` elements. A `call` element starts with the `routingPhase`.

```
                            ...
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="connect" minOccurs="0">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="time" type="xs:time"/>
                                                    <xs:element name="ani" type="xs:string"/>
                                                    <xs:element name="dnis" type="xs:string"/>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="play" type="timeOnlyType"
minOccurs="0"/>
                                        <xs:element name="menu" minOccurs="0"
maxOccurs="unbounded">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="time" type="xs:time"/>
                                                    <xs:element name="choice" type="menuChoice"/>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="routeToSkillGroup" minOccurs="0">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="time" type="xs:time"/>
                                                    <xs:element name="service" type="xs:string"/>
                                                    <xs:element name="skillGroup"
type="xs:string"/>
                                                    <xs:element name="peripheral"
type="xs:string"/>
                                                    <xs:element name="location" type="xs:string"/>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="queue" minOccurs="0"
maxOccurs="unbounded">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="time" type="xs:time"/>
                                                    <xs:element name="queueLength"
type="xs:positiveInteger"/>
                                                </xs:sequence>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="routeToAgent" minOccurs="0"
maxOccurs="unbounded">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="time" type="xs:time"/>
                                                    <xs:element name="agent" type="xs:string"/>
                                                </xs:sequence>
```

```
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    ...
```

The routing phase contains the elements `connect`, `play`, `menu`, `routeToSkillGroup`, `queue` and `routeToAgent`. All these elements have attribute `minOccurs="0"`, because a customer can abandon at any stage. `maxOccurs` is one by default. `Connect` contains simple elements `time`, `dnis` and `ani`. `play` has type `timeOnlyType`, which is defined later on as an element which contains one child element: `time`. `menu` can occur more than once, because customers have to make successive menu choices. `menu` contains simple elements `time` and `choice`. `choice` has type `menuChoice`, which is defined later on as a nonnegative integer of one digit. `routeToSkillGroup` contains simple elements `time`, `service`, `skillGroup`, `peripheral` and `location`. `queue` contains `time` and `queueLength`, the latter being a positive integer. Because `queue` appears each time the queue length decreases, `maxOccurs` is set to unbounded. The last child element of `routingPhase` is `routeToAgent`, which can occur more than once, if an agent does not answer.

`routingPhase` is followed by `agentPhase`:

```
                    ...
                    <xs:element name="agentPhase" type="agentPhaseType" minOccurs="0"/>
                    ...
```

Because an `agentPhase` can also occur as a child element of transfer, we create a custom type: `agentPhaseType`, which is defined later on. After the `agentPhase`, the `endOfCall` follows:

```
                    ...
                    <xs:element name="endOfCall">
                        <xs:complexType>
                            <xs:choice>
                                <xs:element name="terminate" type="timeOnlyType"/>
                                <xs:element name="abandon" type="timeOnlyType"/>
                                <xs:element name="IVR_only" type="timeOnlyType"/>
                            </xs:choice>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
...
```

The `endOfCall` contains either a `terminate`, an `abandon` or an `IVR_only` element, which is why it is defined as a `choice` instead of a `sequence`. This concludes the definition of the root element `calls`. We have used three custom types which must also be defined:

```
...
<xs:simpleType name="menuChoice">
    <xs:restriction base="xs:nonNegativeInteger">
        <xs:totalDigits value="1"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="timeOnlyType">
    <xs:sequence>
        <xs:element name="time" type="xs:time"/>
    </xs:sequence>
</xs:complexType>
...
```

`menuChoice` is a simple type which is a nonnegative integer of one digit, meaning 0, 1, …, 9. `timeOnlyType` is a complex type. Elements of type `timeOnlyType` contain one child element: `time`.

```
    ...
    <xs:complexType name="agentPhaseType">
        <xs:sequence>
            <xs:element name="answer" type="timeOnlyType" minOccurs="0"/>
            <xs:element name="agentAction" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:choice>
                        <xs:element name="talk" type="timeOnlyType"/>
                        <xs:element name="hold" type="timeOnlyType"/>
                        <xs:element name="consult">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="time" type="xs:time"/>
                                    <xs:element name="agent" type="xs:string"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
            <xs:element name="transfer" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="time" type="xs:time"/>
                        <xs:element name="agent" type="xs:string"/>
                        <xs:element name="nextAgentPhase" type="agentPhaseType"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="wrapUp" type="timeOnlyType" minOccurs="0"/>
            <xs:element name="end" type="timeOnlyType"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

The `agentPhaseType` starts with the `answer` element. The `agentAction` element contains either a `talk`, `hold` or `consult` element and can occur more than once. The `transfer` element contains a `time` and an `agent`, but also a `nextAgentPhase` of type `agentPhaseType`, which creates an internal loop of `agentPhases`. This does not cause any problems in practice, because a call will usually not be transferred more than once. The advantage of this structure is that all data of a call are stored in one place, within one `call` element.
This concludes the description of the XML Schema which can be used to validate XML documents.

5.4 Implementation in the business

5.4.1 Hardware

The use of XML requires little more hardware capacity than an ordinary database. With the internal loops of transferred calls, the computer should store a little more information in main memory compared to a database which writes a record more often, resulting in the calls being distributed over several records. Furthermore, because XML is text-based, it takes more space on hard disks than a traditional database. On the other hand, standard compression techniques like ZIP work well on these text-based files. The second XML example of Paragraph 5.1 was compressed by 65% by using the standard Windows program WinZip 8.1. XML Databases include compression and other functionality.

All in all, this should not lead to real problems. The extra requirements are marginal, especially when we consider the current price of memory and hard disk space.

5.4.2 Software

The adaptation of software, on the other hand, requires more investments. Plenty of (freeware) XML tools can be found on the internet, but the installation and overall getting used to new software is always challenging. Nevertheless, it is only a matter of time before XML will become commonplace, as I have shown in Paragraph 4.3.

The use of XML in a call center requires the following:

- Call routing software which logs in XML;
- An XML database to store call logs;
- An XML query tool to extract information from the data.

The first requirement is most difficult to meet. Writing a single database record every time a particular event occurs is easy, but in order to use XML effectively, a sophisticated tree structure has to be formed. The Document Object Model (DOM) is often used for this purpose. DOM is an Application Programming Interface (API), a set of pre-programmed instructions for manipulating XML. It contains standard functions for adding an element, writing an XML structure to disk and so on. In that way, the complete call process is logged in the computer memory before it is written to disk in XML, all using DOM. Using DOM has another advantage: when another program needs XML data, it does not need to wait until they are written to disk, but it can access the data by using DOM as well. More information on DOM can be found in [7]. Also in [7], a list of tools can be found, that might assist you using DOM.

The current state of the market for XML database products (databases and query tools) can be found in [8]. As a compromise, it is possible to convert the currently used tables into XML and test them. The data cannot be as detailed as described before, because data are stored less detailedly in tables, but this does offer the opportunity to test and experience the advantages that XML offers, without losing the proven safety of database technology. Information on products that can convert tables into XML can also be found in [9].

5.4.3 Performance

In a very large call center, it can sometimes take time to extract information from data, because of the huge number of records a query has to process. When using tables, performance can be upgraded using summary tables containing half hour data (for instance number of calls, average length for that specific half hour interval). With XML a similar solution could work, creating an XML document containing summary data. Even in this case, XML offers greater flexibility than fixed tables.

6 Conclusion

After presenting the challenges call center managers are faced with (Chapter 2) and the problems traditional data structures and information systems present (Chapter 3), I have introduced XML (Chapter 4) as an alternative log data structure (Chapter 5).

My conclusion is that XML offers advantages over tables and that call center managers should seriously consider investing in this technology, based on the following facts:

- Tables are rigid and therefore by definition not suitable to model a process as sophisticated as a call. XML is much more flexible and I have created an XML Schema that closely matches the process of a call I have modeled, proving XML can approximate reality far better than tables.
- XML is on its way to become the standard in data interchange between computer programs. It is already the standard in the area of web services and it becomes more and more accepted in other areas as well. Call center log data are typically data that need to be processed by computer programs in order to extract information from them and they are therefore a perfect application area for XML.

7 Suggestions for further research

7.1 RDF

Resource Description Framework (RDF) is a knowledge modeling framework with an XML syntax. The idea behind RDF is that the element name carries information as well. Using XML, it is logical to use `country` as the name for the element `<country>The Netherlands</country>`. But if we were to call that element `soup` and define it correctly in the XML Schema, the XML document would still be valid: XML does not ensure logic. RDF is a way to use Artificial Intelligence (AI) to ensure this logic in data storage. More on RDF can be found in [7] and [11].

7.2 XML tools

A full description of the XML tools available is beyond the scope of this report. There are numerous ways to create management reports like the one presented in Figure 2.7 from the XML data shown in Paragraph 5.1. Finding out which tools are most practical for the mathematical analysis of call center data could be the subject of another BWI-report. My XML Schema from Paragraph 5.3 and [9] could function as a starting point.

7.3 Agent log

In this research report, I have focused on call log data. To calculate some numbers that are shown in Figure 2.7, we need the telephone log of the agent: when did he or she log on, when did he or she take a break, for instance. This is relatively easy to achieve, using the knowledge of Chapters 4 and 5.

7.4 Relativity

In a relational database, primary and foreign keys are used to create relations. When a call is answered by an agent, for instance, a foreign key is stored instead of the name of the agent. The foreign key is a number, which can also be found in another table, which contains all agents and their numbers (primary keys).

In my opinion, using foreign keys in a call log is not necessary; the ACD can find the name of the agent in a configuration table or XML document and include the name directly. This makes the log clearer. For this reason, I have not extensively researched the relational possibilities XML offers. However, such a technique is available, using the `ID` and `IDREF` attributes. More information about these attributes can be found in [5] and [12].

Appendix A: Bibliography

[1] **Noah Gans, Ger Koole and Avi Mandelbaum**: Telephone Call Centers a Tutorial and Literature Review. Downloadable from http://www.cs.vu.nl/obp/callcenters.

[2] **Ger Koole**: Call center mathematics. Downloadable from http://www.cs.vu.nl/~koole/ccmath.

[3] **Martin Fowler with Kendall Scott**: UML Distilled, Second Edition. Addison Wesley Longman Inc., 2000.
Information about Activity Diagrams can be found in Chapter 9, pages 129-139.

[4] **G.M. Koole and A. Mandelbaum**: Queueing models of call centers: An introduction. Annals of Operations Research, 113:41-59, 2002.

[5] http://www.w3schools.com.
A site with an online XML course.

[6] **William R. Stanek**: XML Pocket Consultant. Microsoft Press, 2002.

[7] http://www.w3.org.

[8] http://www.microsoft.net/net/basics/whatis.asp.

[9] http://www.rpbourret.com/xml/XMLDatabaseProds.htm.

[10] **Suzanne de Gruijter, Joost van Hooff, Hugo Huijser, Shaireen Niamat, Theo Peek, Herman Stofbergen and Tjebbe Witteveen**: Roosteren in het Fortis Contact Center. A dutch report on call center scheduling.

[11] http://www.xml.com/pub/a/2001/01/24/rdf.html.

[12] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/ssxsdannotations_8y9f.asp.

Appendix B: XML software

B.1 XMLSpy

Although XML documents can be created with a simple editor like Notepad, an editor with validation functionality is very practical. XMLSpy is a sophisticated XML editor. Users can write XML documents in code, assisted by helpful tools like a list of possible elements and attributes. One can also switch to a visual mode, in which one can easily add a child element in a very user-friendly way. When using an XML Schema or a DTD, XMLSpy validates an XML document automatically. Figure B.1 shows a screenshot of the program, editing the call example of Paragraph 5.1. On the left side is a window containing information about the current element. In the middle is the XML itself and on the right there are windows containing a list of elements, a list of attributes and a list of entities.

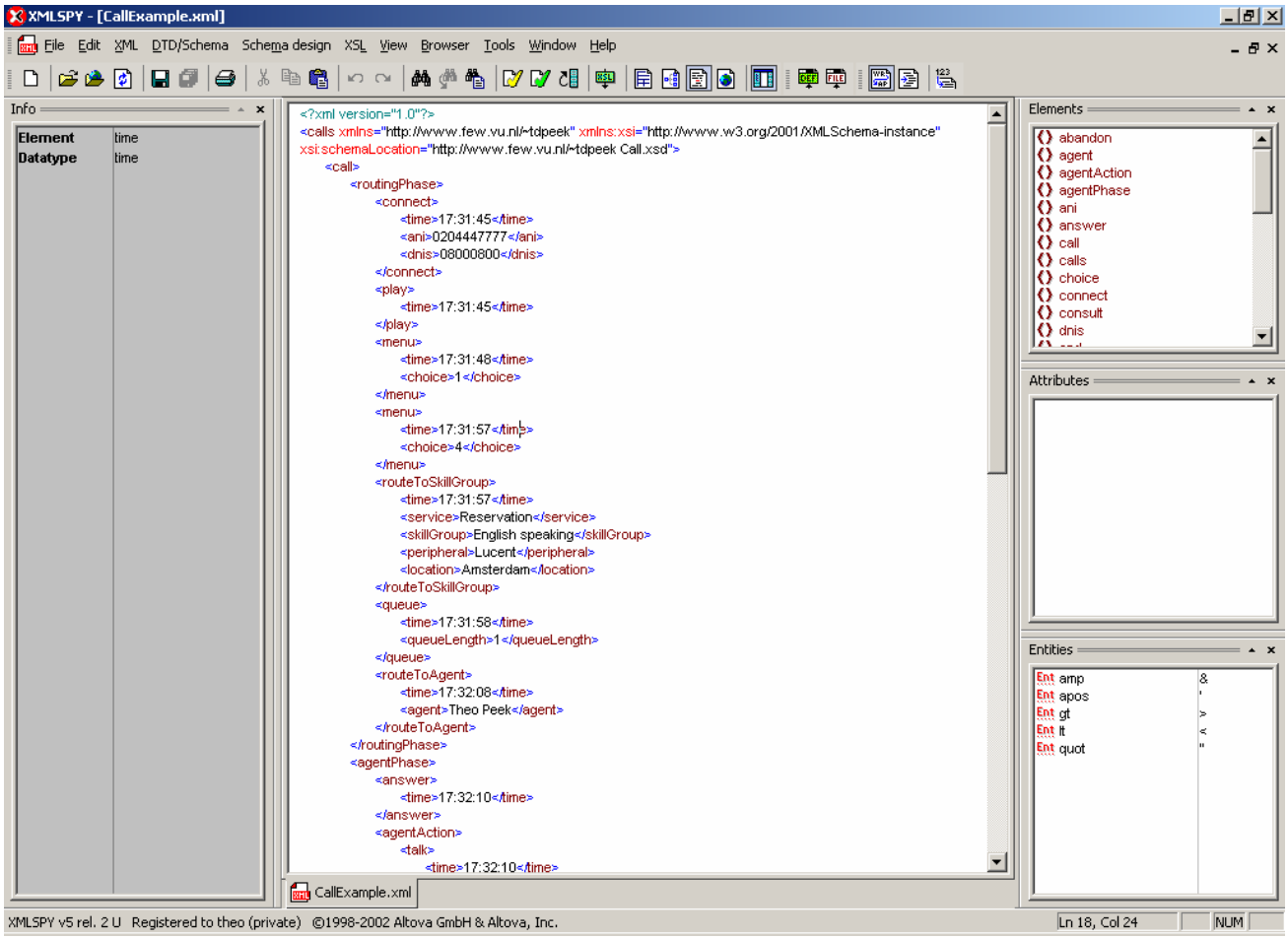


Figure B.1: XMLSpy

When we select 'Validate', XMLSpy validates our XML document, see Figure B.2.

Figure B.2: XMLSpy has validated our XML document.

Switching to 'Enhanced grid view', we can examine our XML document visually, see Figure B.3.

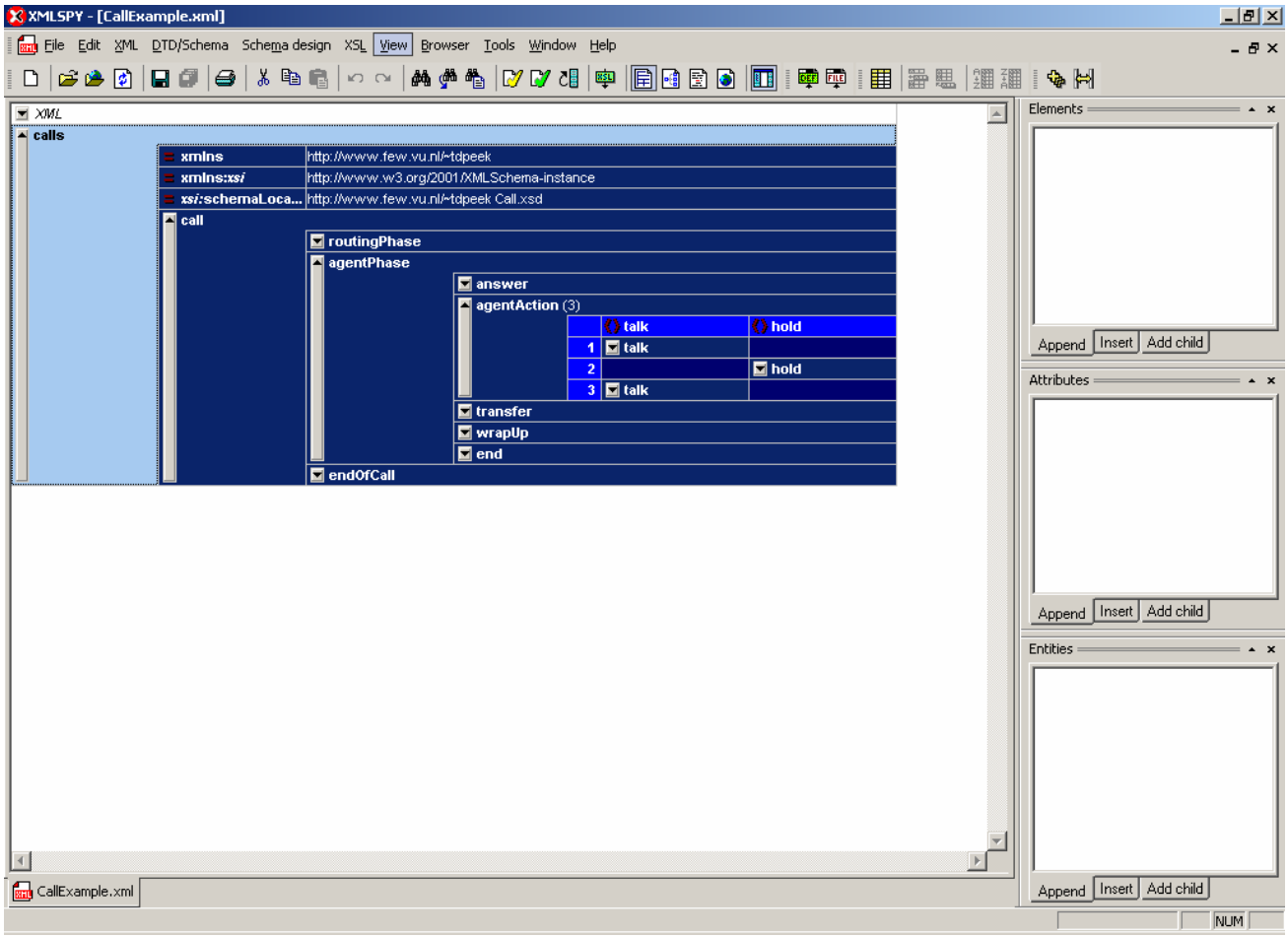


Figure B.3: Enhanced grid view

Figure 5.2 was made with the 'Schema/WSDL Design View'.

XMLSpy can be downloaded from http://www.altova.com. Version 5 Home Edition can be evaluated for 30 days.