00-03-23

Auth: Andreas Oderstad

Call Traffic Monitoring System

- for use with a billing system in a directory inquiry service

Abstract

This Master's Thesis is written as a result of my degree project. The project is part of a larger project aimed to develop a billing system for use with a directory inquiry service. The degree project consisted of developing the call traffic monitoring system in the billing system.

The directory inquiry service takes use of a Private Branch Exchange, PBX, that has a facility called Call Information Log, CIL, which can be used to register information about calls e.g. calling number, called number, duration, type of call etc. This information is to be used as a basis for billing of the directory inquiry services. However, this information cannot be directly used to invoice customers, rather it can only be used for deducing information about details in the performed services. Thus, processing of the information obtained through CIL is required to make use of it in the billing system.

In addition, information related to a service performed during a directory inquiry call can be generated elsewhere in the directory inquiry system. To make the invoices easier to read and to simplify the procedure of crediting services or parts of services, it's necessary to relate this external information to the information obtained from CIL and coordinate the processing. Hence this will require communication with the distributed clients that generates the external information in the directory inquiry system to retrieve and coordinate the information.

During the degree project an application for handling the above-described tasks was developed, tested and integrated into the directory inquiry billing system. Additionally, an Alarm Application was developed to inform personal of abnormalities in the billing systems operation, to enable avoidance and fast recovering from failures.

1 Preface

This Master's Thesis is a result of my Degree Project performed at INEO during the summer and fall 1999. It will conclude my Master of Science in Electronic Engineering at the Royal Institute of Stockholm (KTH).

2 Table of contents

3 Introduction

There were two companies involved in the work of this degree project. A consultant company called *INEO Konsult* employed me, while the degree project was part of a consultant assignment assigned to INEO by *Ahhaaa*. As indicated, the degree project was a part of the larger project corresponding to the consultant assignment. The project group consisted of three consultants from INEO, including myself. The project group leader, Björn Sjöstrand, was also my company advisor.

The degree project part of the project was not distinctively separated from the larger project. Instead it showed that much of the work in the degree project included analysis and design issues related to the overall project. In this thesis I will describe some parts of this analysis and design issues to better give an understanding for the background and the environment of degree project, but will keep it on a high level since the main part of the thesis will focus on the actual development of the call traffic monitoring system.

3.1 INEO Konsult AB

INEO Konsult AB is a consultant company located in Stockholm. INEO provides complete solutions for the Computer Telephony and Messaging markets. INEO has developed systems for call distribution, fax-on-demand, fax mass-distribution, local exchange simulation, conferencing and IP telephony. The customers are large companies and organisations e.g. Ericsson, Swedish Post, Europolitan, Svenska Handelsbanken, CAP Gemini, the Swedish Ministry of Foreign Affairs, and Ahhaaa. INEO is mainly working with software development tools and standard software from Microsoft. The solutions are developed for Microsoft Windows NT, Microsoft SQL Server, Microsoft Exchange Server, and Microsoft Office. The software development is directed towards object-oriented, using C++, Visual Basic and Java to develop reusable software components.

3.2 Ahhaaa

Ahhaaa is a company providing different types of information services. The idea is to be able to provide customers with any information they like, preventing them from being forced searching for information in a decentralized manner. Hitherto the information services provided by Ahhaaa are directory inquiry service and telephony number/address matching. Mostly the directory inquiry service is offered to the customers through telephony operator companies, rather than to their customers directly (4.1).

4 Background

As mentioned above, Ahhaaa provides a directory inquiry service, which is the number one competitor to Telia's directory inquiry. Hitherto, Ahhaaa has only been able to offer a simple service where the customer may obtain the telephone number to someone, whilst Telia could offer more services as for example to redirect the customer directly to the number that he asked for.

To give the customers a better service, a new set of functions was to be implemented into the directory inquiry service system. This introduced the need for a more complex billing model than the one used before. As a consequence, the old billing system used by the directory inquiry service had to be replaced with a new, more sophisticated billing system, to handle the more complex billing model.

4.1 The old billing system

The old billing system, or rather the non-existing billing system, had its limitations. Ahhaaa doesn't handle the billing of their end-customers directly, but rather indirectly through the telephony operators. Until now it has not been any need for a separate billing system to handle the directory inquiry billing. It has sufficed to use the same billing system and billing model used by "ordinary" telephone calls, only with a special directory inquiry fee. Thus the billing system used were the telephony operators billing systems rather than Ahhaaa's.

All calls to Ahhaaa's directory inquiry were registered as directory inquiry calls by the telephony operator. The special directory inquiry fee was applied and details printed on the bill from the telephony operator to the end-customer. Ahhaaa in turn registered the incoming call traffic and invoiced the telephony operator of the corresponding amount.

4.2 The old billing model

The lack of billing system (4.1) resulted in limitations of the billing model that could be used. Since directory inquiry calls were handled the same way as ordinary telephone calls, the billing model had to be the same as for ordinary telephone calls. That is, each call was debited with a total fee based on:

- Start fee: a one-time fee debited for each call.
- Tick fee: a constant fee added to the total fee every tick during a call. A tick is defined as some time-period or periodical event. Often seconds or minutes of a call are used as ticks.

The total fee was then calculated as: Total fee = Start fee + #Ticks * Tick fee.

Different types of directory inquiry services i.e. *fixed-price directory inquiry* or *normal directory inquiry* (continuous minute-based price) could be handled by adjusting the Start fee and the Tick fee respectively to the appropriate values. But each type of service needed to be fixed and mapped to a separate telephone number with the consequence that each type of service had its own telephone number.

4.3 New functions

To offer the customer a better service, several new functions were implemented into the directory inquiry service. Below is a description of the new functions in the directory inquiry. Some of these functions can be offered in combination with others whilst some cannot (4.4.2).

4.3.1 Test Call

When calling the directory inquiry, it should be possible for the broker to make a call to test if a telephone number is correct and working. This is useful when a customer calls back, complaining about a telephone number earlier obtained through the directory inquiry. The broker can make a test call and then if the telephone number is indeed incorrect, the broker can credit the customer (4.3.5).

4.3.2 Redirect

When calling the directory inquiry to obtain an unknown telephone number, it is possible to be redirected to the telephone number asked for. Redirection is debited with a fixed fee and when redirected, the continuous fee is close to the continuous fee of an ordinary telephone call with the

currently used operator, though there is a small margin added to the fee since the call is using resources in Ahhaaa's PBX[1].

4.3.3 Three-party call

A three-party call can be useful when a customer needs to be redirected to the international directory inquiry, but is in need of a translator. Then the Swedish broker can offer to translate during the conversation between the customer and the international broker. It's possible for only a part of the call to be a three-party call and the remaining part to be a normal directory inquiry call. The three-party part of the call is debited with a special three-party-call-fee consisting of a start fee and a continuous fee.

4.3.4 IM-text, GSM SMS messages

When calling the directory inquiry from a cellular phone (GSM), it's possible to get the telephone number asked for, sent as a GSM SMS text message directly to the phone. This function is debited with a fixed fee.

4.3.5 Crediting

When services fail e.g. a telephone number is incorrect or an IM-text doesn't reach the customer, it is possible to credit the customer. When crediting a fixed-price directory inquiry call or an IM-text, the corresponding fee of that service is credited. When crediting a normal directory inquiry call with a continuous fee, all parts of the call that is to be credited have to be specified. That is, the percentage of the total call time that should be credited (since the call is debited with a continuous fee), the number of IM-text messages to credit and any other chargeable item of the service (i.e. article, see 4.4.1) should be specified.

4.4 New billing model

The new billing model is designed to support accurate billing of the directory inquiry services, as well as of other informational services provided by Ahhaaa in the future. It establishes a more general concept of information as a commodity, not directly associated with information obtained through the directory inquiry. The idea is to not limit the billing model, but to keep the possibility of using it for other informational services open as well. The directory inquiry service will be able to use other interfaces as web or GSM SMS without changing the billing model.

This is accomplished by keeping the basic concepts of the billing model on a relatively high level of abstraction and using general terms that are not directly associated with the directory inquiry service or telephone calls. The following sections will introduce the concepts of the new billing model.

4.4.1 Article

An *article* is a piece of information or a single service acting as an isolated chargeable item. Articles constitute the smallest quanta of chargeable items in the billing system. Each article is charged using a model similar to the one described in section 4.2, i.e. consists of a start fee and a tick fee. The start fee and tick fee are based on current *rate* (4.4.6). Examples of articles are *incoming telephony* (customer talking to a broker), *IM-text*, *redirect*, *three-party* and *directory inquiry* (as in fixed-price directory inquiry).

4.4.2 Service

A *service* is a collection of one or more articles. It consists of the articles that together form what the customer is experiencing as a service. It can be defined as a pattern of articles in a parameterized way where each article may appear either a fixed number of times or arbitrary many. That makes it possible to define two types of *services*:

1. *Fixed services* consisting of fixed contents.
2. *Variable services* consisting of variable contents.

---

[1] A redirected call uses both an incoming line and an outgoing line in the PBX. These resources are used during the full length of the redirected call.

Note that even though a fixed service consists of fixed contents, the contents may have variable fees, implying that even fixed services can result in a variable total fee.

Examples of services are fixed-price directory inquiry (fixed service) which may be defined as consisting of only one directory inquiry article and normal directory inquiry (variable service) consisting of one telephony article, arbitrary many IM-text articles and possibly a redirect article.

4.4.3 Daytype

Daytypes are used to categorize days. They are based on the daytypes in the calendar i.e. weekday, weekend, holiday, day before holiday etc.

4.4.4 Timetype

Timetypes are defined to form a fine-grained base for defining rates (4.4.6). This is needed because resolution in days for the rates often is not accurate enough. Timetypes are defined based on time intervals and daytypes, e.g. morning/evening-hours defined as weekdays 0-8, 18-24 and office-hours defined as weekdays 8-18.

4.4.5 Articlefee

An *articlefee* is the fee that an article (4.4.1) charges the customer. The articlefee corresponds to the actual cost for the customer to have the article performed. The articlefee can vary between articles and timetypes (4.4.4) e.g. different articles may have different fees and an article may have different fees corresponding to different timetypes.

4.4.6 Rates

Rates are in fact discount rates applied to timetypes (4.4.4). It's based on operators and volume[2] of articles. A rate can apply to one or more timetypes and for each timetype, there must be a rate defined for every article (4.4.1). Using the dependency between articles, their articlefees (4.4.5) and the current rate, its possible to determine the net fee of an instantiation of a specific article.

4.4.7 Operator

Operators are the telephony operators used by the customers when calling the directory inquiry e.g. Telia, Tele2, Telenordia etc. Rates (4.4.6) are based on the operator used, i.e. different operators have different rates.

[2] Volume of articles is based on the number of articles performed or the total article duration, depending on if the article fee is fixed or tick-based.

4.4.8 Summary of the new billing model

Figure 1 is showing the dependencies between the different concepts described in the previous sections when determining the service fee. The arrows represent the dependency relationship between two concepts.

The timetype is determined by the daytype and the time of day. The articlefee is then determined by the timetype together with the type of article. The article type, volume, timetype and operator then determine which rate to be used. Further on, the net fee for an instantiated article is based on the articlefee and the rate. Finally the total fee of an instantiated service is determined by adding the net fees of all articles in that service.

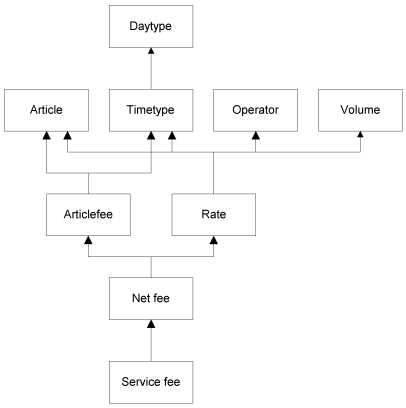

Figure 1 *Shows the dependencies between the different concepts in determining the service fee.*

5 Billing system

5.1 Call Detail Record, CDR

A call detail record, CDR, is a record structure describing events during and details about a telephone call. CDR's are used by most telephony operating companies in their billing systems. They are generated in the public exchanges used by the telephony operators. Different operators use different types and brands of public exchanges, and since no global standard of the format of CDR's exists, they generate different formats.

A CDR normally contains at least information about calling party telephone number (A-number), called telephone number (B-number), time and duration. Often it contains a lot more information. The telephony operating companies has developed billing systems that can use CDR's as input to automatically handle invoicing of their customers and collecting statistic data.

5.2 Basic concepts of the billing system

Ahhaaa does not invoice their end-customers directly, rather indirectly through the telephony operators as described in previous section 4.1. That way there's no need for Ahhaaa to keep records of or creating contracts with their end-customers. The telephony operators become resellers of Ahhaaa's products and services. This is a convenient model since creating customer records and contracts require lots of administrative work.

Due to the concepts that the new billing model introduces, it's necessary register all details about articles performed in a service. This information is not detectable by the telephony operators, since they operate outside Ahhaaa's system. Generating CDR's from the information about services and their articles, in the format used by respectively telephony operator, makes it possible for the operator to use these in their billing systems directly, by only making slight changes to their systems. Thereby it's still possible for the telephony operators to handle the billing of the end-customer, without knowing about the details of the Ahhaaa services.

The design of the billing model is flexible in a way that makes it easy to add articles and services in the future. By designing the billing system in a way that it can receive information about articles and services generated from different parts of the "service system", it will always be possible to add new functions and services to the "service system" and handle the billing of these using the same billing system and billing model.

5.2.1 Transaction

Transactions are the entities in the transaction database (5.2.4) corresponding to instances of a service (4.4.2). To report, make or create a transaction is the action of initializing a transaction and storing transaction data (5.2.2) belonging to that transaction in the database.

5.2.2 Transaction data

Transaction data is the description of a transaction (5.2.1). This includes data describing articles (4.4.1) instantiated in the transaction. When referring to transaction data later on in this thesis, it refers to both complete transactions and parts of transactions.

5.2.3 Article generators

Article generators are the parts in the billing system that generates transaction data (5.2.2) describing articles (4.4.1). They don't actually have to perform the articles themselves, but they are the parts designated to monitor articles that are being performed. An article generator monitors specific types of articles and each type of article is monitored only by one article generator.

The article generators report their transaction data either directly to the *transaction database* (5.2.4) or to another process with the task to coordinate transaction data from several article generators. To coordinate transaction data from several article generators makes it possible to keep articles related to the same service but monitored by different article generators, bundled together. This in turn simplifies further processing in the billing system when generating CDR's and invoicing, e.g. the articles

performed in a service can be listed on the bill to the end-customer to clarify the total charge, services can be credited afterwards more easily since all articles are bundled together.

5.2.4 Transaction database

The transaction database is where all the transactions are stored. The transactions are identified by unique transaction id's generated by the article generators when reporting the transactions.

5.3 Overview of the billing system

On a high level of abstraction matching the high level basic concepts of the billing system, it can be seen as a data flow between producers and consumers. The producers consist of service generators and article generators that produce transactions. The transactions are then stored temporary in the transaction database before the consumer consisting of the CDR generator and the invoicing application consumes the transactions. This is illustrated in Figure 2.

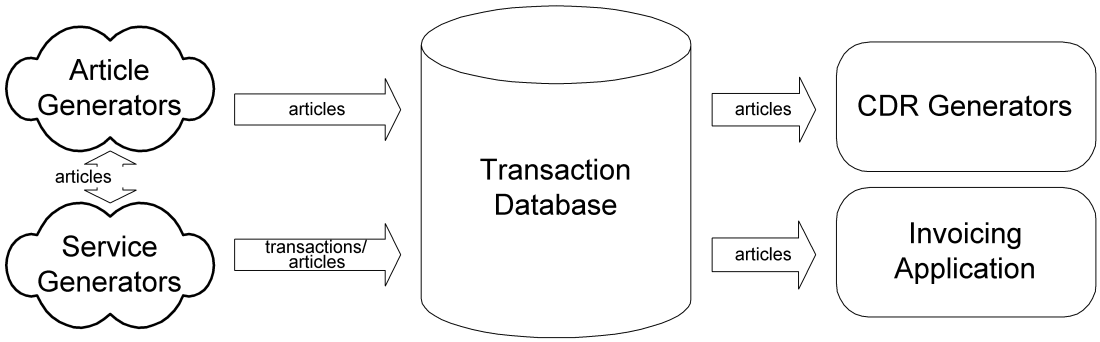

Figure 2

The clouds representing article and service generators illustrates the imprecise definition of an article respectively a service generator. The double directed arrow between the article and the service generators represents a possible article data flow between them, although such flow might not be necessary or might only be in one direction.

The context of the billing system is shown in Figure 3. Ahhaaa's PBX is connected several *telephony operators* PABX's. The *brokers stations* i.e. the broker telephones bundled together with the client applications used by the brokers are connected to the PBX via the private branches and Application Link (5.3.1.1) respectively. The billing system is connected both directly to the PBX via CIL (5.3.1.2) and to the broker stations (client applications). Output from the billing system is the resulting CDR's and bills.

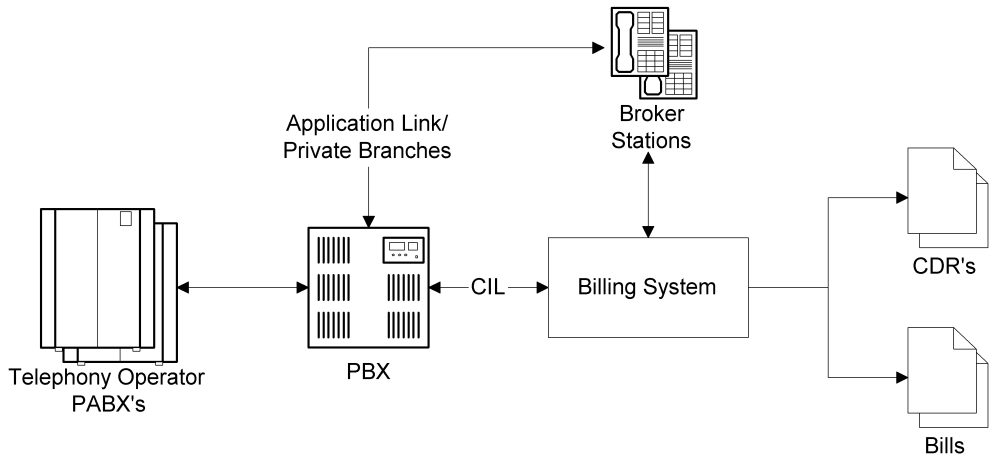

Figure 3

In Figure 4 the billing system is illustrated in more physical detail than in Figure 2. Here the broker client acts as an article generator and the Traffic Log Unit, TLU, is a service generator. This is how the system looks today, with only one article generator and one service generator.

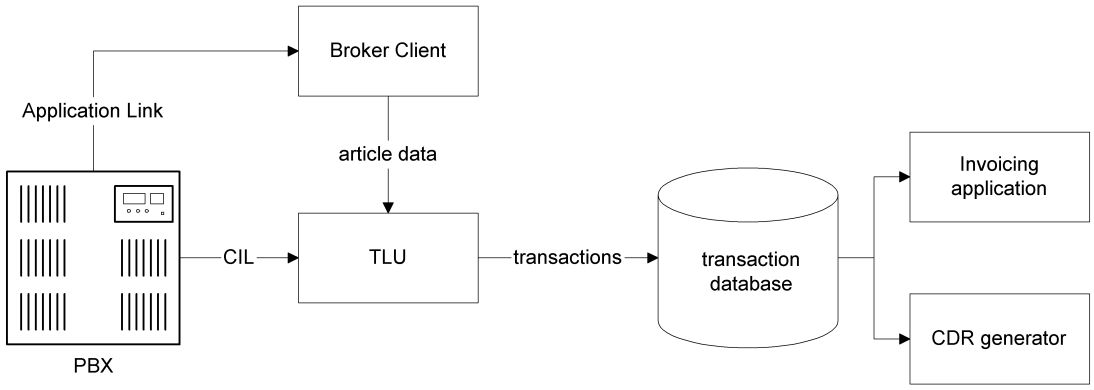

Figure 4

The TLU collects data from the PBX and the broker clients and generates transactions in the transaction database. These transactions are then used by the CDR generator the generate CDR's and in the invoicing application used for handling the invoicing of the telephony operators.

5.3.1 Call data sources

Data describing calls are generated in the PBX. This data has to be obtained using one of the call data interfaces of the PBX. There are two interfaces available for this use, Application Link and Call Information Log.

5.3.1.1 Application Link

Application Link as an application from Ericsson enabling remote control of digital telephones connected to the PBX (an Ericsson MD110 BC8). Using it you can subscribe on events from the connections such as hook-events and connection status events. It's also possible to query connection status information such as calling party telephone number (A-number), called number (B-number), incoming group id etc.

It's possible to retrieve much of the necessary call data from Application Link. However, information about redirected calls cannot be monitored using Application Link, thus Application Link cannot be the sole call data source for the billing system.

5.3.1.2 Call Information Log

Call Information Log, CIL, is a facility enabling logging of call data records. The call data is logged when the call, or an isolated part of a call, has ended. The information is configurable and can include calling party number (A-number), called number (B-number), call duration, call ending time/date, extension line id, status code and so on. Information not directly related to private branches e.g. duration and ending time of redirected calls, is also recorded by CIL. Read more about CIL in section 7.

5.3.2 Choosing call data source

Application Link is already in use by the service system and would be a convenient choice to use as call data source in the billing system. The broker client applications use Application Link for remote control of the broker telephones and to retrieve and display A-number etc. The broker clients will have to act as article generators anyway (5.3.4), so the task would be almost done by letting them use Application Link to retrieve all necessary call data and report it to the transaction database.

Since not all necessary call data is obtained through Application Link, additional call data need to be obtained using CIL. However, CIL can supply all necessary call data by itself, thus it's possible to use only CIL as the call data source.

There are more reasons to use CIL as call data source instead of Application Link:

- Experience in using Application Link has shown it to be unreliable in its operation. Sporadically it stops working and has to be restarted.
- Some call data obtained using Application Link is approximated data, resulting from time measurements (call duration, call ending time).

These weaknesses in Application Link results in the conclusion that CIL is the best choice for call data source in the billing system.

5.3.3 Introduction of the Traffic Log Unit

The task to monitor CIL, analyze the CIL records and extract call data to report to the transaction database falls on the Traffic Log Unit, the TLU. This application has to translate the call records output by CIL into transactions and articles that will be used for invoicing and to generate CDR's. In addition, the TLU has to receive transaction data from other article generators (5.2.3) in the billing system that generates articles belonging to transactions initialized by the TLU. An example of this is IM-text articles generated by the broker client applications.

5.3.4 Broker client application

The brokers use a client application when performing services. These client applications thereby gain knowledge about certain articles performed in services, unknown by any other part of the system, e.g. IM-text articles and credit articles. Hence, these client applications are article generators in the billing system.

The articles generated by the broker client applications always belong to transactions initialized by the TLU. Therefore the broker client applications have to send the corresponding transaction data to the TLU for it to be able to report these articles using the correct transaction id.

5.3.5 Groups and stations

The concepts explained in this section are needed to fully understand the how the analyzing algorithm works. However, the notion of ACD groups is not used in the analysis, but only explains the relationship between the different station identifiers.

In the PBX, there are Automatic Call Distribution groups defined. Each telephone number that's connected to the directory inquiry has one or more ACD group assigned to it. Each ACD group has a queue in which it places all incoming calls in wait for a vacant broker station. Stations are assigned to the ACD groups by its so-called ADN, Additional Directory Number. Every station may have zero or more ADN's defined. This enables each station to be assigned to several ACD groups, because each ADN can only be assigned to one ACD group. Each station also has its ODN, Own Directory Number, which is unique in the PBX and uniquely identifies each station.

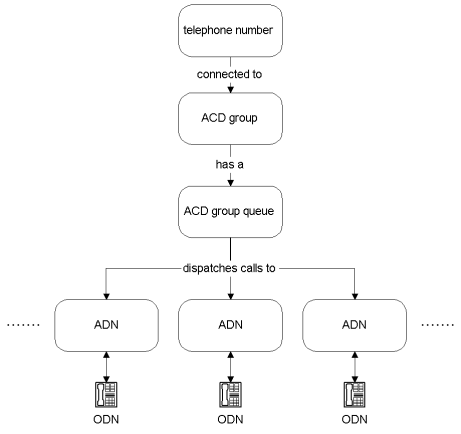

Figure 5

There is another type of groups defined, called Operator Groups, OG. The telephony operators present these as the called number for all incoming calls rather than the actual number called by the customer. The Operator Group identifies which telephony operator is used for the incoming call and eliminates the need for different telephone numbers for each telephony operator. It's necessary to know which telephony operator each call belongs to, to be able to generate correct CDR's.

6 Task definition and expected result

The task of the degree project is to analyze, design, implement, test, and integrate the Traffic Log Unit, TLU in the billing system. The work includes gaining enough knowledge about peripheral equipment to be able to design a robust system that meet the needs and performance requirements. It also includes documenting system maintenance and instructions of the TLU.

Additionally, analysis, design, implementation, test and integration of an alarm application are also included in the degree project. This is not the main part though.

The expected result is a working billing system where the parts included in the degree project are integrated. This system will be used as a part of the commercial system used by Ahhaaa.

7 Call Information Log, CIL

This section will describe CIL, CIL records and CIL operation, which is necessary to understand the design concepts of the TLU in section 8.2.5. Facts about configuration and operation of CIL are based on documents from Ericsson ([8], [9], [10], [11] and [12]) and on experience from experimenting with CIL during the development.

7.1 CIL records

CIL records are records generated by CIL that consists of call data describing the call traffic through the PBX. The CIL record format can be changed to suit different purposes. A CIL record is built of a number of call data fields containing information about ended calls, or ended parts of calls. There are several predefined formats that can be used by CIL, but it's also possible to define custom formats built out of a custom choice of call data fields. In use together with the billing system, CIL is configured with a custom format consisting of only relevant fields to minimize the size of the CIL records. This is to maximize the possible CIL output rate, which in turn minimizes the risk of losses as described in Appendix I. The call data fields used are:

Name	Description	Size/bytes
calling number	The telephone number of the calling part, A-number	20
called number	The telephone number called by the caller, B-number, or Operator Group number of incoming call.	20
call duration	The duration of the call in second resolution, format is HMMSS, wraps after 10 h.	5
status code	A two-character code describing the type of call.	2
date	The date of termination of the call, month and day.	4
time	The time of termination of the call in minute resolution.	4
line id	The ADN of a station or trunk line id of outgoing line[3].	6

This results in a CIL record with a size of 61 bytes. Additionally 3 bytes in overhead in blanks between output CIL records are added. Thus, the resulting CIL record is 64 bytes when output from the serial interface.

The CIL records are constructed according to the following pattern:

calling no	called no	call duration	status code	date	time	line id
20	20	5	2	4	4	6

The 3 bytes overhead are appended to the tail of the CIL record.

7.2 Types of CIL records

There are many different types of CIL records, but only eight different types are relevant in this application. These are the types that are handled by the TLU. Any other type of CIL record, i.e. any undefined type, is just ignored by the TLU. The eight types are defined as:

1. Incoming Call
2. Part of Incoming Call
3. Internal Redirect
4. Part of Internal Redirect
5. External Redirect
6. Part of External Redirect
7. Conference Call

[3] Depending on the type of CIL record as described in 7.2.

8. Internal Call

The fields in the CIL record are filled in with different data for different types of CIL records, with different data meaning different in our viewpoint (the TLU's viewpoint). So can for example the calling number field contain either the calling number (of course) or the ODN, depending on the type of CIL record. Note that these different data (calling number and ODN) is only different from the TLU's viewpoint, but the same from the PBX's. In the following sections when describing the different types of CIL records, the call data is denoted as follows, independent of in which field it appears:

- A-no: the A-number i.e. the telephone number of the caller
- C-no: the C-number i.e. the telephone number of the third party to which the call is redirected[4]
- OG-no: the Operator Group number identifying current operator (5.3.5)
- ODN: the ODN of the broker station
- ODN (2): the ODN of the second broker station in case of redirection to a second broker
- dur: the duration
- date: the date
- time: the time
- ID: the line id of the PBX line used for outgoing call

Other data is explicitly denoted in the fields.

7.2.1 Incoming Call – type 1

An incoming call CIL record is generated by an incoming call that is answered by a broker. This type of CIL record differs from other CIL records generated by incoming calls by that it has the status code "NI". This denotes that the number in the called number field (the second field) is different from the number of the station that answered the call. This is true for all incoming directory inquiry calls since the actual called number is replaced by a so-called Operator Group number before the call is directed to Ahhaaa's PBX (see 5.3.5).

The Incoming Call type has the following fields:

A-no	OG-no	dur	NI	date	time	ADN

7.2.2 Part of Incoming Call – type 2

Long incoming calls generates a partial CIL record each time the duration counter wraps around, i.e. after periods of 10 hours. These CIL records have the following structure:

A-no	OG-no	95959	DI	date	time	ADN

This CIL record is the same as incoming call except for the status code and the fact that the duration always is 95959.

7.2.3 Internal Redirect – type 3

If an incoming call is redirected internally, i.e. redirected to another broker, a separate CIL record is generated for the redirected part of that call. This type of CIL record has the following structure:

A-no	ODN (2)	dur	T	date	time	-

[4] Normally, the number called is the B-number. However, in the case of redirections, the broker is seen as the B-party and the call is redirected to the C-party, thus it is the C-number dialled to connect to the C-party.

The called number field contains the ODN identifying the second station (the (2) after ODN indicates that it's the second station referred to), whilst the line id field is blank. Note that there is no identification of the first station, which introduces problems (see 8.3.6).

7.2.4 Part of Internal Redirect – type 4

If the redirected part of an internally redirected call is long, a partial CIL record is generated each time the duration counter wraps around, similar to the case of a long incoming call. The generated CIL record has the following structure:

A-no	ODN (2)	95959	D5	date	time	-

This is similar to internal redirect except for the status code and the fact that the duration always is 95959.

7.2.5 External Redirect – type 5

Calls that are redirected to an external number generate a CIL record for the redirected part of the call, as in the case of internal redirect. The fields differ somewhat though:

ODN	C-no	dur	T	date	time	ID

The calling number field contains the ODN that identifies the station from which the call was redirected, whilst the called number field contains the number to which the call was redirected, the C-number. The line id field contains the id of the external trunk line used, which is not used.

7.2.6 Part of External Redirect – type 6

As in all other cases of long calls, when an external redirected call is long, CIL records are generated when the duration wraps around at 10 hours. This kind of CIL record looks like this:

ODN	C-no	95959	D5	Date	time	ID

Similar to the external redirect CIL record, except for the status code, and the duration, which always is 95959.

7.2.7 Conference Call – type 7

A conference call CIL record is generated when a station participating in a conference call terminates its call. This kind of CIL record is relevant because three-party calls are in fact conference calls with three parties.

ODN	C-no	dur	L	date	time	ID

This type of CIL record looks the same as external redirect CIL records do, except for the status code.

7.2.8 Internal Call – type 8

Internal calls are calls between stations directly connected to the PBX. When a broker calls another broker e.g. to redirect a customer to the other broker, this type of CIL record is generated:

ODN	ODN (2)	dur	J	date	time	-

Here, both ODN of first station and ODN of second station are registered.

7.3 CIL operation

As mentioned above, CIL generates records holding call data when calls are ended. But this is not totally accurate. Rather, CIL records are generated when the PBX releases resources[5] that were

[5] Resources in a PBX is for example lines, trunks etc.

allocated during a call [7]. In the simplest case of an incoming call that is on-going for a while and then disconnected by both parties, the resources are released when the parties disconnects, thus the CIL record is generated when the call is ended. But if an incoming call is redirected, some resources may be released at the moment of redirect. In this case a CIL record is generated containing call data related to the first part of the call. Then another CIL record is generated when the redirected part of the call is ended, holding call data related to this part. To make correct assumptions about this call it's necessary to consider both CIL records since they are causally related.

There are many types of CIL records of which only eight are defined and relevant in this application. Additionally, the same type of CIL record may appear in many different situations and in combination with other types of CIL records. But not all scenarios are permitted in the directory inquiry system, and the PBX is also used for other purposes than the directory inquiry. Therefore it's necessary to define all scenarios that are permitted and investigate the behavior of CIL in those particular cases. Any unpredicted and undefined scenario will be assumed to be irrelevant and ignored by the TLU.

The following sections will describe all relevant scenarios that should be handled by the TLU, and how the CIL records are generated in these scenarios. The scenarios are:

- Incoming call
- Incoming call is externally redirected
- Incoming call is internally redirected
- Incoming call is both internally and externally redirected
- Three-party call

All results in the scenarios are based on tests. The course of events will be describes by a timeline using the following symbols:

symbol	denotation
A	Calling party, A-party, the customer calling directory inquiry
B	Called party, B-party, the broker answering the incoming call
C	Third party, C-party, the party to whom a call is redirected
D	Fourth party, D-party, the party to whom a call is redirected after C in a series of redirections
N<y>	A new call to y is initialised on an available access line, y is one of C and D
R	The call is redirected to next party. This ends the call for the party performing R.
3	Three-party call is created.
CRx Tv	CIL record is generated and output, x is a sequential numbering The type of the CIL record is v (1-8)
d<CRx>	Duration of CIL record CRx, where x is the sequential numbering of the CIL record
y - z	y is connected to z, y and z is one of A,B,C and D
y answ	y answers the call, y is one of A,B,C and D
y ends	y ends the call, y is one of A,B,C and D

7.3.1 Incoming call

Scenario: Incoming call is answered by broker, is going on for a while and then ended by both parties.

This will generate a single CIL record when the call ends. The call ends when either the broker or the calling party ends the call. Duration will measure from when the broker answers the call until the call is ended as shown in Figure 6.

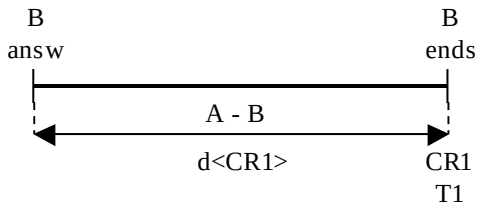

Figure 6

7.3.2 Incoming call is externally redirected

Scenario: Incoming call is answered by broker, it is on-going for a while, and then redirected to an external number.

This scenario has two outcomes depending on whether the call is redirected before or after the C-party answer. Normally when redirecting calls, the call is redirected before the C-party answer, since the broker will not wait for the call to be answered but rather letting the customer wait for an answer. This is called “redirect before answer”. However, sometimes it may be convenient for the broker to be able to wait for an answer, to be sure that the call was in fact answered, and then redirect. Or it may happen that the C-party answered the call so soon after redirection, that the broker didn’t have time to end the call before answer. This is called “redirect after answer”.

In the case of redirect before answer, an incoming call CIL record is generated when C answers the call. The duration in this CIL record spans from B-answer to C-answer. Then an external redirect CIL record is generated when either A or C ends the call, which ends the call altogether. The duration in the second CIL record spans from C answer to the call ending. This is shown in Figure 7.

If the broker redirects the call before answer and the customer never gets an answer, i.e. C answer never occurs, the call is reverted to a normal incoming call (7.3.1) which is ended when the customer, i.e. the calling party gives up and ends the call.

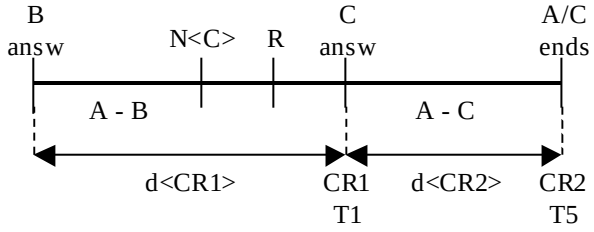

Figure 7

In the case of redirect after answer, no CIL record is generated when C answers the call. Instead, two CIL records are generated at the call ending, one of type incoming call and one of type external redirect. Theses CIL records are *concurrent*, since they are generated *concurrently*. Now the duration in the incoming call CIL record spans from the B-answer the call ending, i.e. spans over the whole call. The duration in the external redirect CIL record spans from C-answer to call ending as in the previous case (redirect before answer). This is shown in Figure 8. Note, in this case B and C are connected for a while, before the call is redirected. The duration of B – C (B connected to C) is included in the external redirect CIL record.

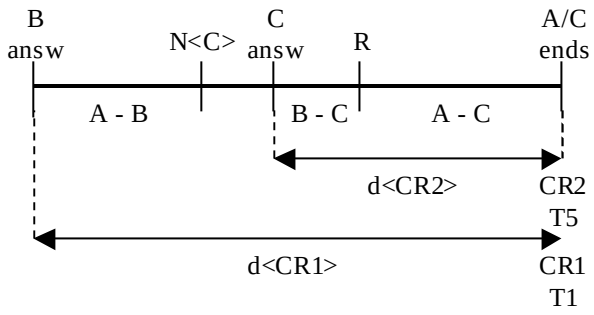

Figure 8

7.3.3 Incoming call is internally redirected

Scenario: Incoming call is answered by broker, it is on-going for a while, then it is redirected to another broker, where it's on-going for a while and then ended.

This scenario may arise when a conversation with a customer has been going on for a long while and the broker for some reason cannot continue the conversation. Then the broker may redirect the call to another broker that will continue the call. The scenario may also arise if a broker client stops working. The call can in that case be redirected to another station where there's a working broker client.

As in the case of external redirect, it's possible to perform "redirect before answer" or "redirect after answer". But in the internal case, the two CIL records corresponding to the incoming respectively redirected parts of the call are generated equally in both cases, that is, generated similar to the case of "redirect before answer" in the external redirect case. However, only "redirect after answer" can be permitted due to trace ability reasons described in section 8.3.6 since this case generates one additional CIL record needed for the trace-ability. For completeness, both cases will be shown here, even though the first cannot be permitted.

Figure 9 shows the scenario when "redirect before answer" is used. The first CIL record is generated when C answers, and its duration spans from B-answer to C-answer. The second CIL record is generated when the redirected call is ended.

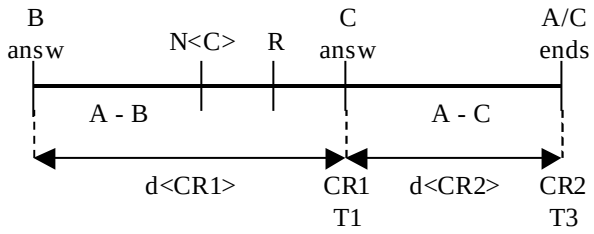

Figure 9

In Figure 10 the scenario, when using "redirect after answer", is shown. Now there are three CIL records generated, the two CIL records corresponding to the incoming and the redirected part of the call respectively, and one additional CIL record corresponding to the part of the call when the brokers are talking to each other (B connected to C). This additional CIL record is of the internal call type and is concurrent to the incoming call CIL record corresponding to the incoming part of the call.

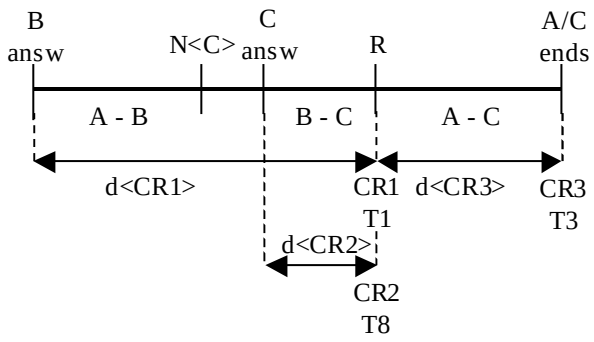

Figure 10

7.3.4 Incoming call is both internally and externally redirected

Scenario: Incoming call is answered by broker, it is going on for a while, then it's redirected to another broker and is going on still a while before it's redirected to an external number, going on for a while and finally ended.

There's nothing new about this scenario, it's only a combination of an internally redirected call and an externally redirected call. The same requirement on the internally redirected part of this call still exists as in the simple "internal redirect" case, which is that redirect must take place after C-answer, thus B and C are connected for a short while. The externally redirect may use either redirect before or after answer as in the simple "external redirect" case. The only difference is that instead of an incoming call CIL record, there will be a internal redirect CIL record involved in the externally redirected part of the call.

In Figure 11 the scenario using "redirect before answer" is illustrated, and in Figure 12 "redirect after answer" is used.

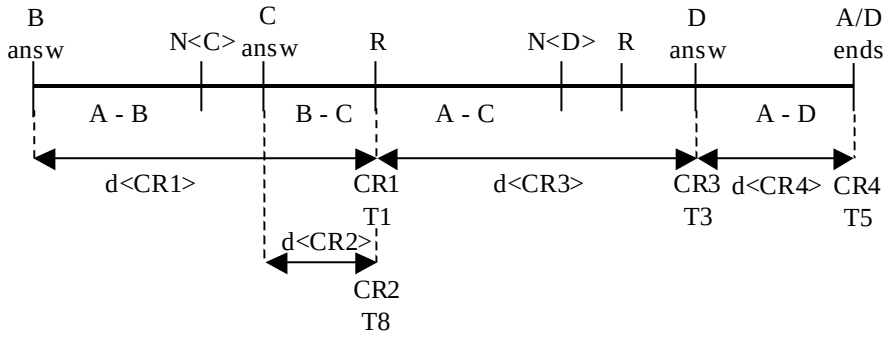

Figure 11

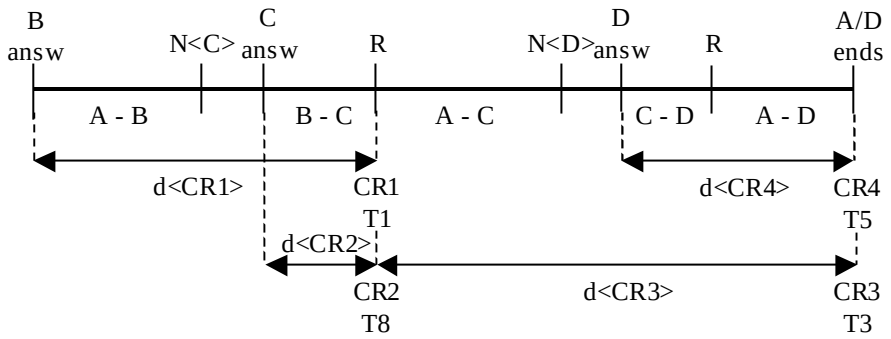

Figure 12

7.3.5 Three-party call

Scenario: Broker answers the incoming call, conversation is going on for a while, then a new call is initialized to a third party and a three-party call is created. The three-party call is going on for a while and is then ended by all parts.

This scenario will have two different outcomes depending on the order in which the three parties end the call. A three-party call is to the PBX the same as a conference call. All conference calls have a leader, and in the case of a three-party call, the broker becomes the leader, since it's the broker that initiates the conference. In the case of three parties in a conference call, if the leader ends the call, the remaining two parties will be connected together in the same way as when a call is redirected. That is, if the broker ends the call, the three-party call will revert to a redirected call. The behavior will be equal to when call is "redirected after answer", thus no CIL record will be generated until the call is ended.

The fact that the three-party call reverts to a redirected call, makes it impossible to determine whether a call that looks like a redirected call, is actually a redirected call and not a reverted three-party call. This is handled by defining *three-party-call* telephone numbers. Every redirected call, or seemingly redirected call is verified by determining if the C-party telephone number is defined as a three-party-call telephone number or not. If it is, the call is assumed to be a three-party call, otherwise a redirected call.

The scenario is illustrated in Figure 13. It shows that there is no possible way to separate the part of the call when all three parties are connected from the part of the call when only A and C are connected, as a redirected call. There are only two CIL records generated in exactly the same manner as in the scenario with an external redirected call.

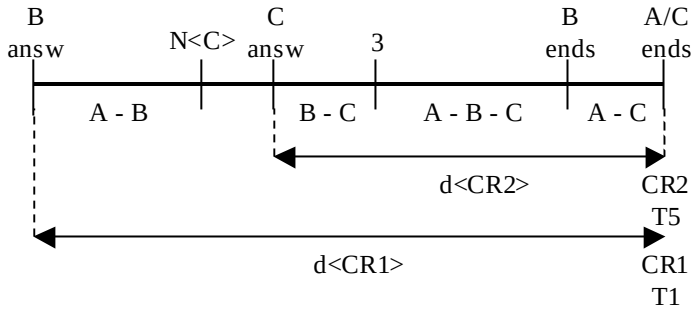

Figure 13

The scenario becomes totally different if either the A-party or the C-party ends its call before the B-party, i.e. the broker. The case when conversation between broker and customer is continuing after the third party has ended is illustrated in Figure 14. During the three-party call, C is ending its call and a conference CIL record, conference call, is generated. The duration in this CIL record spans from C-party answer to C-party call ending. When the remaining conversation between A-party and broker ends, an incoming call CIL record is generated with duration spanning over the total call length.

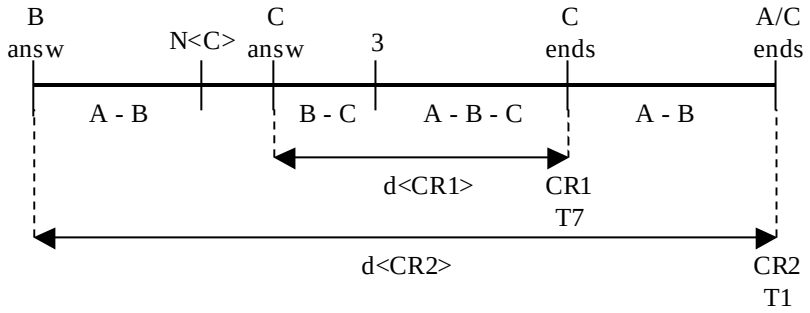

Figure 14

In Figure 15 the other way around is illustrated, the A-party ends the call and the B and C-parties continues the conversation. Now the incoming call CIL record is generated first, when the A-party ends the call, and the conference CIL record is generated when the remaining conversation is ended. This

creates overlapping durations as illustrated in Figure 15 that will force limitations upon the duration of the conversation between A and B (8.3.7).

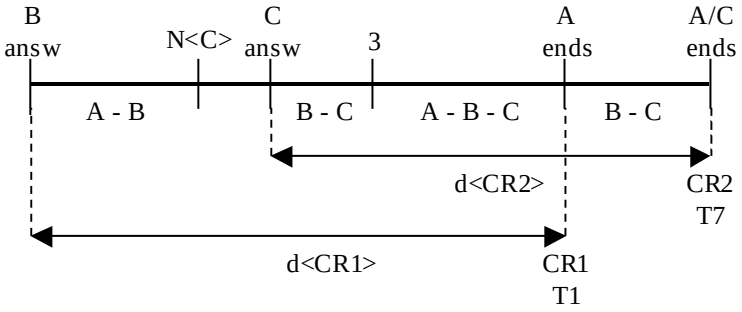

Figure 15

7.3.6 Further scenarios

The above-described scenarios can be combined into new scenarios in many possible ways. For example can an internally redirected call be internally redirected several times before ended or externally redirected, or even made into a three-party call. The appearance of these combined scenarios is simply the same as the appearance of the simple scenarios joined together. Therefore the handling of these combined scenarios doesn't have to be specialized, but can be handled in the same way as several simple scenarios following each other.

7.3.7 Long duration calls

The CIL record duration field counter wraps around every ten hours. When this happens, partial CIL records are output (types 2, 4, and 6, see 7.2). Then the following CIL record starts from the last partial CIL record output. That way there's no limitation on how long call that can be represented by the CIL records, since partial CIL records can be generated indefinitely.

8 Traffic Log Unit, TLU

8.1 Specification of TLU

During the analysis and design of the billing system, properties, functionality and interfaces of parts in the system were defined and fixed. This divides the system into independent parts in such way that:

1. Work on different parts of the system is independent, which allows different people to work on different parts independently of each other.
2. The system becomes modularized. Thus it's easy to modify or update any module in any way, as long as the required properties and the interface are the same.
3. A part of the system can be tested separately without requiring that all the other parts with which it interacts are finished. They can be simulated using simulation modules.

This resulted in the following specification of the TLU:

- Monitor all calls in the PBX without losses, using Call Information Log, CIL.
- Use the CCITT V.24 interface on a SIU board to receive CIL data from the PBX.
- Filter the CIL data for relevant call data.
- From the filtered CIL data, extract information, analyse and generate transactions in the transaction database.
- Generate unique transaction ids for each new transaction.
- Receive articles from other article generators and merge these into the corresponding transactions.
- Generate information to make transactions traceable back to originating CIL data.
- Handle database connection errors without losing call data, by buffering.
- Guard against losing call data in case of computer failure.
- The TLU must support at least 60 000 incoming calls per day, distributed in a typical incoming call distribution of today.
- The TLU will operate on a Windows NT 4.0 Server in a Windows NT 4.0 based LAN architecture.
- The TLU will run as a service under Windows NT 4.0.

8.2 Requirements for the TLU design

The specification consists of specific requirements for different properties of the TLU. To produce a design that fulfils these requirements, properties of other environmental entities e.g. the SIU board serial interface, the CIL data contents and the database connection, has to be investigated. Only with enough knowledge about the environment and its properties, it's possible to know how to design the TLU.

The different requirements for the TLU design can be categorized into the following general categories:

- Performance
- Robustness
- Persistency
- Redundancy
- Trace-ability

The following sub sections discuss issues concerning properties of the environment and what demands these properties make on the design of the TLU, separated in these general categories. The discussion

addresses both requirements stated in the specification of the TLU and requirements that follow of these requirements.

8.2.1 Performance requirements

The performance of the TLU in terms of the traffic load that can be handled must be high enough to avoid the TLU from becoming a bottleneck in the system. There are two general performance requirements:

1. The average call data throughput of the TLU must be equal to, or greater than the average call data output from CIL. In this case the amount of call data is considered as the amount of information or calls rather than number of bytes.
2. The TLU has to be ready to receive CIL data from the serial interface frequently enough to avoid the internal buffer on the SIU board from overflowing (Appendix I).

These requirements are explained below.

8.2.1.1 Retrieving CIL records

A critical part of the operation of the TLU is when receiving CIL data from the serial interface. This is due to the fact that there is a limited buffer space in the serial interface, and when that buffer space becomes full, CIL records are dropped. This is not acceptable since the primary goal is to monitor all call traffic without losses as specified in the specification. Therefore the TLU should be designed in a way that it will be able to receive data from the serial port as frequently as possible, to minimize the risk of overflowing the buffer and dropping CIL records.

In Appendix I there's an investigation of the performance limitations when using the serial interface on SIU board as output of CIL records.

8.2.1.2 Throughput of the TLU

It's necessary that the call data throughput of the TLU is not less than the call data output from CIL is in average. This assumes that the TLU is using internal buffers to compensate for temporary higher call data output from CIL. The reason is obvious since if the throughput of the TLU is lower than the output from CIL in average, the internal buffers in the TLU will become overflowed over time. As specified, the required average throughput must be corresponding to at least 60 000 calls per day.

8.2.2 Robustness

The TLU will have to frequently use database connections to retrieve information and store information from and into databases. In case of database connection failures, the TLU must not fail in its operation. The response to a database connection failure should be different in two different situations:

1. Communication with a database is not crucial for the success of the current operation. In this case the communication with the database should be avoided and the operation continued.
2. Communication with a database is crucial for the success of the current operation. In this case the operation should block until the database connection is available again.

Some operations use data generated by other parts of the system and stored in the database. If this data is needed frequently it's possible to cache data locally to improve performance and robustness of these operations and refresh the data once in a while. If a database connection fails when trying to refresh data in cache, the operation can either postpone the refreshing and use the old cashed data, or block until the database connection is available again.

8.2.3 Persistency

Persistency deals with keeping transaction data persistent no matter what problems arise. When the transaction data is reported to the transaction database, it is stored persistently in the database, or at least is the database's responsibility. But before the transaction data is reported, its responsibility lies on the TLU.

Using primary memory for buffering call data and transaction data makes it vulnerable for any kind of failure implying process termination. To guard against this, all data could be stored on persistent media. However, when manipulating data during processing and analysis, it's better to store the data in

primary memory for reasons like performance, flexibility and simplicity. The set of data currently processed is limited though. Thus, it would be desirable to store all data that is either waiting to be processed or have already been processed, persistently.

8.2.4 Redundancy

Redundancy of the TLU operation could be achieved by using two TLU's running on different machines. If the primary TLU fails, the secondary TLU could take over the operation. There are several difficulties in putting this into practice. The TLU will keep an internal state based on the history of many events. This state has to be either replicated to or shared with the secondary TLU to be able to achieve total redundancy. There are also problems related to transaction uniqueness and knowing which transactions has already been reported, to avoid reporting transactions either none or multiple times.

At this state of the development of the billing system, the redundancy requirements of the TLU operation has been removed due to the complexity in achieving satisfactory redundancy and to the uncertainty of its need.

8.2.5 Trace ability

To be able to trace errors, all transactions in the database should be traceable back to the originating data using the transaction id. Therefore logs should be created in a way that makes it possible to determine which CIL record originates from every transaction. Also the original CIL output should be stored to enable verifying the trace data using original CIL data.

8.3 Analyzing CIL records

8.3.1 Introduction to the analysis

The analysis needed to create transactions from the call data in the CIL records is not as easy as just copying and pasting call data from the CIL record into transactions. If several CIL records are generated from the same call (7.3), they are related and to make the correct assumptions of what has happened during the call, it's necessary to combine call data from all CIL records originating from that same call. The problem is that there's no unique identifier that identifies the call from which the CIL records originate, thus there's no easy way to identify which CIL records originate from the same call. As a consequence it will be necessary to use timestamps together with station identifiers (ODN and ADN, see section 5.3.5) to determine if two CIL records originates from the same call.

Another issue of the analysis is that when the first CIL record originating from a call is generated, there's sometimes no way to know if there will be additional CIL records generated from the same call or not. In ignorance of this, it's necessary to process the CIL record already generated, as if there won't be any more CIL records generated from this call. Then by storing the call data from this CIL record in so-called *call records*, before discarding the processed data, possibly later generated CIL records from the same call may be processed by using the call data from earlier ones.

The above described properties forces the analysis to be made per CIL record basis rather than per call basis, i.e. each CIL record that arrives is processed individually according to current circumstances. Each type of CIL record is handled according to a predefined workflow that is designed to cover all scenarios handled by the TLU in which this type of CIL record can participate.

8.3.2 Determining CIL record type

The first thing to do when analyzing a CIL record is to identify the type of the CIL record. If the type cannot be determined, the CIL record is marked as irrelevant because of invalid type and ignored by the continuing analysis without even examining the contents. The computation to determine the type of a CIL record is illustrated in Figure 16. The most determining factor of the CIL record type is the status code. As can be seen in Figure 16, types 1, 2 and 8 are determined only by the status code field in the CIL record. When it comes to redirected calls and three-party calls, it's necessary to check if there's an external line allocated or not, i.e. if the line id field is blank (internal case) or not (external case), to be able to separate internal cases from external cases.

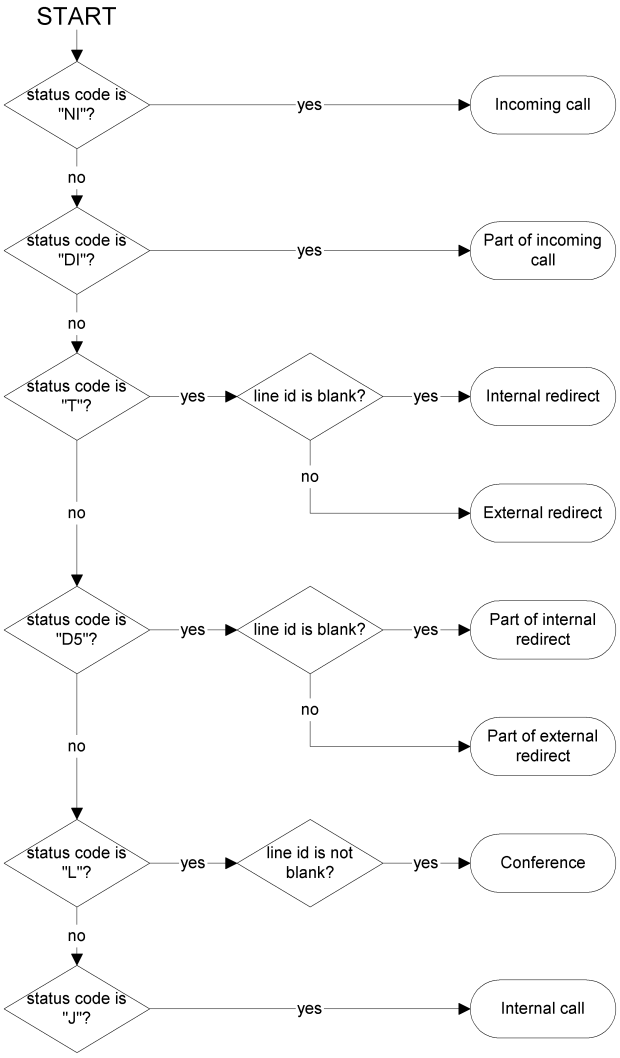

Figure 16

If the algorithm gets stuck due to the lack of one of the arrows “yes” or “no” in a decision and the algorithm cannot proceed, the type of that CIL record is considered to be invalid. The CIL record will be marked as irrelevant because of invalid type.

8.3.3 Related CIL records

Related CIL records are CIL records that originate from the same call or chain of calls (redirections) that relate to the same customer at a given occasion. The order in which CIL records are generated by different scenarios is described in section 7.3. In the simplest of the scenarios there’s only one CIL record generated of type Incoming Call (7.2.1). But in every other scenario, there are several CIL records generated. All these CIL records are related but they carry different pieces of the call data describing the complete call.

As an example, the scenario when an incoming call is redirected to an external number before answer, there are two CIL records generated. First there’s a CIL record of type incoming call generated when the C-party answers the call. This record has call data that identifies the incoming OG-number and the A-number of the customer. Then when the redirected call is ended a second CIL record of type external redirect is generated. This CIL record contains information about the duration of the redirected call and the external number to which the call was redirected. But to be able to bill the customer correctly it’s necessary to have information about the A-number, i.e. the customers telephone number, and the OG-number of that call. This information cannot be found in the second CIL record, but in the first. That’s why it’s necessary to be able to relate the second CIL record to the first and obtain the missing information from the first.

Since there are no unique call identifiers that can be compared between CIL records to find out if they are related, timestamps has to be used. Every CIL record is marked with a timestamp at its arrival. The time difference between arrivals of the first and the second CIL record is in this case equal to the duration of the redirected call. Therefore it's easy to relate the two CIL records by subtracting the duration of the redirected call from the arrival time of the second CIL record and find a CIL record with a timestamp equal to the result. This is illustrated in Figure 17.

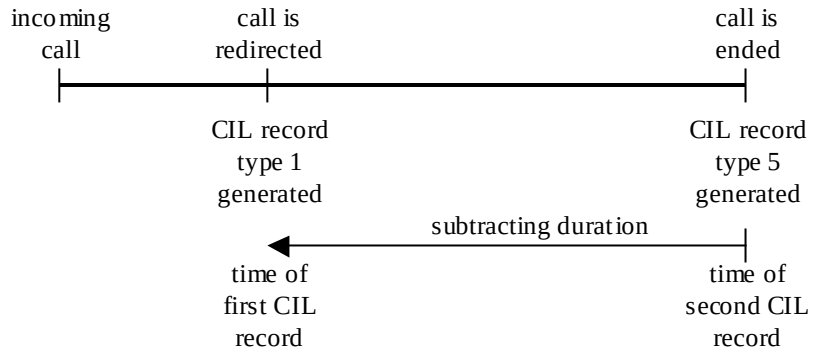

Figure 17

The same method of relating CIL records output at different times applies to all scenarios except when CIL records are output concurrently (see 8.3.4). However, this method involves registered and measured time with limited resolution and precision. That's why small deviations from the calculated time value have to be allowed. The resolution limitation (seconds) also makes it possible to get several matches. But there are further conditions that have to be met by related CIL records.

All CIL records identify the broker station involved in the call by either the ODN or ADN. Related CIL records must always identify the same broker station. Thus all CIL records can be grouped by their respectively broker station identifiers. This minimizes the risk of getting a false match since a single broker station can only (or is only allowed) to handle one call at a time.

As mentioned above, each CIL record is marked with a timestamp of its arrival, i.e. when the TLU receives the CIL record from the serial interface. The reason for not using the timestamp in the CIL record (call ending-time field) is that it only supports minute resolution, which is too rough to reassure that correct matches are made. Additionally, these CIL record timestamps depends on the internal time in the PBX and difficulties may arise when this time is altered (adjusted or day-time saving altering).

8.3.4 Concurrent CIL records

Certain types of CIL records, or maybe pairs of CIL records, can be concurrent as mentioned in section 7.3.2. That means that they are output concurrently by CIL. The interface is serial, thus they may not be output simultaneously but rather in a sequential order. The order in which they are output is undefined and additionally, some other CIL record output may be interleaved due to concurrency properties in the PBX. This makes it a little harder to bundle concurrent CIL records together.

Why must concurrent CIL record be bundled together? Why can't they be treated as all the other CIL records in a sequential fashion? Well, the reason for this is that concurrent CIL records differ from sequential CIL records in what time span their duration values cover (7.3.2). Unfortunately, it's impossible to distinguish concurrent CIL records from sequential by any other means than to find its concurrent counterpart. Therefore it's necessary to establish concurrency relationship before processing the call data.

To establish a concurrency relationship between two CIL records, it's necessary to compare the broker station identifier and the timestamp of them both. The comparison needs to be performed between all CIL records of types that can be concurrent, which are output within a certain time-window. The size of the time-window need to be large enough to always keep concurrent CIL records within the same time window, but also small enough to minimize the risk of establishing false concurrency relationships between sequential CIL records. What size the time window should have is determined by running long tests to register the maximum deviation between two concurrent CIL records. This value can then be used as the maximum allowed deviation[6].

[6] The maximum deviation registered is 1 second.

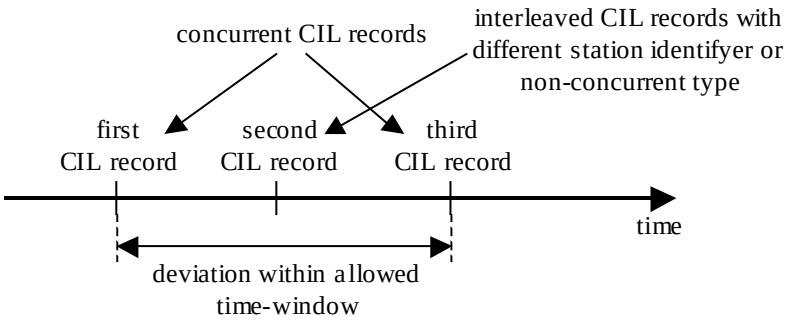

Figure 18

Figure 18 shows two concurrent CIL records that are generated within the allowed time-window and will be determined to be concurrent if they have matching station identifiers. They are interleaved by another CIL record that is either of a type that cannot be concurrent or has a different station identifier than the others. If the interleaved CIL record would have the same station identifier as the other CIL records and is of a possibly concurrent type, the analysis algorithm would incorrectly assume that this CIL record is concurrent with the closer of the two other CIL records, even though it's not concurrent to any of them. Note that this can only happen if the broker handles two incoming calls simultaneously, which is not permitted.

The types that can be concurrent and how they can be paired together are as follows:

- "Incoming Call" concurrent with "External Redirect"
- "Incoming Call" concurrent with "Internal Call"
- "Internal Redirect" concurrent with "External Redirect"
- "Internal Redirect" concurrent with "Internal Call"

Only if CIL records that seem to be concurrent match one of these type pairs, are they determined to be concurrent.

8.3.5 Irrelevant CIL records

All calls in the PBX generates CIL records and since in this case the PBX is used also for other purposes than the directory inquiry, CIL records that are not relevant for the directory inquiry billing system are generated. Therefore each CIL record must be examined and determined to be relevant or irrelevant before further processing. This is done after the type checking described in section 8.3.2. It's of course only necessary to examine CIL records of valid types since invalid-typed CIL records are already marked as irrelevant during type checking.

The relevance-checking algorithm follows the diagram in Figure 19. If progress in the algorithm is prevented by the lack of some transition from a decision, that CIL record is considered to be irrelevant. The CIL record will be marked as irrelevant with the "failed" decision as the cause e.g. irrelevant because of invalid ODN etc.

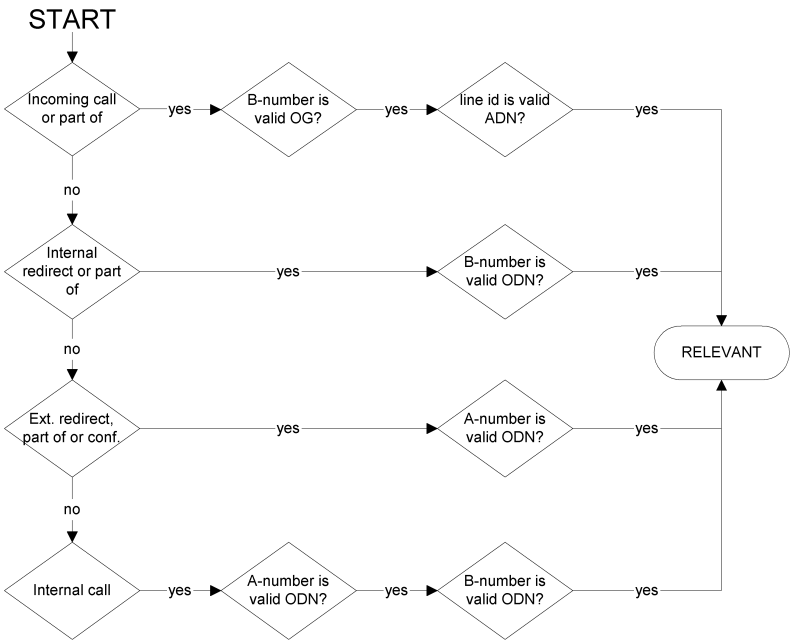

Figure 19

Since the PBX is used for other purposes than the directory inquiry[7], it's also possible that calls follows scenarios that are not permitted and consequently not handled by the TLU. As a consequence, sometimes it's not sufficient to use the algorithm in Figure 19 to determine if a CIL record is indeed relevant. Even CIL records that pass this algorithm as relevant may be irrelevant since they may originate from a call that doesn't correspond to any of the defined scenarios. E.g. a CIL record of type 5 may be irrelevant even though it has a valid ODN, if a call that is not a directory inquiry call is redirected from a valid broker station. The only way to know if this seemingly relevant CIL record is indeed relevant is if it's related to a relevant CIL record. Analogous, the only way to know if this seemingly relevant CIL record is irrelevant is if it's related to an irrelevant CIL record. Therefore all irrelevant CIL records that are not irrelevant because of an invalid type, has to be stored for later CIL records to find, only to be able to determine if the later CIL records are relevant or not. CIL records of invalid type are not stored since its contents cannot be identified.

8.3.6 Handling internally redirected calls

As mentioned in section 7.3.3, internal redirection is only allowed if it follows the "redirect after answer" semantics because only then is an "internal call" CIL record generated. This CIL record is needed to relate the "internal redirect" CIL record to the earlier generated "incoming call" CIL record. In the "incoming call" CIL record the first broker station is identified, but in the "internal redirect" CIL record the second broker station is identified. The only way to relate these CIL records is by using the "internal call" CIL record output concurrently with the "incoming call" CIL record, which identifies both the first and the second broker stations.

Additionally, since the "incoming call" CIL record and the "internal call" CIL record are generated concurrently, the analyzing algorithm is able to realize that an incoming call is internally redirected at the same time it's redirected, which is in fact before the "internal redirect" CIL record is generated. This knowledge makes it possible for the TLU to hold on to the information about this incoming call, omitting to report it to the transaction database until the internally redirected call is ended. Thereby it's possible to append all internally redirected parts of the call (there can be several internal redirections in sequence, see 7.3.6) to the incoming call and generate an article that corresponds to an incoming call that is the sum of the incoming part and all the internal redirected parts of the call. This is what's desirable since internally redirected calls are just continuations on the incoming call and should not generate new initial costs, but only add more ticks to the article (4.4.1).

---

[7] Ahhaaa's PBX is not only used for the directory inquiry service, but also serves as the offices PBX, i.e. all business and private calls to and from employees uses the PBX. All these calls are also registered by CIL.

8.3.7 Handling three-party calls

Three-party calls are a little bit tricky to handle since their outcome in CIL perspective is very different depending on the order in which the participants end their calls (7.3.5). In the case when the broker ends the call before any of the other two parts ends the call, the three-party call is reverted into a redirected call. This leaves no possibility to determine if the call was a three-party call or an externally redirected call by the CIL records that are generated by that call. That's why three-party calls must be limited to defined three-party call telephone numbers, to which no redirection can be allowed. In that way the problem is solved by introducing limitations. Another approach would be to force the brokers to never end their part of the call until they're certain that at least one of the other parts of the three-party call has ended. Sometimes it may be difficult to determine if another party of the three-party call has ended, or it may actually be the case that the broker is not needed any more by the customer and it's desirable that the broker should end the call to be able to accept new incoming calls. That's why the former solution is preferred.

Still unsolved remains the case when the broker is not ending the call first. In this case it's actually possible to determine that it was a three-party call that took place, since a CIL record of type conference, Conference call, will be generated (7.3.5). However, this knowledge is unnecessary when the call can be determined to be a three-party call anyway by examining the third-party telephone number.

Since the CIL records of type incoming call and type conference can be generated in different order depending on which of the parties ends the call first, the A-party or the C-party. This results in two different cases of overlapping durations (7.3.5). Either way it's necessary to have both CIL records before being able to generate any transactions. If the conference CIL record is generated first, i.e. the C-party ends the call first, it can be assumed that there will be a corresponding CIL record of type incoming call generated. Therefore all generated incoming call CIL records with the same broker station identifier are compared to conference CIL record. If it can be determined that the three-party call started during the time span of the incoming call represented by an incoming call CIL record, then it can be assumed that it this is the incoming call CIL record that is related to the conference CIL record.

To determine if a three-party call represented by a conference CIL record started during the time span of another CIL record of type incoming call, it suffices to examine if the starting time of the conference CIL record lies in between the starting time and the ending time of the incoming call CIL record. The ending times of the CIL records are given since they correspond to the timestamps. The starting times are easy to calculate by subtracting respectively duration from respectively ending time. This is illustrated in Figure 20

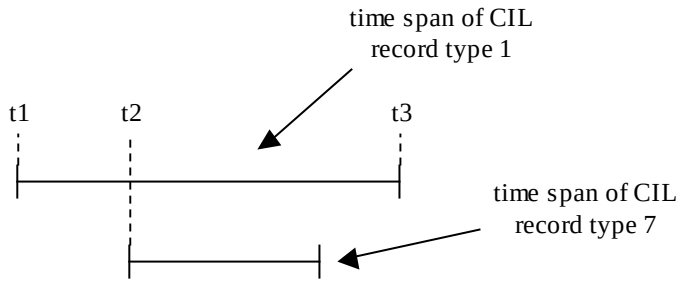

Figure 20 *The starting time of the conference CIL record (type 7), t2, is in between the starting time, t1, and ending time, t3, of the incoming call CIL record (type 1). It can be assumed that the both CIL records are related.*

Now to the case when the A-party ends the call first. This will generate the CIL record of type incoming call first. It will be necessary for this CIL record to wait for the CIL record of type conference to be generated, i.e. for the C-party and the broker to end the call. Then by verifying that the conference CIL records starting time is between the incoming call CIL record starting and ending times as in the previous case, it can be assumed that the CIL records are related. But not every CIL record of type incoming call is part of a three-party call, rather just in exceptional cases. Therefore the processing of an incoming call CIL record cannot wait forever for a conference CIL record to arrive, since it may never be generated. This enforces a time limit on the remaining conversation between the broker and the C-party. This time limit defines for how long time the processing of an incoming call CIL record will wait for a conference CIL record to arrive. If no conference CIL record arrives within this time, the

incoming call CIL record is assumed to be an “ordinary” incoming call CIL record and is processed likewise. This is illustrated in Figure 21.

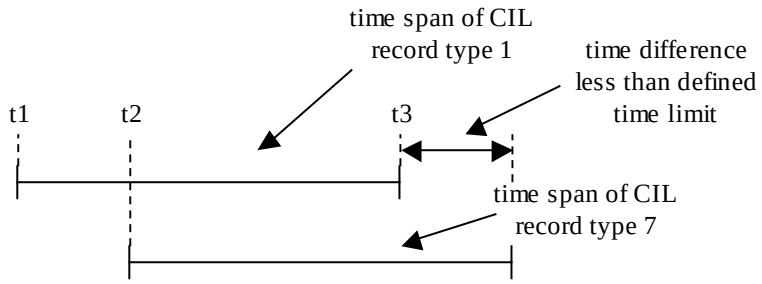

Figure 21 *The starting time of the conference CIL record (type 7), t2, is in between the starting time, t1, and ending time, t3, of the incoming call CIL record (type 1). Additionally the time difference between the ending times is less than the defined time limit. It can be assumed that the both CIL records are related.*

8.3.8 The analysis algorithm

The CIL record analysis algorithm is illustrated in Figure 22. This algorithm handles all valid typed CIL records in all allowed scenarios. If the analysis of a CIL record cannot proceed after a decision that lacks either “yes” or “no” transition, the analysis is interrupted and the CIL record is discarded.

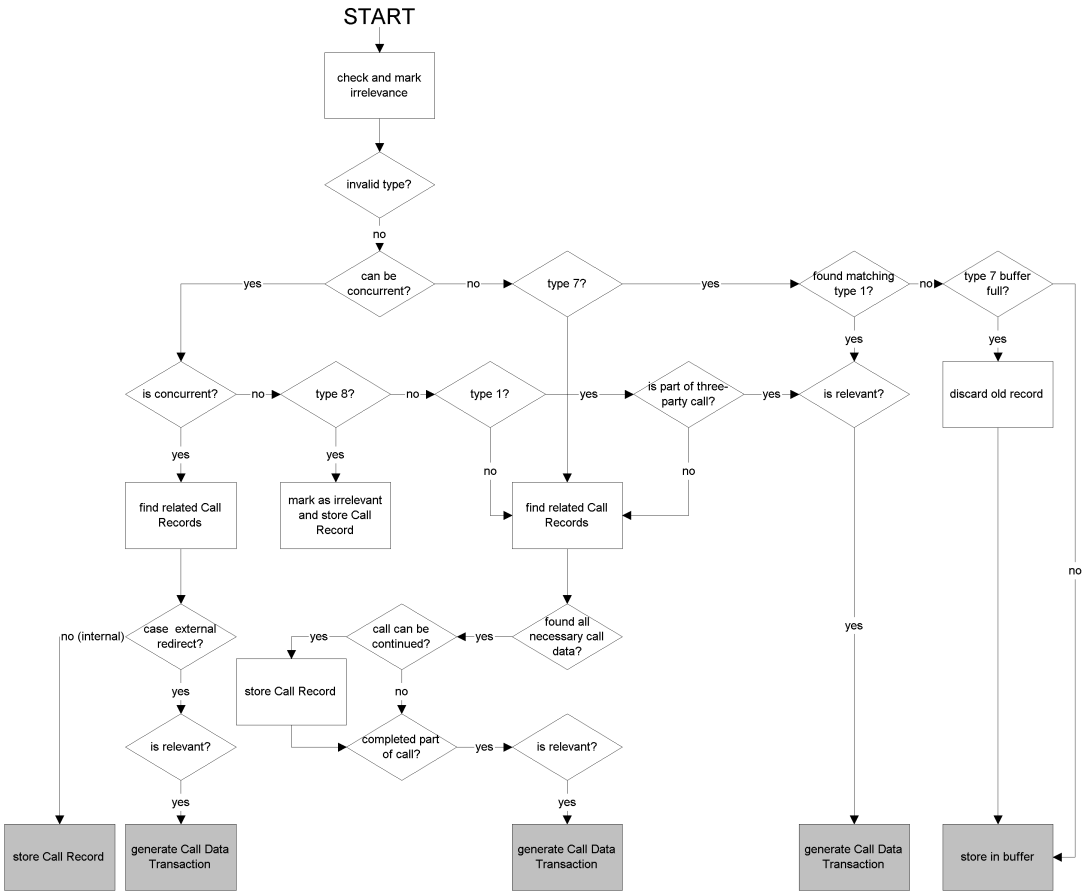

Figure 22

8.4 Design of TLU

8.4.1 Design and Programming language

Most designs follow some kind of programming model such object oriented or procedural programming models. Programming languages in turn supports programming models. Different programming languages support different programming models, thus some programming languages are better suited for a specific programming model than others, and vice versa. This means that the choice of design and of programming language cannot be made independently.

The choice of programming language also depends on the knowledge and experience of the programmer. For a programmer to be able to use a programming language correctly and to utilize its facilities and support of programming models, the programmer need experience in using the language and in supported programming models. Only then will a programmer be able to create a design that leads to a successful result [17].

This will be kept in mind during the following sections when the design and choice of programming language is discussed.

8.4.2 The main TLU operation

The main operation of the TLU is to:

1. Receive call data from CIL by reading from the serial interface.
2. Analyze the call data, create transaction data, receive transaction data from client applications and generate a transaction.
3. Report the transaction data to the transaction database.

Here, steps 1 and 3 are simpler than step 2, which does all the analysis of the call data to create transaction data. If steps 1 to 3 are performed sequentially, all processing of CIL record has to be finished before a new CIL record may be received. The processing of a CIL record may be time consuming and may prevent the TLU from receiving newly generated CIL records immediately. As a consequence the performance of the TLU may drop below an acceptable level.

A solution is to use an interrupt mechanism to trigger an interrupt in the processing whenever there's a new CIL record available to read from the incoming serial port. The processing is interrupted and the new CIL record is read into a buffer holding all incoming CIL records. This requires that the processing algorithm never is blocked by an inter process communication call or a database connection call or by a time consuming calculation. This design approach is vulnerable to blocking calls and it's difficult to keep complex computations from blocking the thread from receiving new CIL records. The risk is that the computation becomes even more complex when integrating the interrupt handling mechanism into it.

An alternative to using some kind of interrupt mechanism in an otherwise sequential operation, step 1 of the operation could run in a separate thread, executing concurrently with the other steps. This of course assumes that the programming language used for implementation and the operating system supports multithreading. This way the interrupts are replaced by context switching handled by the operating system, relieving the call data processing algorithms from having to handle interrupts.

When the main TLU operation is divided into these three separate steps, it's easy to consider each step as a single thread of operations executing concurrently with the other two steps, similar to a production line in an industry. If a multithreaded design is used, it's also possible to separate step 2 and 3 of the operation into different threads. This would result in an overall multithreaded design, using a separate thread for each of the three steps in the main operation of the TLU. Later it will show that all steps consist more or less of *I/O-bound computations*[8] and chances are that the total performance of the TLU will actually increase when using multiple threads [6]. Additionally, the design becomes more understandable when the three naturally concurrent operations are modeled as separate threads.

[8] An I/O-bound computation is a computation that is spending most of its computing time waiting for I/O-operations to complete.

8.4.3 Implementation programming language

Since the design will benefit from a multithreaded programming model as described above, it's preferable to use a programming language that supports this programming model. Additionally it will have to support implementation of services under Windows NT 4.0 as stated in the specification. Thus suitable candidates are:

- C++
- Visual Basic
- Java

Of course there are other programming languages that fulfils the above stated requirements, but these three candidates are the most familiar to the programmer. As described in section 8.2.5, it's a necessary requirement that the programmer is familiar with the programming language to be able to utilize its features.

After considering using each one of the above stated programming languages, the choice was to use C++ with the following motivations:

- The programmer has more experience in using the C++ programming language than in using any other programming language.
- The programmer is familiar with the programming environment of Microsoft Visual C++ and has excellent prerequisites to be able to bring out a good result using this environment.
- The company advisors (Peo Orvendal, Björn Sjöstrand) recommended using C++ as implementation language.

These motivations do not in any way conclude that other programming languages are less suitable for implementing the TLU in general. They just conclude that in this particular case, C++ is the best choice.

8.4.4 Communication with database

The TLU communicates with the database during operation for two reasons:

1. Make transactions with transaction data
2. Obtain data necessary for processing of call data

These two cases are handled differently. When making transactions the data flow is always from the TLU to the database. It's absolutely crucial for transactions to complete successfully since the billing entirely depends on the data in the transactions. In case of a database connection failure, the operation to make the transaction must be retried until it's successful. This means that the operation of making transactions is blocked if the connection to the database fails.

The operation of obtaining data necessary for processing always implies a data flow from the database to the TLU. This data is updated in the database by other parts (administrative parts) of the billing system. Since this data is not updated frequently, performance can be gained by caching the data. In this case, if the database connection fails, the processing operation may continue, using the cached data. Updates of the cache may even be postponed until the database connection works again.

The data flow in the communication with the database is illustrated in Figure 23.

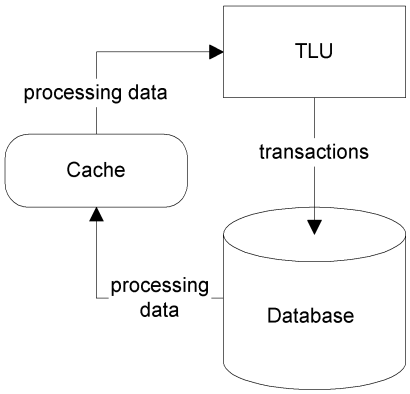

Figure 23

8.4.5 Threading model of TLU design

The design of the TLU is multi-threaded, separating different parts of its operation in different threads. The main reasons for this are to:

- Modularise naturally concurrent components of the operation to simplify the understanding of the design.
- Be able to increase the total performance by separating I/O-based computations into separate threads [6].

In section 8.4.2 the main operation threads were briefly described as three different parts of the TLU operation. These will be named and described in more detail below. Furthermore, additional threads will be added to the model, threads that run administrative computations.

The figure shows the different threads and the dataflow of different types of data. The synchronization between threads working with data in the dataflow, the CIL Reader, Call Data Processor and the Call Data Transaction Manager, is performed when accessing the queues. All push and pop methods on the queues are thread-safe and guaranties exclusive access to the queue. This way the data flow indirectly controls the synchronization of the worker threads.

The Queue Length Monitor thread synchronized with the TLU Manager by signaling that there are new monitoring data available periodically. Synchronization between the Client Report thread and the Call Data Processor is done via the Client Report Store, which is accessed exclusively by the threads in a similar way as for the queues. All threads periodically checks for the termination event from the TLU Manager that signals for termination (8.4.5.4.3).

Context switching is handled by the Operating System (Windows NT) and the priorities of the threads are all set to *Normal*. If the processor load becomes high, the entire process priority may be increased to allow all threads run more often, rather than increasing a single threads priority. The threads taking the most of the processor time are the threads that handle the data in the data flow, whilst the monitoring and supervising parts require little time. The additional process time gained by increasing the process priority will therefore be utilized the most by the data processing parts. This is why increasing the entire process priority makes sense.

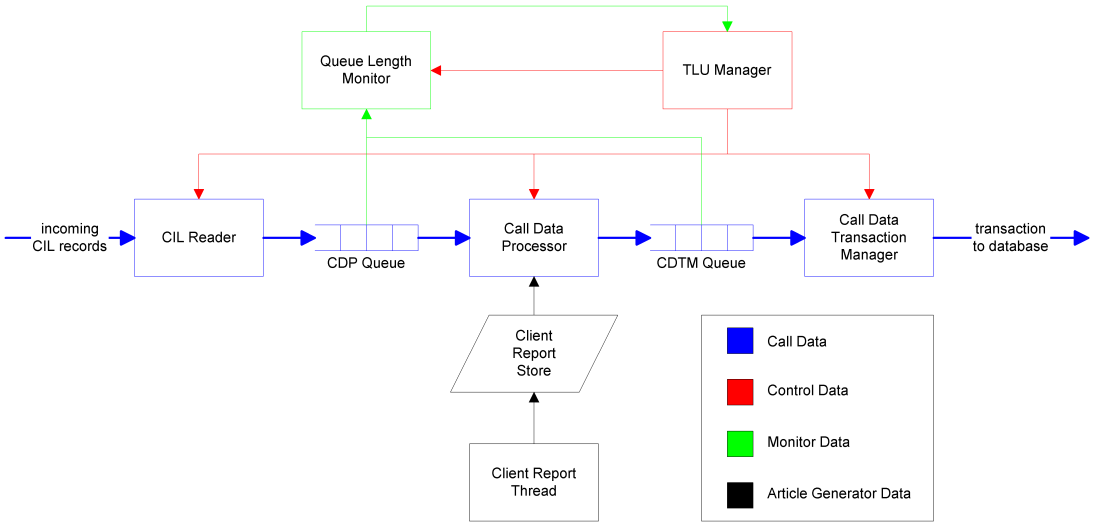

Figure 24

8.4.5.1 CILReader, CILR

The CILReader is the thread with the sole task to receive data transmitted from the serial interface of CIL. This corresponds to step 1 in the main operation of the TLU as described in section 8.4.2. It's important to keep the workload on this thread to a minimum, making it possible for it to always receive data from the serial port when data arrives. Failure in doing so can cause CIL records to be lost as described in section 8.2.1.

On creation, the CIL Reader will open the serial port, which is connected to the CIL output. It will then start receiving CIL records by reading from that serial port. Each completely read CIL record is marked with its arrival time and put into the CallDataProcessor Queue, CDPQ. After that it's ready to read a new CIL record from the port. The workflow is illustrated in Figure 25. The time-consuming part of this operation is, except for the reading of CIL records from the serial port, to put complete CIL records into the CDPQ. The reasons are that the CDPQ is located on disc with relatively long access time and that it's shared with another thread (that dequeues the CIL records from the queue) that will lock the queue from time to time.

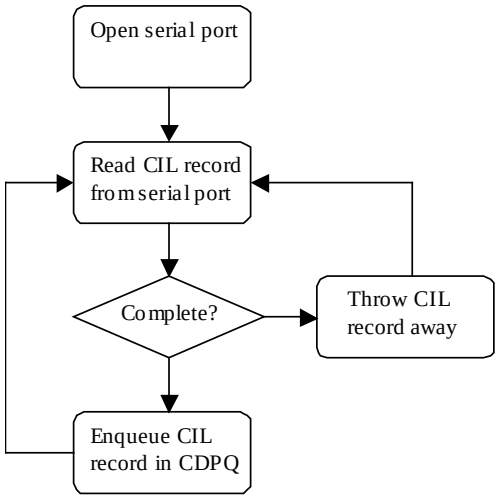

Figure 25

8.4.5.2 CallDataProcessor, CDP

The CallDataProcessor is the thread that analyses the CIL records and creates CallDataTransactions based on the call data in the CIL records. This corresponds to step 2 in the main operation of the TLU as described in section 8.4.2. The analysis consists of interpreting the CIL records to find out what have happened and what articles these actions correspond to. The analysis is described in section 8.2.5.

Not all CIL records hold enough call data to generate a transaction but is related to CIL record that will arrive later and will complete the call data. Therefore there's not always possible to generate a transaction. If it isn't, the CIL record is just stored in the store as usual and the next CIL record is fetched from the queue. When it's possible to generate a transaction, the Client Report Stored is searched for call data to be included in the transaction and the transaction is created. This is illustrated in Figure 26.

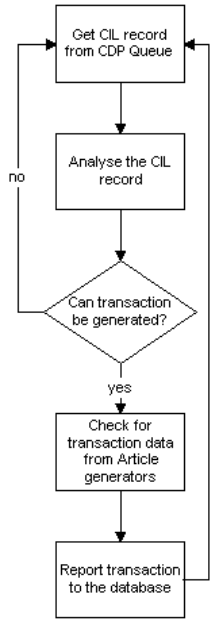

Figure 26

8.4.5.2.1 Non-sequential dataflow in the CDP

The analysis of each CIL record starts in the same sequential order as the CIL records are ordered by in the CDPQ but is not necessarily ended in the same order. This is because the analysis is not sequential but rather handles several CIL records concurrently and some CIL records are analyzed faster than others due to enforced delays as described in sections 8.4.5.2.3 and 8.4.5.2.4. Consequently, the CDP could be designed to be multi-threaded itself, allowing each thread to analyze only one CIL record at a time. However, this would require a more complex synchronizing procedure when related CIL records

are analyzed in different threads. Instead CIL records are buffered in wait for conditions to be met or until timed out, while new CIL records without relational conditions are processed as illustrated in Figure 27.

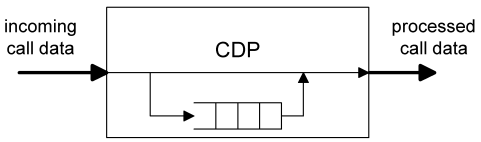

Figure 27 *Call data with relational dependencies is buffered until the conditions are met, whilst call data without dependencies are processed.*

8.4.5.2.2 Communication with database

Communication with the database is necessary to retrieve information crucial for the analysis (8.4.4). When determining if CIL records are relevant, broker station identifiers and Operator Group numbers need to be verified against the database. Also when different types of station identifiers need to be translated to other types of station identifiers (ADN to ODN or vice versa) these have to be mapped through the database. This makes the CDP thread vulnerable to database connection losses. If the information cannot be retrieved, the analysis cannot proceed, and the CDP becomes stalled. The information can be cached to decrease the dependency of a working database connection, since the CDP won't use the database connection as long as the cache is up to date. However, when the cache is invalidated the database connection will have to be used to update the cached data. A consequence of caching data is that the propagation of changes in the database will be delayed until the cached data is renewed.

8.4.5.2.3 Determining concurrency relationship

As described in section 8.3.4, all CIL records that potentially can be concurrent need to be compared with all other potentially concurrent CIL records that are generated within a maximum deviation time-window from itself. As a consequence, all arriving CIL records that potentially can be concurrent need to be delayed until either another CIL record is found to be concurrent with the former, or the time-window has passed and the CIL record can be determined to be non-concurrent.

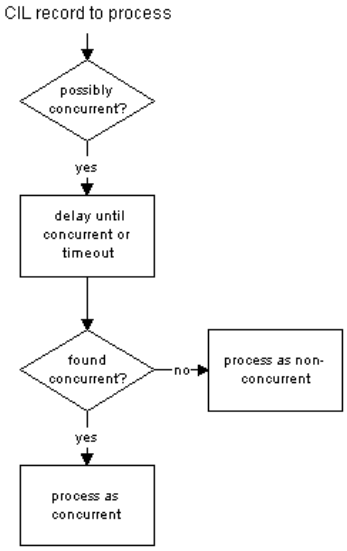

Figure 28

8.4.5.2.4 Recognizing three-party calls

In section 8.3.7 it's explained how three-party calls can generate different types of CIL records depending on in which order the participants end the call and how this is handled in the analysis. To recognize the CIL record of type incoming call as being part of a three-party call in combination with a

conference CIL record it's necessary to delay the incoming call CIL record until the conference CIL record arrives, or until the defined time limit has passed. If a conference CIL record arrives within the time limit, it's possible to deduce that the incoming call CIL record originates from a three-party call and is related to the conference CIL record. If however it's timed out before any conference CIL record arrives, it's assumed to originate from a "normal" incoming call. This is illustrated in Figure 29.

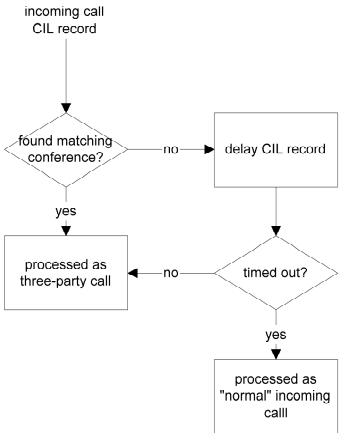

Figure 29

From that follows that all incoming call CIL records, even those that are not part of three-party calls, need to be delayed since it's impossible to know on before hand if they are part of three-party calls.

Similarly, all conference CIL records need to be delayed in wait for the corresponding incoming call CIL record to arrive if it hasn't already arrived when the conference CIL record is analyzed. This delay doesn't have to be limited by any time limit though, since conference CIL records are never generated unless there was actually a three-party call. However, it suffices to buffer only one conference CIL record for each broker station, and replace this if a new conference CIL record for the same broker station arrives, since a broker cannot participate in several simultaneous three-party calls.

8.4.5.3 Call Data Transaction Manager

The Call Data Transaction Manager is the thread that reports the transactions created by the Call Data Processor to the transaction database. The transactions are located in the CDTM Queue, where the CDP has put them, so the operation is simply a matter of fetching the transactions from the queue and report them to the database, one by one (Figure 30).

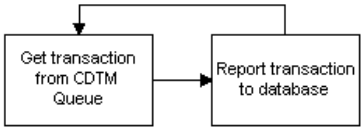

Figure 30

8.4.5.3.1 Report transactions

Reporting transactions to the transaction database includes initializing the transaction, create new articles and report that the transaction is complete. When communicating with the database, a reporter id corresponding to the TLU is used as identification of the reporting process. This id is obtained from the database when initiating the CDTM.

If a transaction initialization or article creation fails it's reported as an error and logged. Then the CDTM continues with next transaction or article. When an initialization fails, the Operator Group number is checked against the database since it may have become invalid since when the CDP found it to be valid (8.3.5). If it's indeed invalid this is not logged as an error, rather as an irrelevant transaction.

8.4.5.3.2 Handle database connection loss

If the connection to the transaction database is lost, the Call Data Transaction Manager cannot continue. The operation is stalled and the thread continues trying to connect to the database indefinitely. Each failed retry is logged in the error log. Meanwhile, all incoming transactions buffer up

in the Call Data Transaction Manager Queue. When connection to the transaction database is re-established, the operation of the Call Data Transaction Manager is resumed.

8.4.5.4 TLU Manager

The TLU Manager is responsible for the operation of the TLU, creating and deleting threads as well as creating and deleting globally accessible object. It does most of its work during start up and termination of the TLU. When the TLU is running, this is the thread that responds to control events from the user. It then controls the other threads to perform whatever the user requested.

8.4.5.4.1 Start up

On start up, the TLU Manager will create a registry key object that holds the registry information for the TLU, controlling parts of the TLU operation. It will also create the TLU Log where progress and errors are logged. Then it will create and initialize the Call Data Processor Queue and the Call Data Transaction Manager Queue. Then it can create the other threads.

8.4.5.4.2 During operation

During operation, the TLU Manager waits for control events from the user and handle incoming events accordingly. Normally, it will not receive any user control events until the termination event arrives, instructing the TLU Manger to terminate the TLU operation. Thus the TLU Manager will be blocked and not consume any processing power at all.

8.4.5.4.3 Termination

Even though the TLU should operate at all time, there may be times when it needs to be shut down e.g. for system upgrades. In this case it's important that the execution of the TLU may terminate without errors. Termination request is signaled from the *user* by setting a dedicated *termination event* to its signaled state.

On termination, the TLU Manager will see to that the other threads are terminating correctly and in the correct order of complete operation. This is important to avoid the loss of data that is currently processed by TLU. After all threads have terminated successfully, the TLU Manager de-allocates all globally accessible objects that it owns and terminates.

The threads need to be terminated in the following order:

1. CIL Reader
2. Call Data Processor
3. Call Data Transaction Manager
4. The remaining threads in no particular order

This order corresponds to the call data flow. When terminating the CIL reader, it will put any currently handled CIL record into the CDP Queue and then terminate. Then when terminating the CDP, it will keep running until it is finished processing the currently processed call data. In continuance, the CDTM will be terminated first after finishing the current transaction. The Call Record Store and the Client Report Store are not stored persistently though, hence all data in these stores will be lost when terminating the TLU.

8.4.5.5 Queue Length Monitor

The Queue Length Monitor is the thread monitoring the length of the Call Data Processor Queue and the Call Data Transaction Manager Queue. Periodically it checks the length of these queues and logs a warning in the error log if either queue has reached the warning limit. The maximum queue length and warning limit are configurable by registry keys.

8.4.5.6 Client Report Thread

The Client Report Thread accepts reports from all article generator clients and stores the reports in the Client Report Store. All incoming reports are validated and time stamped before stored. The store is shared with the CDP, which will fetch the reports from the store and report the articles included in the reports to the database (8.4.5.2).

8.4.6 Log files

The TLU generates several log files during operation:

- Progress/Error log
- Trace log
- CIL log

8.4.6.1 Progress/Error log

In the Progress/Error log, messages about important progress steps of the operation is logged e.g. during start-up, progress messages about status of queues and threads are logged. During normal operation there are very few progress messages that are being logged in the Progress/Error log.

Error and warning messages are also logged in the Progress/Error log. This kind of messages are generated when during normal operation, some operation fails, the connection to the database is lost, transactions fails to be reported or some required piece of information necessary for successful processing of call data is missing.

Only a few extremely severe errors are reported to the database's error log, i.e. will be fetched by the alarm application, but most messages won't. Therefore it's a good idea to periodically check the Progress/Error log to see if there have been any problems.

8.4.6.2 Trace log

In the trace log, all call data processed by the TLU is logged. The log includes the original CIL records together with the transactions and articles deduced from it. This log is used to trace transactions back to the originating CIL records output by the PBX. This is important if other parts of the system reacts to suspicious looking transactions or articles, e.g. in the invoicing system. To be able to find out if the transactions and articles are correct and why the look the way they do, it's necessary to be able to find out what CIL records they originate from and how the TLU interpreted them. This is what the trace log is for.

8.4.6.3 CIL log

The CIL log is not really a log, merely a dump of the CIL output from the PBX. When the TLU reads the CIL output from the serial interface it writes down a copy of the output in a file. The trace log described above contains the original CIL records, but in a modified and possibly reordered state. The CIL log on the other hand contains the unmodified CIL output that can be used to verify the originating output if the information in the trace log is confusing due to modifications or reordering.

9 Implementation of TLU

The implementation programming language has already been chosen to be C++ as discussed in section 8.4.3. In this section details of the implementation as what type of technologies and data structures is used by different parts of the TLU, will be discussed. However all details will not be included.

9.1 Requirements of the TLU operation

There are some direct requirements from the specification of the TLU that affects the way the TLU is implemented. These requirements are discussed below.

9.1.1 System environment

The computer network used by Ahhaaa is completely Microsoft Windows NT based, using Intel architecture in both workstations and servers. The database server used in the billing system is Microsoft SQL Server 7.0. The TLU will initially run on the same machine as the database server, together with other parts of the billing system such as the Alarm Application, CDR generator etc. However, it may be moved to run on another computer in the future.

9.1.2 Run as a system service

The TLU should be implemented to run as a Windows NT system service. This is required for the TLU to be able to run without any user logged on to the system. Users should be able to log on and off at the machine without stopping the TLU. Also if the machine has to reboot, the TLU should be able to start automatically. This is easy to configure in the service manager if the TLU runs as a service.

9.2 Choice of technologies

9.2.1 Programming model

The natural choice of programming paradigm when using C++ is of course the object-oriented. This paradigm fits the design described in section, abstracting threads and globally accessible objects as objects in the object orientated programming model.

The design was developed with C++ in “the back of the head” as described in section 8.4.3. This means that the design was developed to take use of the object-oriented programming paradigm of C++. Thus it comes as no surprise that the programming language, the programming model and the design fits more or less perfectly together.

9.2.2 Inter process communication

Communication with other processes (e.g. broker clients) in the system can use several different technologies:

- Pipes
- Sockets
- NetBIOS

These three APIs all provide means for communication between processes, but they also have their differences. In the following sections, similarities and differences between these APIs will be explored and one of the technologies will be chosen for inter process communication in the billing system. The facts are based on articles by Ruediger Asche ([1], [2], [3], [4] and [5]).

9.2.2.1 Pipes

A *pipe* or *named pipe* consists of a unidirectional or bi-directional communication stream. One process write to one end of the pipe and another process read from the other end of the pipe. When a pipe is created, it must be uniquely named, that is unique on the computer on which it’s created. This enables the pipe to be uniquely identified with a tuple consisting of the computer and the name of the pipe.

The two ends of the pipe are called server- and client-end. The process creating the pipe (the server) always gets a handle to the server-end of the pipe. Clients can get a handle to the client-end of the pipe by using a Win32 API function call supplied with the tuple of the server computer name and the name of the pipe. There is not possible for a client to get a handle to the server-end of the pipe.

The pipe is physically created on the machine on which the creating (server) process resides. The pipe representation in the OS is OS-dependent, i.e. pipes may have different representations and naming conventions in different OS. Thus, difficulties can arise when using pipes to communicate between processes on different machines with different OS. Also, by the use of computer name as part of the identifier, the inter process communication is limited to within the same local network or domain, where the computer names are defined.

To establish a connection between a server and a client, the server must first create the pipe. It then waits for a client to connect to the client-end of the pipe. When the pipe is created, a client can connect to the client-end by using a function call to get a handle to the client-end of the pipe. When doing so, the server is notified that a client is connected and communication can begin.

Communication is simply a matter of reading from and writing a byte stream from and to the ends of the pipe. If the pipe is unidirectional, writing can only be performed to one of the ends of the pipe and reading only from the other end. If either the client or the server disconnects from the pipe the other process is notified and connection is released.

A pipe provides a high abstraction of the underlying network used for the communication, making choice of communication protocols impossible. There is a built in security function allowing the server associating the pipe with an access control list. Then only users included in the access control list will be accepted on connection attempt. They will have to authenticate themselves by using their username and password on connection.

9.2.2.2 Sockets

A *socket* is a communication channel end-point, similar to the ends of a pipe. The socket is identifying the communication channel end-point rather than the actual communication channel as in the case of pipes. The identification consists of the IP address of the other computer and the port number on the other computer to which the other process is connected.

By using the global IP address and the port as identifier, the communication model becomes network transparent, i.e. processes can communicate via an internet using sockets. Sockets early became the standard representation of a communication channel end-points of a TCP connection, though today other communication protocols such as UDP, NetBEUI, IPX etc. can be used through sockets as long as sockets are implemented in their respectively protocol stack.

To establish communication between a server and a client, the server needs to connect to a vacant, well-known port (to the client) on its computer and wait for a client to connect to it. When the server is waiting for connect, the client can issue a connection attempt to the server using the IP-address of the server computer and the port number to which the server is connected.

Communication can be performed in the same way as when using a pipe as communication channel, i.e. reading and writing a stream of bytes using the sockets as the communication channel end-points. Unlike pipes, sockets are always bi-directional. Since sockets can take use of non-connection-oriented protocols like UDP, there's a datagram-oriented API to use with sockets that implements datagram semantics.

Like pipes, sockets can be an abstraction of the underlying network and protocols used in the network layer. When instantiating a socket, no communication protocol has to be specified. But, there is also possible to specify which protocol to use, thus sockets also implement a lower level of abstraction than pipes, allowing the user more control over specific details in the communication.

9.2.2.3 NetBIOS

NetBIOS is a inter process communication API that provides a flexible addressing scheme allowing processes to dynamically assign themselves unique names, which other processes can use to initialize communication with them. The name-computer resolution tables are distributed to all computers in the local network that supports NetBIOS. Thereby, if one process on one computer adds a new name to the table, all processes on all computers within the LAN can establish communication with that process by using the new name, after the name-table update has propagated through the LAN.

NetBIOS also supports names that relates to groups of processes rather than a single process. This makes multicasting simple, since NetBIOS has built-in support for creating groups and establishing communication with them.

The NetBIOS API is somewhat different from the otherwise similar APIs of pipes and sockets. Instead of executing functions as in the case of pipes and sockets, commands are executed by passing a pointer to a network control block (NCB) to the OS, filled in with the proper commands and data.

9.2.2.4 Conclusion – Socket API is the one to use

The properties of the inter process communication between the TLU and other service generators are:

- Unidirectional
- Peer-to-peer
- Synchronous

This type of communication is supported by all of the described APIs. However, NetBIOS is a bit more complex and flexible in its addressing scheme than the others, and since this complexity is not needed here, it's best avoided. Thus we have the remaining two candidates: pipes and sockets. The pipe and socket APIs are very similar and since pipes can be configured in unidirectional mode, maybe pipes should be used?

However, even though today all article generators that the TLU have to communicate with, exists within the same LAN as the TLU itself, in the future new article generators may reside in a different LAN than the TLU. Since pipes don't support internet communication, sockets are the better choice for this application. Additionally, sockets is the one technology used in the system so far for communication between processes and using the sockets API in this case keeps an uniform technology in the system.

9.3 Implementation details

This section will describe the implementation in more detail that the design structure described in section 8.4. It will explain what data structures and what methods are being used to build the design. However, some details will of course be left out to keep this thesis on a fairly high level. To describe all details it would have been best to just publish the source code, which of course is confidential.

9.3.1 Classes

In this section an overview of the classes in the TLU is presented. The classes are used to abstract conceptually units in the TLU design and to establish the relationship between them.

9.3.1.1 Thread classes

The threads: CILR, CDP, CDTM, TLUM, QLM and CRS (see Figure 24) are all implemented as classes derived from a base thread class CWorker, simplifying the thread management by providing several member functions for manipulating the threads. The threads helper functions are implemented as private member functions of its class hiding them from other classes and threads, and its state is kept in private member variables. The class hierarchy is shown in Figure 31.

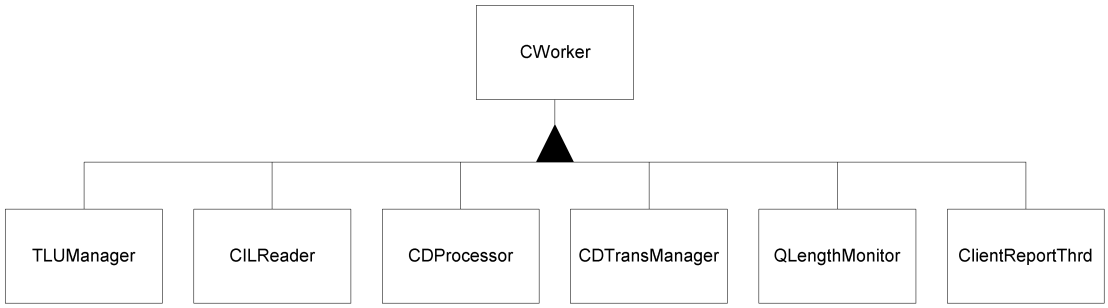

Figure 31

9.3.1.2 Data classes

The ClientReport, CILRecord, CallRecord, CallData and CallDataTransaction classes provide abstractions for different types of data.

The CILRecord class is a serializable class used by the CILR thread to combine the raw CIL record data with timestamps and insert into the CDP Queue.

The CallData class abstracts the raw CIL record call data information by providing access methods to access separate parts of the data. When the CDP has got the CILRecord from the queue it converts it to a CallData object to enable easier processing.

The CallRecord class is used to store call data in the CallRecordStore (9.3.1.3). This class provides methods for appending call data from a CallData object to a CallRecord object already containing call data from an earlier CallData object. This is used to combine call data from several related CIL records to avoid storing redundant call data and to provide easier access to call data from related CIL records. The class append method knows when the data is related and appends the data only if it's okay.

The CallDataTransaction class objects represent the transactions to report to the transaction database. These are serializable and used by the CDP to store the transaction data in and then insert into the CDTM Queue for the CDTM thread to handle.

The ClientReport class abstracts the article data received by the ClientReportServer from the article generators, in combination with a timestamp. The ClientReport objects are the type of objects that is stored in the ClientReportStore (9.3.1.3).

9.3.1.3 Store classes

The ClientReportStore and the CallRecordStore stores data objects of types ClientReport and CallRecord respectively (9.3.1.2). The class relationships are shown in Figure 32. These store classes provide simple Insert and Get methods for inserting to and getting data from the store. The internal representation of the store classes is a multi-dimensional, dynamical data storage structure implemented using basic abstract data types like maps, lists and arrays.

The data is sorted when stored and the sorting is performed when inserting the data using a kind of insert-sort algorithm. This is necessary to effectively be able to find relevant data when using the Get methods with criteria's.

The store classes manage their memory allocation and sees to that data older than the limit for keeping data is thrown away and that its memory is freed. This limit is set to ten hours which corresponds to the maximum time the TLU have to wait for a long duration call to generate a partial CIL record (7.3.7). If no CIL record containing call data related to the stored call data is generated within ten hours, the stored call data is determined to be unnecessary. When data is obtain from the store using the Get methods, the data is effectively removed from the store and the memory is freed.

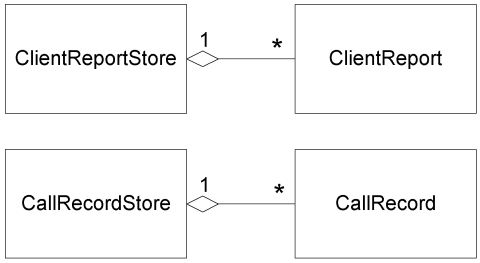

Figure 32

9.3.1.4 Directory Number classes

The DirectoryNumber class represents a telephone number and provides only methods for setting and modifying the telephone number, without verifying its correctness. From this class there are two classes, ADN and ODN, derived that represent ADN and ODN numbers of the broker stations respectively (Figure 33).

The ADN class provides a converting method, "toODN", which allows an ADN to be converted into the corresponding ODN. This method queries the database for the result.

The ODN class provides an "IsValid" method that enables verifying if the number stored in the ODN class is indeed a valid ODN. This also queries the database for the result.

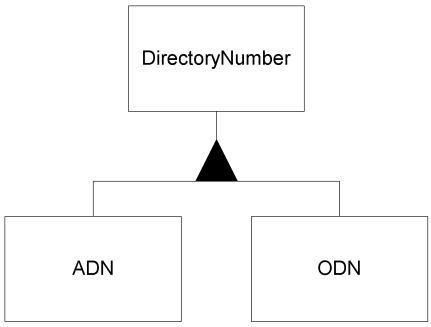

Figure 33

9.3.1.5 Transaction classes

The class Transaction represents the actual transaction and stores the transaction id, A-number and OG-number that are common to all articles in the transaction. Related to the Transaction (Figure 34) is a list of articles represented by objects of the class Article. An Article contains specific information about the article as broker station identifier (ODN), type of article, B-number, duration, timestamp etc.

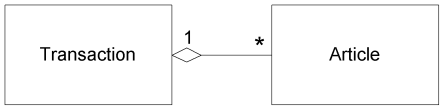

Figure 34

9.3.1.6 Queue class

The class used for implementing the CDPQueue and the CDTMQueue is a template providing a persistent queue for any serializable object that is derived from the basic MFC class CObject. The queue is persistent by using the hard disc drive as storage media rather than RAM. The queue is thread-safe and always keeps a consistent state persistently stored on disc by executing a kind of 2-phase-commit algorithm when inserting or removing objects from the queue.

The size of the queue is fixed and the disc space is used in a circular buffer way, always using the same fixed-size file. The queue-file can optionally be pre-allocated on creation to verify and to be guarantied to get the necessary amount of disc space. If no pre-allocation is made and if the disc runs out of space during insertion of a new object into the queue, an exception is thrown.

The queue classes provide means for recovering an existing queue file and returning it to the exact state the queue object was in when destroyed.

10 Testing of the TLU

Before the TLU and the billing system could be taken into operation it had to be tested gradually up to the point where the system is running with operative status. In the matter of the TLU, tests have been run throughout the development of the application. During analysis and design of CIL record processing, the parts of the TLU reading CIL records from the serial interface was operative and constantly under testing. In fact, the CIL Reader thread in the TLU was actually separated into a different process during development to ease the debugging and testing of other parts. Similar, when the transaction reporting parts of the TLU (CDTM) was developed, the analyzing and processing parts were running to supply the transaction reporting mechanism with debug data.

10.1 Correctness testing and adjustment of parameters

The complete operation of the TLU had to be tested for correctness, that all calls registered by CIL was received by the TLU, that the call data was interpreted correctly in all supported scenarios, that unsupported scenarios and irrelevant CIL records did not create incorrect transaction data and that the transactions were generated and reported to the transaction database correctly. This involves trying out parameter values that need to be trimmed for optimum reliability in the operation.

The client broker reporting and matching of client reports onto transactions was first tested by artificially generating calls and client reports matching these calls in predetermined scenarios. When the matching procedure was functioning satisfactory, tests using the broker station client were performed to verify the operation in real scenarios.

10.2 Performance testing

The specification of the TLU required the performance of the TLU to be enough to handle 60 000 incoming calls per day in average distributed in the same fashion as the distribution of the incoming calls today. Performance of the TLU was tested using simulated traffic rather than real traffic since the performance is in terms of traffic load that the TLU can handle. The test was performed to examine the performance of the three different steps in the TLU operation and to determine where the bottleneck is. Using performance-monitoring tools to measure time and printing results in the debug trace window, an approximation of the performance of the different steps of the operation was obtained.

10.2.1 CIL generator

To test the limitations of the TLU, a CIL generator was used to generate CIL records at an extremely high rate. This way the maximum rate at which the CIL Reader thread in the TLU can receive CIL records could be determined, as well as the maximum call data flow through the TLU.

The CIL generator generates artificial CIL records and outputs them to the serial interface as fast as possible, i.e. there's no idle time in the process. These CIL records are solely of type "incoming call", which will result in one transaction for each CIL record. The actual traffic when the system is in operation will generate other types of CIL records that may require more processing time, but this test will give a hint of the performance of the TLU. The serial output uses simple hardware flow-control that will block the send Win32 API method call made by the CIL generator when outputting data, thus limit the actual CIL generator rate to the rate at which the receiving process receives the data, if this rate is lower than the maximum CIL generator rate. This way there will always be CIL records available for the CIL Reader thread in the TLU to receive.

10.2.2 Performance monitoring

To monitor the performance of different parts of the TLU, time to perform parts of the operation was measured using performance-monitoring tools. The system time was registered every 20^{th} CIL record received by the CILReader and processed by the CDProcessor, and every 20^{th} transaction performed by the CDTransManager. The time difference between the registered points were then calculated and the average time to receive and process 20 CIL records respectively perform 20 transactions were calculated. The results were output to the debug-window and imported into Excel for creating graphs illustrating the results.

10.2.3 Results

The maximum rate with which the CILReader thread can receive CIL records from the serial interface is illustrated in Figure 35. The average time to receive 20 CIL records after receiving totally 10 000 CIL records is 1563 ms. This value corresponds to 12.8 CIL records per second or 46 065 CIL records per hour.

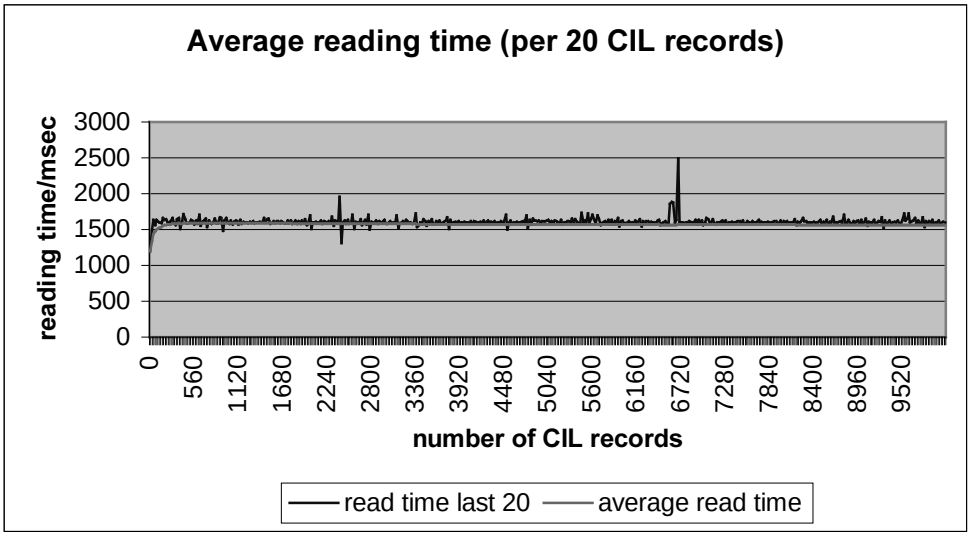

Figure 35

At this rate the CDProcessor thread cannot keep up, hence the CDPQueue will increase in length over time. There's always call data available for the CDProcessor to process, thus the processing time registered by the performance monitoring will correspond to the maximum processing time of the CDProcessor. The result is illustrated in Figure 36.

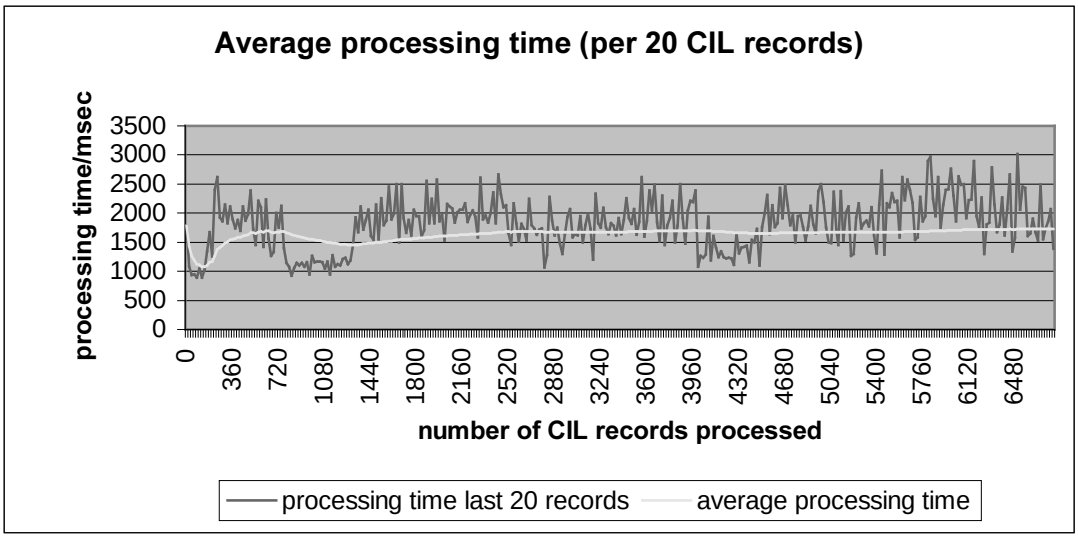

Figure 36

Here the average time to process 20 CIL records after a total of 6780 CIL records, is 1725 ms. This corresponds to 11.6 CIL records per second or 41 739 CIL records per hour.

As shown by the above results, the CDProcessor cannot keep up with the maximum CIL record rate supported by the CILReader. The same relationship exists between the CDProcessor and the CDTransManager, the CDTransManager cannot keep up with the rate of transactions generated by the CDProcessor. This means that the transaction rate measured corresponds to the maximum transaction rate supported by the CDTransManager. The result is illustrated in Figure 37.

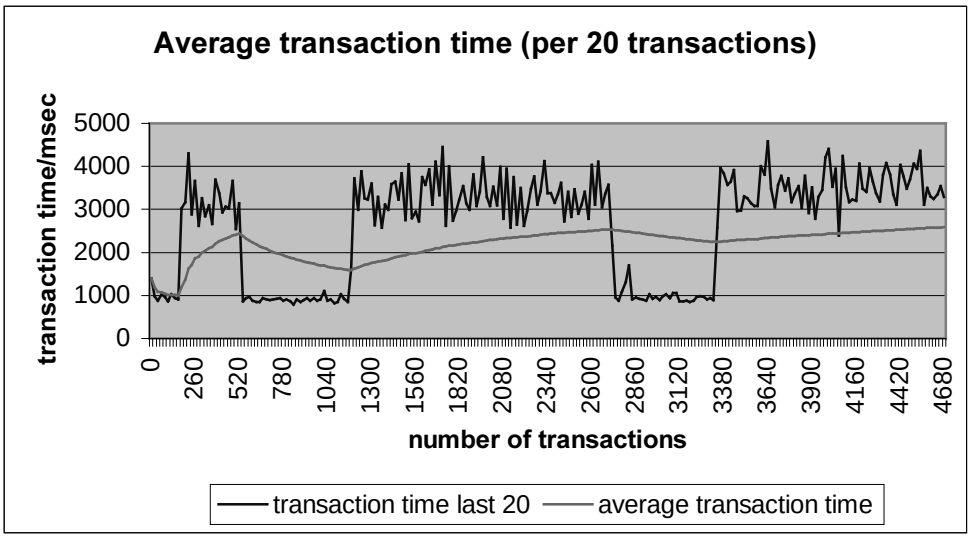

Figure 37

The average time to perform 20 transactions after a total of 4680 transactions is 2583 ms. This corresponds to 7.7 transactions per second or 27 875 transactions per hour. As can be seen in Figure 37, the time to perform 20 transactions drops to around 1000 ms once in a while, whilst it's closer to 3000 ms the rest of the time. These differences probably have to do with variations in load on the database.

10.2.4 Summary

In Figure 38 the CIL record throughput for each processing part in the data flow is compared. It's obvious that the transaction throughput represents the lower limit of the total throughput of the TLU, with its 7.7 transactions per second. However, this assumes that every transaction originates from a single CIL record, which is certainly not always true. If more than one third of the CIL records are bundled together in groups of two or more in each transaction, the processing throughput will begin to limit the total throughput.

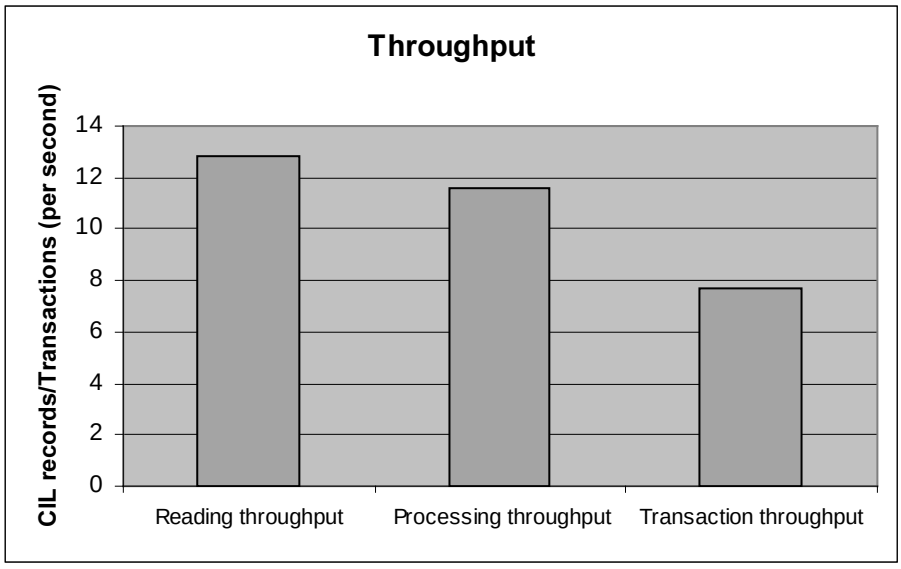

Figure 38

10.2.5 Conclusion

The results show that the performance of the TLU is more than enough to handle the traffic load required in the specification. The requirement is for the TLU to handle 60 000 incoming calls per day which will correspond to approximately the same amount of generated CIL records. With the same

distribution of the calls as today, this will not render to more than 10 000 incoming calls during a single hour with approximately the same number of CIL records generated. The results of the test shows that the TLU is able to receive and process more than four times this traffic load, which gives enough margin to conclude that the performance requirements are well within limits.

The traffic generated by the CIL generator differs from the "real" traffic in that it's only generating "incoming call" types of CIL records. This may underestimate the processing time required by the CDProcessor somewhat since processing of other types of CIL records may be more complex. On the other hand, the "incoming call" type of CIL record always generates a transaction, thus the simulation may result in an unfairly high load on the CDTransManager, which in the simulation is the bottleneck in the data flow. However, since the resulting performance of the TLU is well above critical level, the difference in the generated traffic from the "real" traffic doesn't impose any uncertainties of whether the performance is enough to handle the required traffic.

10.3 Memory usage testing

Another part of the testing was to examine the memory usage of the TLU under operation. The TLU is using dynamic structures to store call data and client reports during operation. The amount of memory used to store this type of data varies with traffic load. Using the Windows NT performance monitor the memory usage could be recorded over time and then compared to the traffic load during the same period to deduce the relationship between the memory usage and the traffic load.

The dynamic call data is stored for 10 hours (9.3.1.3), thus the memory usage of the TLU depends on the last 10 hours traffic. The memory usage also depends on the type of traffic since certain traffic generates more dynamic call data than other (different number of CIL records, see 7.3). A rough estimation can be done though by monitoring the memory usage by the TLU and comparing to the traffic load during the same period.

The maximum memory usage recorded by the performance monitor for each day was plotted against the total number of incoming calls during that same day which resulted in the graph shown in Figure 39.

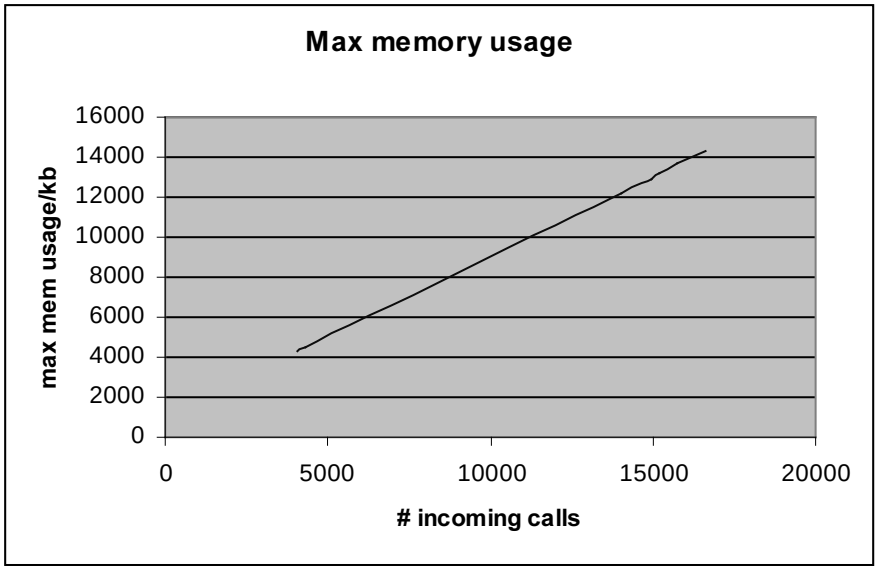

Figure 39

As can be seen in the graph, the memory usage is proportional to the amount of incoming calls. From the graph the proportional constant relating the incoming traffic to the maximum memory usage can be approximated to 0.8, hence the maximum memory usage of the TLU can be approximated by:

Maximum memory usage = 0.8 * # incoming calls

Using this approximation it's possible to estimate the memory usage when the traffic reaches the limit that the TLU is required to handle i.e. 60 000 incoming calls a day:

Maximum memory usage at maximum required traffic = 0.8 * 60 000 = 48 000 KB

This approximation can be used when determining the memory need for the machine on which the TLU will execute.

11 Alarm Application

Included in the problem definition of the degree project (6) were analysis, design and implementation of an Alarm Application that should be able to notify appropriate people about abnormalities in the operation of the billing system. This is a small part of the project and will described short in the following sections.

11.1 Error messages

The billing system generates monitor data based on the transaction data in the transaction database. This monitor data is then examined and if there are reasons to believe that an error has occurred in the system, or that an error may occur soon, an error or warning message is generated and stored. External processes can, using stored procedures in the database, retrieve these error and warning messages (Figure 40). This is used by the Alarm Application for it to be able to determine when to alarm the personal.

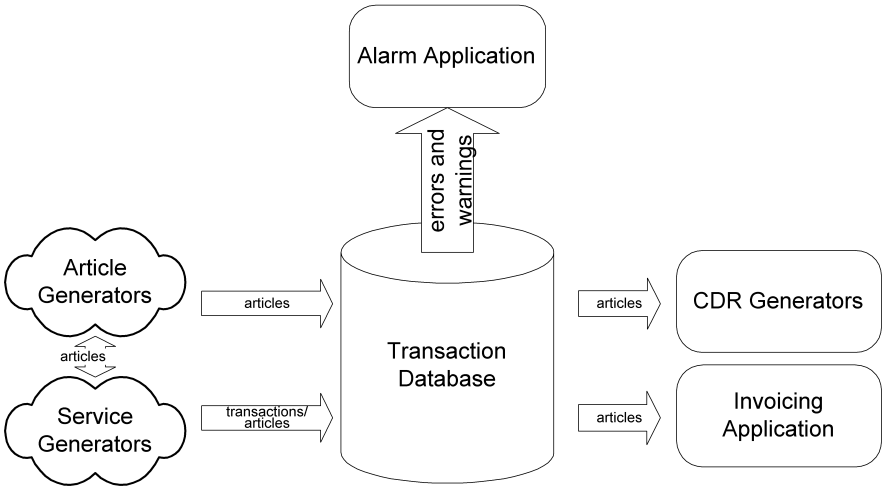

Figure 40

11.2 Specification of Alarm Application

- Retrieve error and warning messages from the monitoring database periodically.
- Inform the appropriate people using GSM SMS text messages.
- Log all errors in an error log on persistent media (disc).
- Run as a Windows NT Service.
- Configurable parameters in the registry.

11.3 Alarm Application design issues

11.3.1 Retrieving error messages

The monitor data warnings and errors, called error messages, are retrieved by using a stored procedure in the database. A maximum number of error messages that should be retrieved in each batch can be specified when calling the procedure. Then a result set containing all error messages up to the batch limit is returned.

The retrieval of error messages is made periodically. The period should be short to enable short response time to errors to minimize the damage they may cause. However, when there are many error messages generated, this retrieval period may be lengthen to keep the “clients” to which the messages are sent from being flooded. This is related to the subject of section 11.3.2.

11.3.2 Handling many error messages

When the billing system is operating normally, there shouldn't be any warnings or errors in the monitor data. However, when abnormalities in the operation occur, there's a high possibility that many errors and warnings are generated by the same fault in the system. Then it makes no sense to send all these error messages as SMS text messages to the receiver, who would be flooded with messages. But if the maximum number of messages retrieved in each batch is limited to just a few, it may take a very long while for the error messages stored in the database to be emptied. This in turn prevents new errors from getting through since old errors may be continuously reported even after the source of the error has been fixed.

This means that in the case of many errors, the batch size should be large to quickly handle the error messages in the database so that new errors may be retrieved. At the same time only a few messages from each batch should be sent to inform personal what it's all about and then a message informing how many warnings and errors that has been generated. Since all messages are logged in the error log, it's possible to read the specific and likely duplicated error messages later when in place.

This calls for a configurable threshold value of the maximum number of error messages that is sent in each batch. If the number of messages exceeds this threshold, only the first maximum number of messages is sent and then a message containing information of how many errors and warnings that were fetched in this batch is generated and sent.

11.3.3 Formatting of error messages

The formatting of the messages is complete when they're fetched from the monitor database and needs no further formatting. However, since SMS text messaged can only consist of a maximum of 160 characters, the pre-formatted messages are truncated if they exceed this maximum number of characters.

11.3.4 Specifying telephone number to send messages to

In the registry there should be a key containing the telephone number that the SMS text messages should be sent to. If several telephone numbers are set, all messages will be sent to all telephone numbers.

11.4 Design of Alarm Application

11.4.1 Implementation programming language

The choice of programming language for implementing the Alarm Application is C++ with similar motivations as in section 8.4.3. In addition code for communicating with the database and with the Ahhaaa SMS Service used for sending SMS text messages (11.4.4.2) were already available and made the implementation process effective.

11.4.2 Design overview

The design of the Alarm Application is fairly simple, since the operation of the application is simple. The design is built on two threads:

1. The **main thread** initialising the Alarm Application on start-up. During operation this thread is waiting for the termination event from the system and then handles the termination of the application.
2. The **working thread** that does the work during operation. This is the thread that periodically checks the monitor data and takes the appropriate actions.

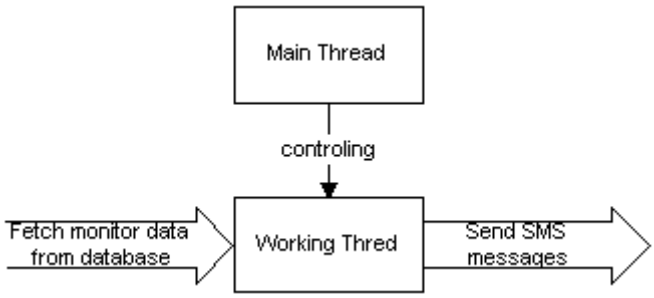

Figure 41 *The two threads in the Alarm Application design, the main thread and the working thread.*

11.4.3 Main Thread

The main thread is the thread started by the Windows NT Service Manager when the service is started. This thread will create and initialise the working thread. When the working thread is started, the main thread is blocked until it receives a termination event from the Service Manager. Then it will terminate the working thread, release all allocated resources an terminate. This will terminate the Alarm Application altogether.

11.4.4 Working Thread

The working thread is created, initialised and started by the main thread as described in section 11.4.3. It will do the work of fetching error messages from the monitor database and send them to the specified telephone numbers as SMS messages.

11.4.4.1 Fetching error messages from monitor database

The working thread creates a database connection to the monitor database on initialisation. It uses this database connection to call a stored procedure in the monitor database that returns a resulting set containing all error messages. The maximum number of error messages that is returned in the set can be specified so that in the case of extremely many messages in the monitor database, not all messages are returned at once becoming unmanageable. This number is set to the maximum batch size defined in the corresponding registry key.

If the number of messages received is higher than the maximum number of messages to send, a message telling how many messages were received is generated. Then the first maximum number of messages to send is sent together with this message. Also the fetching period is changed to the period that should be used when there are many errors in the monitor database.

When the number of messages received is lower or equal to the maximum number of messages to send, the fetching period is set back from the “many errors period” to the “normal” period.

11.4.4.2 Sending SMS text messages

To send SMS text messages the Ahhaaa SMS Service is used. The message is packaged in a SMS structure containing except for the message itself, also the receiving telephone number and other parameters that controls the way the message is handled. Then connects to the SMS Service using a socket to the well known SMS Service port and sends the SMS structure to the SMS Service. It’s possible to request a report that is returned asynchronous and informs the sender of the status of the message. This is disregarded here since if the message couldn’t be sent, there’s no other means by which the error messages can be sent (not for now anyway).

12 Testing of the Alarm Application

The testing of the Alarm Application consisted of verifying its operation and calibrating the parameters of maximum number of messages to send, number of messages to retrieve each batch and fetching period when there are few errors respectively when there are many errors, to give the application a reasonable behavior in different situations. However, these parameters are to be changed in the future to better suit the system in operation.

13 Resulting Product

The result of this project is a fully working billing system for use with a variety of services in Ahhaaa's information system. The specific parts belonging to the degree project resulted in a Traffic Logging Unit that enables analysis and registering of calls in a directory inquiry service and an Alarm Application that in a flexible way reports warnings and errors to appropriate personal using GSM SMS text messages.

The TLU is designed especially for use with the current directory inquiry system operated by Ahhaaa and takes use of specific properties of this system. But it's modularised in a way that separates the interfaces to the PBX and to the broker station clients from the analysing and processing parts as well as from the parts interacting with the transaction database. Therefore it should be relatively easy to modify the TLU from for example reading CIL via the serial interface of a SIU board of a MD110 BC8 to using an Ethernet interface of a MD110 BC10. The application should also work when distributed onto computers in the network or even onto different network communicating via an internet. However, there's no security built in, thus communication over an insecure network probably requires implementing security handling mechanisms.

The Alarm Application resulted in a small, not very resource demanding application that enables reporting of errors and warnings using GSM SMS text messages. The configuration parameters enable configuring the application to behave appropriate in different situations, i.e. both normal and extremely erroneous situations. To use other media than SMS text messages when reporting errors, it's simply a matter of replacing or adding appropriate modules that supports interfaces to different media, e.g. sending e-mail, using some kind of WAP service etc.

14 Conclusions and future work

The system designed and implemented in this project is well adapted both for today's usage and for future demands with easy upgrading. The traffic load handled by the TLU is far greater than the traffic load of today and than the estimated traffic load for a relatively distant future. There are other parts of the system (the PBX CIL interface) that will need upgrading long before the performance of the TLU will become an issue.

When it comes to the Alarm Application, there are many possible ways to improve and add features in the traffic monitoring and error reporting parts of the system. The priorities haven't been high on these parts during the development of the actual system. Future will show if and how these parts of the system needs to be modified and upgraded, when the main system has been operating a while and the understanding of it's behaviour and weaknesses is better known. Hitherto, there's been no need for a more sophisticated monitoring and alarming system than the one provided by the monitoring data in combination with the Alarm Application.

14.1 Major problems during the work

In the beginning of the project, little was known about CIL and it's behaviour. The first impression of the task was that it would be easy and not very complicated to extract call data from the data output by CIL. This however, turned out to be very wrong. The documentation available on the properties of the CIL data and the output was very limited, most documentation discussed configuration of CIL from which the behaviour had to be deduced. Together with the lack of documentation was the fact that CIL was not operative when the project started and didn't start working until one month afterwards. This resulted in a first design approach that wasn't very well suited for the application and later had to be gradually changed to better support consequences of new knowledge.

The A-number field in the CIL output didn't contain the A-number at first; rather it was a reference to the physical line resource used by the incoming call in the PBX. This didn't really delay the work since the line-id number could substitute the A-number during the development of the system. But when InterNordia still didn't get the A-number presentation to work and when the development of the system was almost finished, it began to be a little worrying since the whole concept builds on a working A-number presentation from CIL. In addition it was more difficult to verify that the system worked as it should when there were no "real" A-numbers to verify in the system. Finally after lots of upgrading in the PBX, InterNordia got the A-number presentation to work and everything was fine.

15 References

[1] Asche, Ruediger R. 1995. Communication with Class. *MSDN Library – July 1999*

[2] Asche, Ruediger R. 1995. Garden Hoses at Work. *MSDN Library – July 1999*

[3] Asche, Ruediger R. 1995. Power Outlets in Action: Windows Sockets. *MSDN Library – July 1999*

[4] Asche, Ruediger R. 1995. Aristocratic Communication: NetBIOS. *MSDN Library – July 1999*

[5] Asche, Ruediger R. 1995. Plugs and Jacks: Network Interfaces Compared. *MSDN Library – July 1999*

[6] Asche, Reudiger R. 1996. Win32 Multithreading Performance. *MSDN Library – July 1999*

[7] Conversation with technicians at InterNordia.

[8] Ericsson Business Networks AB. *Extra Facility Description.* Document number: 1/1551-APD 101 02 Uen, Revision: F.

[9] Ericsson Business Networks AB. *Interworking Description.* Document number: 1/155 19-CNA 103 321 Uen, Revision: G2.

[10] Ericsson Business Networks AB. *Installation Instructions.* Document number: 1/1531-CNA 103 321 Uen, Revision: G.

[11] Ericsson Business Networks AB. *Command Description.* Document number: 24/190 82-APD 101 02 Uen, Revision: F.

[12] Ericsson Business Networks AB. *Parameter Description.* Document number: 24/190 84-APD 101 02 Uen, Revision: F.

[13] Jain, Raj. 1991. *The art of computer systems performace analysis.* ISBN 0-471-50336-3, USA: John Wiliey & Sons, Inc.

[14] Kleinrock, Leonard. 1976. *Queueing systems – Volume 2: Computer Applications.* ISBN 0-4710491110X.

[15] Körner, Ulf. 1997. *Köteori och tillförlitlighetsteori.* ISBN 91-44-00480-X, Lund: Studentliteratur.

[16] Råde, Lennart and Westergren, Bertil. 1995. *BETA – Mathematics handbook for science and engineering.* ISBN 91-44-25053-3, Lund: Studentliteratur.

[17] Stroustroup, Bjarne. 1997. *The C++ programming language – third edition.* ISBN 0-201-88954-4, Addison Wesley.

Appendix I. Capacity limitations using SIU board

In this section an approximation to the CIL record loss rate in the SIU board output buffer at different traffic intensities will be calculated. This approximation is made to investigate the limitations of using the serial interface for output of CIL data.

I.I. Introduction

The approximation is based on a simple model of a queuing system, a M/M/1*B system. This system assumes exponentially distributed interarrival times and exponentially distributed service times, using only one server and having a limited buffer space of B buffers ([13] ch. 31.4). Here, the arrival process represents the arrival of newly generated CIL records to the output buffer and the service process represents the process of transmitting the CIL records over the serial connection. Exponentially distributed interarrival times are often used in telephony based queuing systems to model the arrival process and has shown to often be a very good approximation (see [15] ch. 10). The time to transmit the data over the serial line is intuitively constant in this case, since the size of each CIL record and the transmitting rate are constant. However, as also mentioned in [15], systems with general and deterministic service times can often be very closely approximated using exponentially distributed service times in the model. By using exponential distributions rather than general, the calculations can be simplified but still be accurate enough to be used for approximating the limitations of the system.

I.II. The queuing model

As mentioned above, the queuing model that will be used is M/M/1*B[9] i.e. exponentially distributed interarrivaltimes and exponentially distributed service times, using only one server and having a limited buffer space.

This means that the interarrival times are exponentially distributed with arrival intensity λ, hence mean interarrival time is $1/\lambda$. Similarly, the service time is exponentially distributed with service intensity μ, hence mean service time is $1/\mu$. From this the *traffic intensity,* ρ, can be obtained by ([13] ch. 31.2):

$$\rho = \frac{\lambda}{\mu}$$

The intensity of the arrival process is the only parameter that will be varied during the investigation, since this is the only parameter that corresponds to the traffic load on the system (the more traffic, the more CIL records are generated). All other parameters will be kept constant. This allows calculating the CIL record loss rate at different arrival intensities to conclude the maximum arrival intensity supported by the system when using the SIU board for output of CIL data.

I.II.I. Parameters

The arrival intensity, λ, will be varied.

The service intensity, μ, is based on the transmitting rate and the size of the CIL records to transmit. In this case the CIL record size is 64 bytes (7.1). The maximum transmitting rate of the SIU board is 19200 baud at which 1024 bytes are guarantied to be transmitted within 3 seconds [9]. This renders to:

$$\mu = \frac{1024}{64 \cdot 3} = \frac{16}{3} \ \left[\text{CIL records / second}\right]$$

The number of buffer places is based on the CIL record size and the total buffer size. The CIL record size is as already mentioned 64 bytes and the total buffer size is 1024 bytes [8]. Thereby the number of buffer places is obtained:

$$B = \frac{1024}{64} = 16 \left[\text{CIL records}\right]$$

I.II.II. Calculating CIL record loss rate

By constructing and analyzing the corresponding birth-death process, the probability of having n CIL records in the buffer, p_n, is obtained to ([13] p.538):

$$p_n = \rho^n \cdot p_0$$

$$p_0 = \left[1 + \rho \frac{1-\rho^B}{1-\rho} \right]^{-1}$$

From this the CIL record loss rate is calculated by λp_B, i.e. the product of the probability of having a full buffer and a new CIL record arrives.

I.III. Result

The CIL record loss rate was calculated for different arrival intensities, λ, and plotted in a graph (Figure 42). From the graph it can be determined that the CIL record loss rate may be neglected when the CIL record arrival rate is less than close to 4 CIL records per second. When the arrival rate reaches 4 CIL records per second, the output buffer starts to fill up and CIL records are lost.

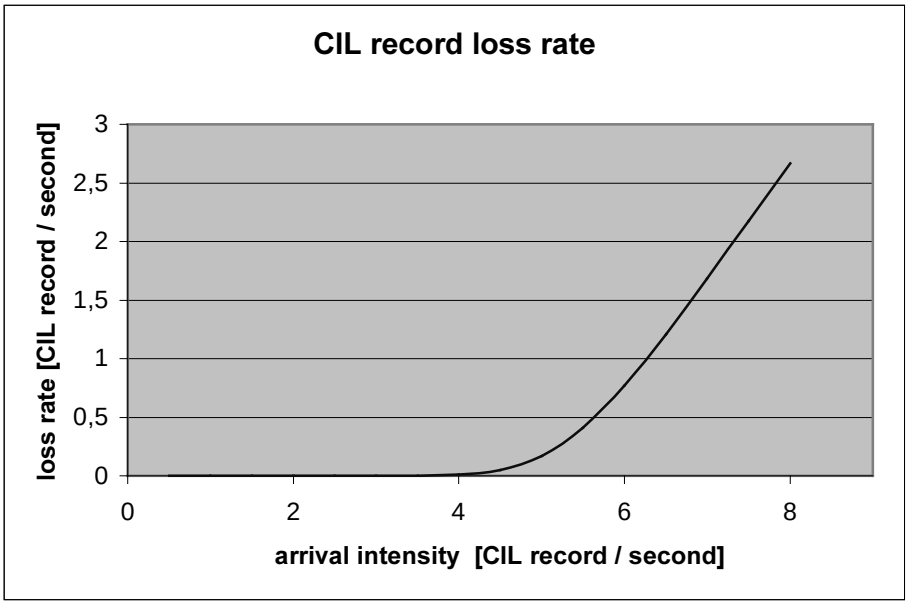

Figure 42

In Figure 43 we can see that this arrival rate, when the output buffer starts to fill up, corresponds to a traffic intensity close to 80%.

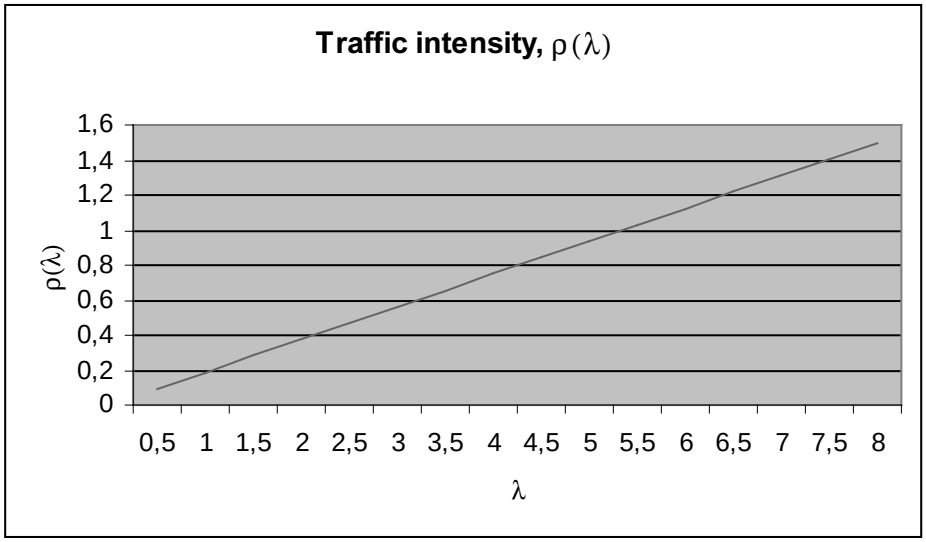

Figure 43

I.IV. Conclusion

The call traffic today during the most intensive hours generates the maximum arrival intensity of approximately 1 CIL record per second, which is far lower than the critical arrival intensity when CIL records begins to get lost. Event if the traffic load increases to three times the traffic load of today, there's no immediate risk of loosing CIL records due to filled output buffer in the SIU board. In addition, performance limitations in the PBX seem to render a natural scattering of output of CIL records, which further decrease the intensity and thus also the risk of buffer overflow.

Appendix II. Glossary

Call Data Processor, CDP	Thread that analysis and processes the call data in CIL records.
Call Data Transaction Manager, CDTM	Thread that makes the transactions with the transaction data delivered by the CDP, to the transaction database.
Call Detail Record, CDR	Record containing details about a call, e.g. duration, calling number, called number etc.
Call Information Log, CIL	Facility in the Ericsson MD110 PBX that makes it possible to generate records containing information about calls.
CDP Queue, CDPQ	Thread safe queue holding call data between the CILR and the CDP, before processed by the CDP.
CDTM Queue, CDTMQ	Thread safe queue holding transaction data between the CDP and the CDTM, before processed by the CDTM.
CIL Reader, CILR	Thread that reads the incoming CIL records from the serial interface and puts them in the CDPQ.
CIL record	Record delivered by the CIL in the PBX, containing information about calls.
Client Report	Reports containing information about articles performed by the clients.
Client Report Store	Store for Client Reports, containing information about articles performed by the clients.
Client Report Thread	Thread receiving the Client Reports from the clients and putting them in the Client Report Store.
Private Branch Exchange, PBX	Telephony exchange used internally by companies or organizations. Here refers to Ahhaaa's PBX used in the directory inquiry service and for other purposes.
Queue Length Monitor, QLM	Thread that monitors the lengths of the CDPQ and the CDTMQ, and reports the status of the queues to the TLUM.
SIU board	Serial Interface Unit board in the PBX, used for transmitting the CIL records.
TLU Manager, TLUM	Thread that supervises all the other threads in the TLU and sends termination request on termination.
Traffic Log Unit, TLU	The unit in the billing system that monitors the call data from the PBX, and other article data from the clients in the system.