"Television usage evaluation
through extended data mining
on logged usage data"

by

Alexandre D'Erman

August 16, 2007

Thesis submitted in partial fulfillment
of the requirements for the degree of
Civil Engineer in Electromechanics

Thesis supervisor:
Prof. Esteban Zimanyi

Academic year 2006-2007
Acknowledgements

While this thesis undoubtedly was an individual work, I would like to express my gratitude to Prof. Esteban Zimanyi who may feel he did not see much of me during the last twelve months, but who nonetheless has been of great support when help was most needed. I would also like to address a big thanks to my colleagues Jo and Annick for the free dutch lessons I was granted; I truly hope we can finish this project together. And last, but by no means least, I would like to thank all my friends and family who have supported me from the start on this year-long journey through unknown territory; I could not have made it without you.
Abstract

Data has always held a central role in strategic decisions, be it in the business field or any other. When analysts express a forecast over an observed tendency, they do so backed by their gathered data. This has prompted the start of a project at Philips Upmarket Televisions which aims at logging the consumer’s behaviour with respect to his/her television set. This implies that a post-processing environment had to be designed in order to retrieve useful information from the loggings. This thesis describes the steps that have been undertaken in order to refine the initial set of requirements. A whole post-processing environment was designed and implemented, combining C++ and PHP programming with database management and analysis, in order to answer all the questions that spawned this project. In the end, the model is tested with the help of a custom dimensional database analysis software, showing what great information the Philips decision-makers will be able to rely on for their future product upgrades.
Contents

1 Introduction 6

2 Presenting the subject 8
  2.1 The Importance of event logging : practical examples . . . . . . 8
    2.1.1 Event logging in airplanes . . . . . . . . . . . . . . . . 8
    2.1.2 Event logging in trains . . . . . . . . . . . . . . . . . . 10
    2.1.3 Event logging in cars . . . . . . . . . . . . . . . . . . . 10
    2.1.4 Event logging in computers . . . . . . . . . . . . . . . 11
  2.2 Event logging in televisions . . . . . . . . . . . . . . . . . . 12
    2.2.1 Goals - scope of the project . . . . . . . . . . . . . . . 13
    2.2.2 Implementation - television-side . . . . . . . . . . . . . 14
    2.2.3 Implementation - post-processing . . . . . . . . . . . . . 16
    2.2.4 Legal issues . . . . . . . . . . . . . . . . . . . . . . . . 16
  2.3 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    2.3.1 Business intelligence . . . . . . . . . . . . . . . . . . . 17
    2.3.2 OLAP reporting . . . . . . . . . . . . . . . . . . . . . . 18
    2.3.3 Data warehousing . . . . . . . . . . . . . . . . . . . . . 20
3 The complete solution

3.1 Describing the log: format and interpretation .............. 23

3.1.1 The log file ........................................... 24

3.1.2 The configuration file .................................. 26

3.1.3 Interpretation ......................................... 27

3.2 Requirements and constraints ............................... 29

3.3 Immediate answer ......................................... 30

3.4 Thinking it through......................................... 32

3.5 The complete system ...................................... 33

3.5.1 The television user ..................................... 33

3.5.2 The web site ........................................... 34

3.5.3 On the post-processing side ............................. 36

3.5.3.1 Why is a database needed? ......................... 38

3.5.3.2 How many databases? ............................... 38

3.5.3.3 Downloading and processing the files ............ 41

3.5.3.4 Analysing the results .............................. 42

4 Implementation ............................................. 45

4.1 Definitions ................................................ 45

4.2 Requirements - update .................................... 46

4.3 The DIMENSIONAL database ............................... 48

4.3.1 Designing the database ............................... 48
4.3.2 Filling the database ............................................. 56
4.4 The LOSSLESS database ........................................ 58
4.5 The ETL process - Introduction ............................... 60
4.6 The ETL process - Extraction (multi-language interaction programming) ........................................ 64
4.7 The ETL process - Transformation ............................ 64
  4.7.1 Which log files to process? ................................. 67
  4.7.2 Parsing of the log files .................................... 67
    4.7.2.1 Correcting the timestamps ............................ 69
    4.7.2.2 Filtering the RECORDMAP ................................ 70
  4.7.3 Parsing of the configuration file(s) ...................... 72
  4.7.4 Post-processing ............................................ 73
  4.7.5 Filling the databases ....................................... 77

5 Witnessing the results ............................................ 79
  5.1 Using the LOSSLESS database ................................. 79
  5.2 The dimensional database analysis software ............... 81
  5.3 Results ....................................................... 82
  5.4 A word on data mining ....................................... 83

6 Conclusion ....................................................... 90
List of Figures

2.1 Black box before use ........................................... 9
2.2 Black box after use ................................................ 10
2.3 Windows Xp crash ................................................. 12
2.4 Event generation and logging ..................................... 15
2.5 Chain of events .................................................... 15

3.1 Structure of a record ............................................... 24
3.2 Ambilight ............................................................ 29
3.3 The first answer ..................................................... 31
3.4 Registration web page for the project ............................. 35
3.5 The television menu that allows downloading the log file ... 36
3.6 Schema of the remote database ................................... 37
3.7 Basic schema of the environment .................................. 38
3.8 Tracking database .................................................. 40
3.9 Complete schema of the application ............................... 44

4.1 The Multimedia Content Browser ................................. 49
4.2 Schema of the DIMENSIONAL database ......................... 57
4.3 Example - filling the DIMENSIONAL database ................. 58
4.4 Log file naming ............................................. 60
4.5 LOSSLESS database - basic schema .......................... 61
4.6 LOSSLESS database - user profile part ....................... 62
4.7 LOSSLESS database - configuration part ...................... 63
4.8 LOSSLESS database - logging part .......................... 64
4.9 Schema of the Transformation and Loading application .... 66
4.10 Parsing of the log file ..................................... 68
4.11 The updating of the timeStamp .............................. 70
4.12 Correction of the records’ timestamps ....................... 71
4.13 Parsing the configuration file .............................. 72
4.14 Post-processing of the log file ............................. 74
4.15 Writing to the database .................................... 77

5.1 Query result on the LOSSLESS database ..................... 81
5.2 Dimensional Database Analaysis Software: choice of fact (left) - choice of dimension (right) ...................... 82
5.3 Time active in function of the hour of day ................... 85
5.4 Time active in function of the feature name, for the duration of the logging ........................................ 86
5.5 Number of button presses in function of the hour of day ... 87
5.6 Number of button presses in function of the button name ... 88
Chapter 1

Introduction

The distinction that needs to be made between data and information is well-known: a business cannot thrive on data, but rather on the reporting power it develops against its data. Decisions are made by people with experience and with solid reports in their hands. Knowing this, Philips Upmarket Displays wanted to build a post-processing environment to analyse logged user behaviour. This paper develops the answer to this problem in four parts.

- The first part exposes the history of event logging before enumerating for the first time the goals of the project and defining some key concepts to this work.

- The second part starts by describing the data (the log files and their interpretation) which will constitute the input of the whole environment. After showing how *flexibility* can sometimes be a powerful word that makes the difference between a simple solution and one that involves so much effort in its making. It is then shown how the initial requirements were bettered, and how the interaction between different modules was conceived rather than having one large application doing everything. These different modules include a web hosting, a local hosting, four databases and the all-important ETL process.

- The third part describes the practical implementation of the environment. This is where the two data warehouses are designed from scratch,
and the ETL process is programmed. All this is done via one last look at the specifications, in order to determine the exact shape and content of both datawareHouses.

- In the end, the fourth and final chapter closes the subject showing how well the dimensional datawarehouse can be used to obtain the results that were wished for.
Chapter 2

Presenting the subject

2.1 The Importance of event logging: practical examples

These few paragraphs will assess the importance of event logging in various fields, by examining through some practical examples what motivated its implementation and how expectations have been met.

2.1.1 Event logging in airplanes

In 1957, David Warren, a young Australian chemical scientist with a strong interest in electronics, conceived a device that would record conversations in airplane cockpits and store them in a crash-proof container. This device answered the need to understand the reasons behind plane crashes, a need that arose during investigations on a series of accidents involving the de Havilland Comet, the first jet-powered commercial airliner. Dr David Warren imagined introducing a recorder in each flight, with the possibility to recover it after a crash [1]. His demonstration version of the recorder, which could store four hours of speech in addition to snapshots of instrument variables, was met with some scepticism in his home country. Fortunately, that same
This invention has since developed into what is commonly called the ”Black Box” (despite its bright ”security red” color), a device that can store up to 25 hours of flight parameters and voice recordings in view of analysing the possible causes of a crash.

A recent plane accident which killed all 121 people on board was explained thanks to the retrieval of its black box. In August 2005 a Helios Airways plane on its way from Cyprus to Athens drew circles during two hours above its landing point, and then crashed because of fuel starvation. A problem in cabin pressurisation was deemed to be the cause of the crash, with all passengers and crew members supposedly unconscious. Analysis of the recordings showed that two of the crew members managed to stay awake using portable oxygen bottles, but they were refused access to the cockpit because of the locked ”terrorist-proof” door. Such an analysis could prove invaluable in the building of future aircrafts, in view of increasing measures so that the same mistakes cannot be made twice. [2]
2.1.2 Event logging in trains

Such devices are also used in trains. Train event recorders were first used in Switzerland, in the 1890’s, when the world’s first speed recorder was created, the "System Hausshaeleter". That mechanism kept track of the train’s speed and path in function of time. Nowadays, these crash-proof, fire-proof and nearly-everything-proof logging devices store up to 48 hours of safety-critical data in a flash memory device. Such data includes speed measurement, distance traveled, and use of throttle, brakes and headlights.

The gathered loggings are primarily used in case of accidents (or minor worries), but this does not rule out the monitoring of the driver’s actions for evaluation. For example, in july 2006 in Valencia an underground train derailed, killing 41. The black box was retrieved, and the analysis of its data suggested that the driver was going twice as fast as he should have, thus ruling out a terrorist attack. [3]

2.1.3 Event logging in cars

After the success of the integration of black boxes in planes and trains, it was just a matter of time before someone thought of fitting them in individual vehicles. In cars and lorries, however, the famous black box is called an
Event Data Recorder. It is based on the Sensing and Diagnostic Module, the component that determines when the car is having a crash, and when the airbag must be deployed (based on acceleration peaks). In 1999, car manufacturer General Motors upgraded its SDM module to include loggings of the vehicle’s speed, engine RPM and the position of the pedals, plus some internal status variables (oil level, ...). Since then, other manufacturers such as Ford or Peugeot have followed.

The initial scope of the project was to ease the detection of problem areas in a company’s cars. Lately, however, these same loggings have been used as evidence in a series of trials in the United States, which has sparked a debate about who really owns such recordings and whether their use is an infringement of privacy.[13] It has to be said that the projected use of such devices in all cars has in some cases completely degenerated: some think they could be used to integrate gps-based speed limiting (so that a car could never exceed the legal maximum speed), while others push for an automatic enforcement of speeding through the monitoring of the events recorded in every car: if Mr. X drives too fast while going back home from work, a fine would be waiting for him in his mailbox. Science fiction? [6]

2.1.4 Event logging in computers

Lastly, a much more apparent form of logging is the one used in software for debugging. Any program, however simple, must provide some form of output, so that the programmer can understand if his software’s behaviour is the one he was hoping for. The output can be made directly visible, but is more generally stored in what is commonly called a log file, with a "log" extension. With the ever-increasing use of the Internet through broadband connection, programmers became aware that information could be gathered after development, directly from the user. It is a well known fact that when a program crashes on a Windows XP platform the user is prompted to send information to Microsoft in order to add information to a database with all known issues, in view of solving the most recurring problems (which, of course, they never do).
It will be noted, however, that the user is informed that his contribution will remain strictly ”confidential and anonymous”. As in the previous example of logging in cars, the legal aspect of this procedure is very important, because the user still has the choice to refuse to be involved.

After having assessed the importance of event logging in various aspects of current technology development, we are able to define the project better: its goals, its implementation, and lastly the legal aspects of carrying out such a task.

### 2.2 Event logging in televisions

Logging is all about gathering information during runtime. In the field of television design, such information is very useful for evaluating the user’s experience of the product as well as for tracing possible bugs in its software and failings of its hardware. Historically, however, user experience has always been evaluated within the company walls (in test rooms), thus never providing a true understanding of the customer’s behaviour with respect to his own television set. Furthermore, software development has never received any input from the users, because all debugging and optimisation is done locally.

Indeed, once a product is tried and tested at hardware and software level, and once its security features have proven to be effective, it can be commercialised. This overly-simplified vision of a product’s validation steps leads us straight to user-testing. The Customer Experience Center (CEC department) at Philips Bruges contacts people who are affiliated with the company,
asking them to spend half a day testing a television: unpacking, wiring, installation and usage (note: popcorn not allowed). During that time, the users are being filmed and timed, the mistakes they make being the most valuable data the company can gather. CRs, or Change Requests, are then generated based on these observations, and hopefully the product can evolve.

Never, however, has a user been observed whilst coolly watching television in his own sofa. Obviously, his behaviour would then be radically different. He would not be asked, like during testing, to perform a predefined task, thus trying to focus on the task at hand. He would more probably just crawl his way through the menus, making every possible mistake before finding the option he wants, and maybe even uncover bugs in the television's operating system in the process. Such a behaviour is not reproducible in a testing room, although being much more interesting than the routine mentioned earlier.

This is all about to change, as the CEC department at Philips has defined the first few objectives of an ambitious project that aims at providing the company with invaluable data collected directly during the customer’s use of the product. The following paragraphs will describe the goals of the project, and what structure had to be put in place on both sides of the project in order to achieve them.

2.2.1 Goals - scope of the project

As has been described earlier on, event logging is about fetching information where there is most of it - at the user’s, when the product is in use. There are two main applications for this: firstly, the logging of all events within a product (internal events and inputs) can be used to track down the causes of errors which can present the end-user with an unpleasant experience. The major concern for a company building a product, considering the ever-increasing quantity and complexity of the hardware and software used, derives from the fact that a single person (or even a debugging team) can never be expected to uncover every existing problem. The most important ones need to be solved so that the product can be commercialised, but some little quirks never show up until the product is used by the client, hence the importance of event logging. The second objective, which is perhaps even more productivity-oriented, is to acquire a better understanding of the
consumer’s needs to a point which is otherwise not possible. By analysing
the logs, the company can see where it went wrong in predicting the user’s
behaviour or his needs. The success of the introduction of a new feature can
thus be evaluated based on its use in real-world conditions, not any more
only in a testing lab. The same can be said of the remote control: it could
appear, after analysis, that certain buttons are almost never used by anyone.
Why keep them and clutter the remote with useless buttons?

As the logging in Philips televisions is being implemented at the time of
writing, the software layer having been built and about to be included in
a small number of test sets, the job at hand was to design and create a
software analysis environment for the post-processing of the user loggings.
This environment would enable the Customer Experience Center at Philips
Brugge to gather intelligence about the aforementioned objectives, and thus
make strategic decisions based on such results.

2.2.2 Implementation - television-side

The television set is basically a piece of hardware, plugged into a power socket
and an input signal socket, and managed by an operating system. This OS,
codenamed Jaguar within company walls, is responsible for the handling of
all external events and internal calls to the modules of which it is composed.
Every key press and every new external connection is ”caught” by an Event
Listener, and the Event Handler launches the appropriate command, be it
the opening of a menu (as when a USB device is plugged in) or a program
change (when a cassette is inserted into a connected VCR). The OS also
handles warm reboots which happen in the background. It would be inter-
esting to know how often these happen during normal use. Certain button
combinations may induce software crashes unbeknownst to the user and to
the software programmers, and this is part of the information Philips would
like to have access to.

In the modified (i.e. the one that enables logging) version of the software,
all these internal and external events are stored into the on-board memory.
Whenever the user plugs an external USB device, he is given the possibility
to upload the current log file, after which the internal memory is flushed and
the logging restarts.
In some way or another, the log should then be sent to Philips in order to be integrated into the post-processing.
2.2.3 Implementation - post-processing

No particular environment was imposed. The only request was that the environment needed to be flexible. As I was soon to learn, this was in fact the most difficult aspect of the program, and also the more restrictive.

Having never programmed at such a level, and having no experience whatsoever in the interaction of multiple languages, I had no criteria to help me choose the software and the environment I should use. Luckily, one lifeline was thrown at me just as I set on my quest: Jo and Annick informed me that the cheapest solution would be the best one, because the budget for the running calendar year had all been spent (on other projects, obviously). This eased quite a lot the research that I was about to make, as I thereby chose to investigate only the cheapest of them all: open-source software, preferably looking at well-established, well-supported and well-documented solutions.

Of course, this choice had its downsides, the main one being that open-source software is primarily built for Linux, thus my working on a Windows OS machine would bring along its fair share of mandatory troubleshooting. Furthermore, the very existence of open-source is based on the cooperation of many people, most of whom are unfortunately programmers. This leads to very mature software which can compete with its commercial alternatives, but which most of the time lacks documentation and ease of use. The quality of the documentation will thus be a big factor in the choice of the software that I shall use.

The final solution and the reasons that led to it will be explained in Chapter 3.

2.2.4 Legal issues

Obviously, this project as a whole becomes interesting only when the number of users participating is big enough. An analysis with too few people involved would be influenced by specific user behaviour, and would not return statistically interesting results.
... so why not introduce logging in all televisions? This would generate an enormous amount of information which could lead to a decision-maker’s heaven.

Apart from the technical demands that such a project would imply, the problem is mainly of a legal nature. Users must be told that their behaviour will be logged, which would probably scare them off towards the competition. A contract tying the company to the user must be signed by both parts. One one side, the user agrees to actively participate in the project, where “actively” means that he is encouraged to ”actively” watch television as much as possible, and every once in a while plug in a USB stick and send the generated file to Philips. On the other side is the company, who certifies that all collected information will remain strictly between Philips walls, and will be used solely for product enhancement.

2.3 Definitions

This section will define a few words that have become rather trendy in these times when data explosion has largely surpassed man’s ability to make good use of such enormous quantities of data. Words such as Business Intelligence, OLAP reporting and Data Warehousing have become commonplace in any activity that tries to make sense of large quantities of data (and it seems to be our case), which is why it might be of interest to actually be able to define them adequately.

2.3.1 Business intelligence

This generic term is used to describe a series of concepts, technologies and software tools that permit the gathering and the subsequent analysis of data in view of increasing a business’ intelligence. This intelligence is in fact a company’s awareness of what happens in its commercial scope (including competitors, legal framework and customer behaviour), as well as its internal functioning (employees’ wages or production tracking, for example).
It should be noted that it is not the abundance of data that benefits a company, but rather the amount of useful information that the BI environment can extract from such data. In the era of "data explosion", it becomes increasingly difficult to draw the thin line that separates these two concepts, but that is what BI software is built for, and analysis experts are trained for.

Whether such knowledge is really helpful to support decision-making is a debate that has been going on ever since these concepts were born. Some say that business intelligence is primarily used for "post-decision monitoring of the effects of decisions". Others assert that the simple fact of compiling data from different data sources (which is more often than not necessary for gathering meaningful information to support decisions) means data is converted into a "lowest common denominator consistent set", thus losing much of its appeal. Even its detractors, however, cannot argue about the fact that Business Intelligence provides a way to aggregate different sources in a way which was not possible even ten years ago, and that it sees ever-growing competition between established commercial heavyweights (IBM, Oracle, Microsoft) and open-source newcomers (Pentaho, BizGrez, OpenI) reach an all-time high, much to the benefit of the end-user.

One thing should always be kept in mind, however, when implementing a BI framework: in many cases in which the planning was not done correctly BI solutions cost more than the benefits they generate. The learning curve is not only steep, but also quite long.

### 2.3.2 OLAP reporting

OLAP (OnLine Analytical Processing) is a term used to describe solutions that enable fast, powerful and efficient reporting on large quantities of data. It generally is implemented as a layer on top of one or more databases.

The OLAP Report produced a series of criteria that provide guidelines for evaluating OLAP tools, without giving details about how to achieve these particular goals.

- Fast: Meaning that the majority of queries should be answered within 5 seconds, with all but the most complex calculations taking less than 20
seconds. Indeed, it has been observed that when a query takes more than 30 seconds, the end-user tends to think that the system has crashed, and aborts the calculations. Achieving such speeds is up to the software vendor, but generally this is done by pre-calculating a great deal of the data that can be analysed. Of course, full pre-calculation becomes impossible when the amount of data grows, and in that case the software needs to adapt by optimising its use of free storage space.

- Analysis: Means that the software should be able to generate whatever report or analysis the user wants (within its designed capabilities). If this is done by giving the user a way to define custom calculations, it must be implemented in a user-friendly manner without involving any programming skills.

- Shared: Means that multiple-user access to the data should be possible, without sacrificing confidentiality at any level. Furthermore, if write-access to the data is introduced, data integrity should be guaranteed. Security is said to be the main weakness in most OLAP tools.

- Multidimensional: The main requirement for an OLAP tool. Indeed, the most natural way of navigating the data is along different dimensions (time, product type, region) in a hierarchical manner (year, month, day, ...), as opposed to the relational model upon which transaction databases are conceived. Thus, the user must have a multidimensional conceptual view to work on.

- Information: Many guidelines are defined under this name, which mostly reflects the fact that an OLAP tool must give access to large quantities of information without sacrificing power nor integrity. It should be known that The OLAP Report introduces a series of considerations that can be taken into account when creating a software, among which are the amount of data duplication, memory needed, disk space usage optimisation and integration with data warehousing (a concept that will be described later on).

With these five words, the OLAP Report aims at defining the goals that are meant to be achieved by any OLAP application.
2.3.3 Data warehousing

This term has been used in the previous paragraph, in the section about Information. Indeed, data warehousing is all about making information accessible to querying/reporting tools, whatever its format.

A data warehouse is "a collection of subject-oriented, integrated, non-volatile, and time-variant data to support management’s decisions".[17]

- Subject-oriented, meaning that the data in the database is organized so that all the data elements relating to the same real-world event or object are linked together

- Time-variant, meaning that the changes to the data in the database are tracked and recorded so that reports can be produced showing changes over time

- Non-volatile, meaning that data in the database is never over-written or deleted, once committed, the data is static, read-only, but retained for future reporting

- Integrated, meaning that the database contains data from most or all of an organization’s operational applications, and that this data is made consistent.

Any company larger than your average family business has to keep track of all its transactions. This is usually made possible by using an OLTP system (OnLine Transaction Processing) optimised for transaction speed and heavily normalised. A system built for transactions, however, will not be at its best when containing too much information. To query against it for a massive report involving hundreds of tables is not a good option, all the more if another user tries to write to it while such calculations are processed. Also, within a company many databases can exist simultaneously: local retailers’ databases, central office database containing records for all web site users, employees database, and many possible others. It is much more useful to merge them all for reporting tasks.
This has motivated the introduction of centralised databases that are called data warehouses. These generally hold all of a company’s data, and are built on specific hardware and storage models, pretty much useless for transactions but designed for querying and reporting. Every once in a while (yearly, monthly, weekly or even daily), the company’s transactional databases need to be emptied in order to remain speedy enough. Their data is then transferred to the data warehouse. Data from different sources must be extracted, then cleaned up, and lastly loaded into the warehouse. This process is called Extraction, Transformation and Loading (ETL). In many cases, whilst extracting and loading requires only a knowledge of the source and destination databases’ structures, the interesting part is the data cleanup. Indeed, wrong information is a major source of data misinterpretation, and very difficult to spot (let alone correct it).

More important is the form of a data warehouse. It can be a relational database or a dimensional one, each solution having its own limitations. Dimensional databases are often used in data warehousing because their manipulation by the end-user does not require specific knowledge about database structure or about programming. Indeed, the dimensional approach is often cited as being the closest to the way we intuitively see the world. Furthermore, its two main designs, the star and the snowflake, enable much pre-calculation to boost run-time performance. The star schema is the simplest form of data warehouse design, where a lone fact table is connected to each dimension table and also contains the useful data. The snowflake modifies this model while remaining pretty close to it: the dimension tables are normalised, and as such are separated into additional layers (the internal one being directly related to the facts table).
Chapter 3

The complete solution

After having examined the benefits of logging in the design of televisions and of television software, this chapter will expose the solution that has been chosen for fulfilling the goals. In order to ensure that the final solution takes into account all the different aspects of the project, we will first describe the way a log file can be read and translated, after which we will examine in detail the requirements and constraints that had been defined when the project was started. An in-depth analysis will then show how the requirements can be translated into some functional constraints. Lastly, the complete schema of the environment will conclude this chapter. Also, it is important that the analysis remain as platform- and software-independent as possible.

3.1 Describing the log: format and interpretation

This section will explain a log’s structure, and the way it can be read and translated into a readable format. This is essential in order to understand what can really be done with the gathered information.
3.1.1 The log file

As was explained in section 2.2.2 on page 14, a log file is progressively filled with events caught by the event listener built in the television software. Of course, the interesting part is not really the file creation. What enables us to examine the customer’s habits is the extraction of the data that is contained in that file.

First of all, because the log file is encrypted during the transfer to the USB external drive, the file must be decrypted so that the pre-defined format can be recognised during parsing. The encryption algorithm used is the Tiny Encryption Algorithm (TEA), so called because of its extreme speed and efficiency. This algorithm, developed by David Wheeler and Roger Needham of the Cambridge Computer Laboratory, also has a very small footprint and limited overhead, thus making it ideal for embedded systems. Its implementation in different languages is completely licence-free.[18]

After decryption using the freely distributable routine exposed on source [18], we end up with a decrypted file, ready to be parsed and analysed.

A log file is a series or records, where each record is composed of a 10 byte fixed length part, followed by a variable length one that contains eventual parameters, with a couple of bytes allowing for synchronisation preceding the record. The exact structure is as follows:

- The timeStamp indicates the number of seconds that have passed since 01/01/1996, midnight.
• The eventID is, as its name implies, the identifier of the event that the software has detected at that very moment.

• The parameter type indicates the number of bytes that must be read for each parameter value. When the parameters must be read as characters, this byte reads 0x01, while it reads 0x04 for integer parameters (thus being 32 bit integers).

• The parameter number is simply the number of parameters that must be read before the checksum.

• The checksum is calculated with a bitwise exclusive OR on all bytes in the record.

Here are three consecutive records taken from a real log file:

| record 1 : | FE EF F5 92 55 14 00 00 03 00 04 03 ...   |
| TimeStamp : 0x145592F5 |
| EventID : 0x00030000 |
| ParameterSize : 0x04 |
| ParameterNumber : 0x03 |
| Parameter 1 : 0x00000003 |
| Parameter 2 : 0x00000000 |
| Parameter 3 : 0x00000528 |
| CheckSum : 0x32 |

| record 2 : | FE EF F9 92 55 14 00 00 03 00 04 03 ...   |
| TimeStamp : 0x145592F9 |
| EventID : 0x00030000 |
| ParameterSize : 0x04 |
| ParameterNumber : 0x03 |
| Parameter 1 : 0x00000003 |
| Parameter 2 : 0x00000000 |
| Parameter 3 : 0x00000440 |
| CheckSum : 0x46 |
### 3.1.2 The configuration file

So far, all we get in reading a log file are numbers, each with their own timestamp... not very useful. What we need is a reference, a lexicon that would translate these numbers into meaningful information.

This is the configuration file, which is generated along with the television software. It looks like an XML file but it is not, which is why the parsing is not as easy as it could be (or should be). It contains every event that can be raised during television operation, but also all its possible parameters. The file is organised in four main groups of information, separated by start- and end- flags:

- The release information part, which contains the version of the television software which corresponds to that particular configuration file. The flag is RELEASEINFO.

- An enumeration of all the groups of events. For example, all events related to the TeleText feature are part of the ENUMGROUP "ceaptxt". The flag is ENUMGROUP.

- An enumeration of all possible integer parameters, grouped by ENUMGROUP. Indeed, parameters are not defined for single events, because all events within a certain ENUMGROUP share the same parameter definitions. The flag is ENUM.
• An enumeration of all the possible events, also grouped by enum-group. The flag is event.

Here is an example, with some lines taken from a real configuration file:

```
<RELEASEINFO>
  ITEM( SoftwareId , "JX31E-0.0.48.0.2" )
</RELEASEINFO>

<ENUMGROUP>
  ITEM( ceapamb )
  ITEM( KEYGROUP )
</ENUMGROUP>

<ENUM>
  ITEM( ceapamb , 0 , "AmbColour" , 2 )
  ITEM( KEYGROUP , 0 ,"keySourceRc6" , 3 )
  ITEM( KEYGROUP , 2 ,"keyStepUp" , 1088 )
  ITEM( KEYGROUP , 2 ,"keyAmbLightMode" , 1320 )
</ENUM>

<EVENT>
  ITEM( 0x104b9 , propids_Amb AmbientModeList , 1 , ...
       ... ceapamb , 1 ,"To Set the Ambilight Mode" )
  ITEM( 0x30000 , Keys , 1 , KEYGROUP , 1 ,"The Key Pressed " )
</EVENT>
```

3.1.3 Interpretation

By combining this information with the three records that were described earlier on, we end up with the correct translation of a log file’s records. Table 3.1 illustrates the results.
From what elements can be interpreted (it can be noticed that the second parameter of the keypresses cannot be translated, because it is not defined in the configuration file), these three consecutive records read as follows:

On 23 October 2006, at 12:35, the user pressed a key on the remote (the RC-6 protocol is a standard that is used for remote controls), namely the key AMBLIGHTMODE, which means that the user probably wanted to change the way the Ambilight function behaved. A menu must have popped up, although it does not show here. In future versions of the logging software the corresponding menu pop up should also trigger an event. Once in the menu, the user pressed the key STEPUP, thus changing the selection, which triggered a change of Ambilight Mode to COLOR.

**Note:** Ambilight is a functionality present exclusively on Philips high-end sets which generates a halo of colored light around the screen, lessening the strain on the viewer’s eyes (especially in a dark room), and giving the impression of a larger picture. Different settings are available (DYNAMIC, RELAXED, ...), each providing the user with a different experience. For example, the DYNAMIC setting is much more aggressive and has much stronger colours, thus being more adequate for action movies than romantic comedies.
3.2 Requirements and constraints

The basic requirements, as they were defined at the beginning of the project, were as follows:

- A program should be written (for example in C or C++) to parse the log files as they are retrieved from the users’ televisions.
- The program should provide us with information on the interaction a user has with his television set.
- The post-processing should output a file containing all the information in columns, and one line per television set that delivered the data (i.e. something that could be exported to an excel spreadsheet).
- Useful information would include: how does the customer change channel (zapping, favourites menu, digit keys), does he use teletext, EPG, Content Browser? Other important information would be related to pattern-recognition: how often do users enter in particular sub-menus without ever doing anything in them?
- The software should also provide adequate filters, so that the output represents only meaningful information. For example, on a second-level
analysis, the inclusion of the keys Next Channel, Previous Channel, Volume Up and Volume Down should be avoided, because we know very well that they are the most used keys.

- Lastly, the analysis environment should be flexible, meaning that if new features are implemented into the television’s interface, they should appear in the results with little or no modification to the post-processing software.

### 3.3 Immediate answer

With such requirements in mind, a first solution would be to implement, as it was originally planned, a C++ program that would enable parsing the log files, filtering and post-processing the recorded data, and that would then output the information to a text file. Exactly as was imagined at the beginning of the project.

This program would take a logfile as an input. The log file would first need to be decrypted by reversing the aforementioned Tiny Encryption Algorithm. After decryption, the file could be read record by record. The parsing of the file implies that its records be stored in order to be translated. The post-processing of the records implies the existence of efficient algorithms that, based on series of records, would be able to build information that can be outputted. There can be a filtering layer directly after parsing to remove unimportant data before processing it - not all records are useful for our purposes. There could also be a second filtering layer after post-processing, to only keep the information we want out of all calculations that will have been made.

Typically, a Graphical User Interface could give the user of this software the possibility to filter information according to the reports that he needs.

This approach would not be very difficult to implement, once all the information that we need at the output is well defined. However it has some major flaws, all basically expressed with one single word: flexibility. The only true
Figure 3.3: The first answer
constraint that was formulated is in fact a casualty of such a design. Indeed, if everything has to be coded inside a single program, all algorithms must be included so that the end-user can choose between different pre-defined filters. If the analyst ever thinks of something that should be of interest to him and that is not covered by the actual software, the programmer is in for some patching which could take him days before the result can be used. With this schema, there is no such thing as a configurable report.

This is the reason why alternatives to this direct route must be examined. The following sections will address this more in detail.

3.4 Thinking it through...

After a basic analysis of the implications of the main constraint - flexibility - the requirements can be redefined, or at least bettered. What the analyst should be given is an environment that takes as an input the log files, and then allows him/her to freely select the information he wants, as easily as filtering the one he does not need. Also, it would be very useful if he/she could benefit from the information contained in multiple logs at once. This means that aggregation of data should be made easy. For example, the choice should be given to aggregate data based on time (what is the general behaviour, for all users, within a certain time frame?), or even based on a certain user characteristic (age and family situation could yield very interesting results). Thus, the software should definitely be a Graphical User Interface, and should allow for:

- data filtering
- aggregation of data
- user profiling
- pattern recognition

Keeping in mind that all of this should be left to the analyst, and the programmer should be as useless as possible once the software is finished. The
only thing that the analyst should do is simply select the information he needs, and then report on it. The output should be graphical, and allow for fast interpretation of the results, for example based on charting capabilities.

Now that the requirements are well-defined, we can start finding answers to this complicated problem.

3.5 The complete system

In order to fully understand what the full-blown environment should look like, it is important that the project be examined as a whole, starting from the television user, and straight up to the analyst.

3.5.1 The television user

The first aspect of the project is making sure that the user sends regularly, in a way or another, his log files. Indeed, without raw material even the most sophisticated of frameworks would find it difficult to generate results. In order to ensure that the user does not progressively "chicken out" of the project, the only solution is making it as easy as possible for the customer to participate. Many solutions have been examined by the CEC department, and the one that has been chosen is the most sensible one: a web site where the user can easily upload his files, easily being the keyword, and then be granted a smile and a big "Thank you for participating!". Discussions have also led to the web site being a place where the user can download the latest television software for his/her set.

In order to keep track of the users that will be part of the project, and also to enable a certain amount of user profiling in the post-processing, users will be required to enter personal data. The following attributes have been deemed useful for the current project (a screenshot of the registration page of the web site is shown on figure 3.4):

- contact information (name, address, phone number)
- country
- age
- profession
- number of adults at home
- number of children below 12 and between 12 and 18 (two very different behaviours)
- number of televisions at home
- devices (usb key, digital camera, PDA, ...)

In the end, this makes it rather simple for the customer. After having registered on the web site, he/she is granted access to the television software’s download page. Some simple instructions explain what to do next: put the new software on a USB drive, install it on the television, enter in a sub-menu within the television interface to fill in the userID that he/she has been given. This activates the logging module on that particular set, and the user can lean back and start zapping. After some weeks, the on-screen display should read ”memory full”, which means that a USB drive can be used to download the generated log file. This flushes the internal memory, and the user can log on the web site once more, and upload his file.

### 3.5.2 The web site

Once the logs are uploaded, they are stocked on the online (remote) server which will serve as temporary storage space before being downloaded one by one, on a regular basis, to the local server. Whenever a user uploads a log file, a new entry is also added to the database hosted on the server. This database must keep track of all the different users that are part of the program. Each user can have one or multiple sets registered to his name, but one set can
Figure 3.4: Registration web page for the project
Figure 3.5: The television menu that allows downloading the log file

relate to only one user at a time. Each set is uniquely defined with the combination of its model ID and of its serial number. Lastly, every television set will generate a certain amount of log files, which will be stored in a table called tvLogs. This table will also need a column called isDownloaded whose default value is FALSE (or rather zero), in order to keep track of the files that will have been downloaded from the web server to the local server.

The remote database’s schema is illustrated on figure 3.6.

3.5.3 On the post-processing side

This is the part that this paper is about: once the file is uploaded, how can it be retrieved and processed to yield some results? The basic structure of the post-processing environment should look like on figure 3.7: fetch the files from the remote server, then process them, and finally output the results.
Figure 3.6: Schema of the remote database
3.5.3.1 Why is a database needed?

Now comes the hardest part: how is it possible to fulfill the requirements in the best of ways? It has already been proven that storing the results of the processing task in text files is really not a brilliant solution because it lacks flexibility. The ideal solution would be to store information through a system built for large amounts of data, whilst keeping it accessible in a structured and efficient way. This is typically where a database would come in handy. Indeed, a database can be defined as a "collection of information of data organised so that its contents can easily be accessed, managed and updated" [14], and is built to keep large amounts of data accessible. A database is managed and made accessible by a Database Management System (DBMS), which also allows for data aggregation and filtering, our major requirements. Furthermore, if designed correctly, a database can easily be analysed with custom-made software or with existing database analysis software (see OLAP, in section 2.3.2 on page 18).

3.5.3.2 How many databases?

After having chosen to use a database management system to store all processed data, it is important to define what the number and the shape of the database(s) will be.

**One - the tracking database**  First of all, a local database needs to be copied on the remote one for its structure. Indeed, once the files have been
downloaded from the remote server they need to be stored and kept track of. Only one slight modification is necessary with respect to the remote database’s schema: the aforementioned \textit{isDownloaded} column needs to be changed to \textit{isProcessed}, indicating if a file stored locally has already been imported into the results set by the parsing software. This software would then only need to identify the stored files for which \textit{isProcessed} equals zero by querying this database, and start parsing them. This database will be called the \textbf{TRACKING} database, because it keeps track of all downloaded files. Its schema is shown on figure 3.8.

\section*{Two - the dimensional database} Now on to the results. It is a well-known fact that the easiest way - or at least the most natural way - to analyse data is through a dimensional model. All-the-more for someone who should not be bothered with all the knowledge that entity-based databases require. In fact, as has already been said in section 2.3.3, the star schema is the most frequently used model because it is intuitively much easier to comprehend than the classical entity-relationship model. Therefore, a star-schema database will be designed for the storage of processed logging data, so that analysis and reporting can be completely separated processes. If we are to build a flexible analysis solution, we must make sure that the database upon which the analysis software will be based is very well designed, according to specific requirements. Thus, a further update of the requirements is necessary in order to identify the correct facts and dimensions to integrate into our database. This will be developed in detail in section 4.3.1.

\section*{Three - the lossless database} It must be said, however, that the use of a dimensional database will result in an inevitable loss of information. Typically in a logging file, to extract some information about the usage of a certain button we might have to consider consecutive records in a file, and translate the information that they carry into one simple phrase: "between 9h00 and 10h00, on a sunday morning, button number 3 has been pressed 5 times". Clearly, the event sequence that has led the user to press the button number 3 five times has dissapeared altogether from the analysis. Maybe the button was pressed five times in a row because the user could
Figure 3.8: Tracking database
not manage to succeed in what he/she was trying to achieve, and we have no way of knowing except by going back to the log file. Algorithms that aim at detecting patterns, software failures or user mistakes need another source of information than the dimensional database we have focused upon until now. Such information should be made available easily, and for as large a number of files as possible. What this means is that, although algorithms can be written to implement pattern recognition on a whole batch of files, it would not be nearly as practical as having a central repository (hence the use of another database) for all records coming from all log files, along with the event translations (taken from the configuration files) for ease of analysis.

This, of course, cannot be done with a simple star schema, however many dimensions we plan on using. A true entity-based database is needed here, meaning that this database should have a structure copied on the one that was imagined for the project as a whole. For example, it should include a table containing all users, another one containing all sets, a third one with references to all logs, but also other tables with everything that is defined in the configuration files: corresponding events, parameters, and translations. Only with such a database structure, rooted on the original data, is it possible to keep all original information intact. Its final format will be examined in section 4.4.

**And that makes three** The local server should therefore contain three separate databases. The process of fetching the files on the remote server and processing them to push information into the various databases will be examined in what follows.

### 3.5.3.3 Downloading and processing the files

Once the structure of the databases is known, the process of transferring (nightly, for example) the information from the remote server to the local data repository can be introduced. Basically, this is done through the following steps:

- First, the remote database must be queried upon, to find out whether there are files that must be downloaded (remember `isdigitDownloaded = 0`).
• If some file names are returned, a transfer process can be launched from the local server, in order to copy the files in the local directory tree.

• Then, once these files are downloaded to the local server, their entries in the remote database must be marked as downloaded (isDownloaded = 1).

• At the same time, the local TRACKING database must be updated, with the addition of new data: new entries for the downloaded log files are necessary, but maybe these files were uploaded by new users or were generated by new television sets, which would involve the creation of corresponding new rows in the database. Note that the isProcessed flag for the new files must be set to zero for what follows.

• Finally, all downloaded-but-not-yet-processed files can be decrypted, parsed and imported into the two local databases (the DIMENSIONAL and the LOSSLESS). After this is done, their isProcessed flags must be set to 1, and the server can go to sleep.

Note that these few steps are exactly what is called ETL: extract data from where it is originally, then transform it so that it serves our purposes, and finally load it into the data warehouse. The exact implementation of all this will be discussed starting from section 4.5.

3.5.3.4 Analysing the results

We now have two separate sets of results in the shape of two completely different databases. Given their differences, they should be analysed in different ways and most probably by different people altogether. The main strong point of our choice to use a dimensional database as a primary source of results is that any dimensional database analysis software can be used, be it a commercial solution, an open-source one, or even a custom-built, graph-outputting piece of software. Thus the whole environment is made of separate modules that simply interact with each other, which is a huge step forward from the starting point: we have evolved from an all-in-one solution with decrypting, parsing, translating and analysis all grouped into a single
application, to separated parsing and analysis allowing for much improved freedom.

In the end, this chapter has shown the importance of a proper analysis of the requirements taking into account the knowledge of the true objectives and of the techniques involved in this project. These requirements have been redefined, and this has led to the subsequent evolution of the solution proposed to a full-fledged environment. The following chapter will explain the implementation of the solution, starting from the choice of software and programming environment, while always keeping in mind that the objective is to have a solution as flexible as can be.
Figure 3.9: Complete schema of the application
Chapter 4

Implementation

This chapter is the one where everything that we have determined up until now will be merged, so that the complete solution can finally be implemented. We will first go through an update of the requirements taking into account the conclusions of the previous Chapter. Then, the complete and final design of both the DIMENSIONAL and the LOSSLESS databases will be justified. Finally, the full ETL process will be described in detail, and with it the implementation part of the project will be finalised.

4.1 Definitions

Before going any further, however, two definitions will be given in order to understand better how information will be stored in the DIMENSIONAL database:

**feature**: a state in which the television software can be, like the teletext, the content browser, or even the standard feature called *onscreen display* (the one that displays the normal images, while enabling the occasional volume bar or channel information panel). There is always at least one feature active at a time. The television software is composed of modules that are activated only when they are called upon. Some can be stacked (an example of stacked features is the Picture In Picture feature which
needs several layers). When multiple features are stacked, only the topmost feature will be stored in our database, meaning that if feature A is activated on top of feature B, we will store the time that A has been active.

**source:** the normal signal source is TV. When peripherical components like a VCR, a DVD player, or even a PC are connected to the television, however, the user needs to change manually the source signal to the one he/she wants to be relayed onscreen. There is always one (and only one) source active at a time. In the log files, a source is defined by 6 parameters.

### 4.2 Requirements - update

The first thing to do, once every element of the chain has been carefully selected, is to ask the CEC department what information is needed from the dimensional database, because the questions to be answered will determine entirely its design. Up until now, the requirements were left a bit foggy (on purpose), so that the design of the environment as a whole was not slowed down by an excess of details. Now that the general structure of both the analysis databases is known, the requirements can be made more specific and, more importantly, can be made definitive. Thus, after an in-depth look at the possibilities granted by extended logging, the following questions are some of those that will need to be answered by a simple query of the dimensional database:

- **TV set activity:**
  - How is the TV switched on, and how frequently?
  - How long is the television on everyday?
  - At what time of day is it more often active?

- **Button presses:**
  - How often are buttons pressed, for remote control buttons and local keyboard (the keys attached to the side of the set)?
- What is the user’s preferred way of changing program (zapping, favourite channel list, or digit keys)?

- Features:
  - Which features are most used? Which ones are never used?
  - How long is a certain feature active with respect to the total watching time?
  - What are the most frequent button presses within a specific feature?

- Teletext (even though it is a feature, it requires special attention):
  - Number of pages watched per session?
  - Number of subpages watched?

- Multimedia Content Browser (another important feature):
  - What is the most viewed content type (photo, video or music)?
  - What file types are most viewed (jpg, mpeg, mp3 ...)?
  - How deep is the folder tree that people use on their removable drives?

- User profiling:
  - Which attributes are decisive for user clustering? Can we make tendencies appear in the link between specific attributes and a general watching behaviour? although this aspect of the analysis bends strongly towards data mining.

- Ambilight:
  - Is it used a lot, or do people grow tired of it?
  - Is often switched on/off, or do people leave it as it is once they have configured it properly?
  - Which mode is most used (Dynamic, Moderate, Off)?
• Sources
  ◦ Do people use a lot their sets to watch DVD’s, or do they use it as a monitor for their PC on occasions?
  ◦ Do they often change signal source?

A very important aspect of the analysis should be the evolution over time of the behaviours. Perhaps the majority of users discover after a short while that they do not like ambilight at all, or maybe their behaviour changes completely after a few months (increased use of the Content Browser, for example, due to its learning curve). A relationship could also appear with the different television models: we could discover that people who buy a big screen television use it less than others (either because they are more busy, or because it was thought simply as an expensive decoration for the home). After all, a television set with Ambilight hung on the wall makes for very psychedelic evenings, as good a reason as any to buy a top-notch set...

Note: The Multimedia Content Browser is a key feature of Philips televisions that lets the user watch pictures or play audio or video files stored on a USB device or a memory card. It is an interface that enables navigation through the folder tree of the removable media drive, and visualisation of single pictures, but also of whole photo albums with musical background or playback of entire music directories as in a playlist.

In the end, a huge leap has been made from the initial set of requirements, and it is one that finally permits a complete definition of the facts and dimensions that will be present in the DIMENSIONAL database.

4.3 The dimensional database

4.3.1 Designing the database

Based upon the answers that we need to provide answers to, it is now much easier to identify the dimensions and the facts that need to be included in the database’s structure.
Figure 4.1: The Multimedia Content Browser
For example, if one question is:

*How many times is a television switched on every day?*

This can be translated into:

<table>
<thead>
<tr>
<th>measure</th>
<th>: number of times the TV is switched on</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>: the absolute time, with a granularity of 24 hours</td>
</tr>
</tbody>
</table>

Let us consider another question:

*What is the evolution of the time spent watching television every day, over the first year of ownership?*

This leads to the following:

<table>
<thead>
<tr>
<th>measure</th>
<th>: time during which the television is on</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>: the relative time, which would start counting on the first day of ownership (or the timestamp of the first record of the first file that the user sends)</td>
</tr>
</tbody>
</table>

By continuing this process of translating some simple questions that were defined by the CEC department into elements of the database, what we end up with is the complete schema of the multidimensional star database. Tables 4.1 and 4.2 summarise the result of this design step, while figure 4.2 shows the schema of the implementation in a relational database management system (which could be any one from MySql, PostGreSql, SqlServer, Oracle, or even IBM DB2).

<table>
<thead>
<tr>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>absTimeDimension</td>
</tr>
<tr>
<td>absolute time, with fields for year, month, day of month, day of week and hour of day</td>
</tr>
<tr>
<td>Dimensions (continued)</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relTimeDimension</td>
</tr>
<tr>
<td>userDimension</td>
</tr>
<tr>
<td>softwareVersionDimension</td>
</tr>
<tr>
<td>tvSetDimension</td>
</tr>
<tr>
<td>buttonDimension</td>
</tr>
<tr>
<td>sourceDimension</td>
</tr>
<tr>
<td>switchOnDimension</td>
</tr>
<tr>
<td>switchOffDimension</td>
</tr>
<tr>
<td>channelChangeDimension</td>
</tr>
<tr>
<td>ambilightDimension</td>
</tr>
</tbody>
</table>
### Dimensions (continued)

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureDimension</td>
<td>all of the television’s features are stored in this dimension. This includes teletext, picture in picture and content browser, among (many) others</td>
</tr>
<tr>
<td>contentBrowserDimension</td>
<td>this dimension is needed in order to track the type of content that the user opens from within the Multimedia Content Browser (which has been presented in section 4.2), as well as the file extensions and folder tree depth of the removable drive’s filesystem</td>
</tr>
</tbody>
</table>

Table 4.1: Listing of the dimensions of the dimensional database

### Measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeActive</td>
<td>we store here the number of minutes that the set was active in one given state (fixed by the application state, the input signal source and the ambilight mode)</td>
</tr>
<tr>
<td>featureNbOfTimesActivated</td>
<td>for the current state, how many times the current feature was activated</td>
</tr>
<tr>
<td>nbOfSwitchOns</td>
<td>for a specific SWITCHONDIMENSION value. The state of the set does not matter here</td>
</tr>
<tr>
<td>nbOfSwitchOffs</td>
<td>for a specific SWITCHOFFDIMENSION value. The state could be useful. As was mentioned earlier, this measure should be suspended until its implementation is possible</td>
</tr>
<tr>
<td>teletextNbOfPagesWatched</td>
<td>obviously, this measure is only used when the teletext feature is active</td>
</tr>
</tbody>
</table>
### Measures (continued)

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>teletextNbOfSubPagesWatched</td>
<td>teletext subpages are the multiple pages that slowly cycle within a standard page (mostly used for airline departures/arrivals and for the tv programme - with morning, midday, and evening subpages)</td>
</tr>
<tr>
<td>contentBrowserNbOfItemsOpened</td>
<td>number of items consulted for a specific value of the CONTENTBROWSERDIMENSION. Obviously, the active feature should be the content browser</td>
</tr>
<tr>
<td>ambilightNbOfTimesActivated</td>
<td>for the current state, how many times the current ambilight setting was activated. Note that OFF is also a possible setting</td>
</tr>
<tr>
<td>nbOfChannelChanges</td>
<td>for the current state and for a specific value of the CHANNELCHANGEDIMENSION, the number of times another channel has been selected. Note that zapping is filtered here: Although one channel change could imply zapping through 12 different channels, this does not appear in this measure but rather in the next one. A channel change is not counted as such and increments the zapping counter when the user changes channel again after less than five seconds</td>
</tr>
<tr>
<td>nbOfZappedThroughChannels</td>
<td>for the current state, the number of channels the user stayed on for less than five seconds</td>
</tr>
<tr>
<td>nbOfButtonPresses</td>
<td>for the current state, and for a specific value of the BUTTONDIMENSION, the number of button presses</td>
</tr>
<tr>
<td>sourceNbOfTimesActivated</td>
<td>for the current state, how many times the current input signal source was activated</td>
</tr>
</tbody>
</table>

Table 4.2: Listing of the measures of the dimensional database

One thing to point out is that in the end the granularity has been chosen to be 1h in the absolute time dimension, so that behaviours relative to the time
of day can be analysed, whilst the relative time dimension has a granularity of 24h, because more details would be useless for that data (nobody really expects to be blown away by the user’s choices between the 354th and the 355th hour of watching).

All measures are either iteration counters (like nbOfButtonPresses) or time counters, in minutes (like featureTimeActive). Each row of the fact table makes reference to the different dimensions, but in a way which differs between them. Indeed, the dimensions that have been defined earlier can be separated into three categories, as shown on figure 4.2.

- One one side are all the contextual dimensions, with information about the user, his set, the television software he is using, and the two time dimensions. These represent the environment variables that should always be present in any analysis.

- The second category is the one with all status dimensions: SOURCEDIMENSION, FEATUREDIMENSION and AMBIGLIGHTDIMENSION. All three always have a specific value: the ambilight feature is either on (in a certain mode) or off, there no in-between status. The same can be said for signal source (one source is always active and shown onscreen) and for the application state (there is always at least one active feature).

- The same cannot be said for the last category of dimensions, which includes all remaining dimensions except the SWITCHONDIMENSION (more on that later). The BUTTONDIMENSION is a prime example for this: its only purpose is for button counting (the counter is the nbOfButtonPresses measure), and nothing else.

As such, the SWITCHONDIMENSION fact should be read as follows:
for a certain user
and a certain television model
and with a certain softwareVersion
during a certain hour (in absolute time)
during a certain day (in relative time)
while a certain source was active
and a certain feature was active
and ambilight was in a certain state

and for a certain button
we have a certain number of button presses

: USER_DIMENSION
: TV_SET_DIMENSION
: SOFTWARE_VERSION_DIMENSION
: ABS_TIME_DIMENSION
: REL_TIME_DIMENSION
: SOURCE_DIMENSION
: FEATURE_DIMENSION
: AMBILIGHT_DIMENSION
: BUTTON_DIMENSION
: nbOfButtonPresses

In this case, we can see that the rest of the dimensions are in fact useless if we only want to count button presses.

However, to take another example, in order to read the information contained in the nbOfChannelChanges measure we should consider:

for a certain user
and a certain television model
and with a certain softwareVersion
during a certain hour (in absolute time)
during a certain day (in relative time)
while a certain source was active
and a certain feature was active
and ambilight was in a certain state

and for a certain way of changing channels
we have a certain number of channel changes

: USER_DIMENSION
: TV_SET_DIMENSION
: SOFTWARE_VERSION_DIMENSION
: ABS_TIME_DIMENSION
: REL_TIME_DIMENSION
: SOURCE_DIMENSION
: FEATURE_DIMENSION
: AMBILIGHT_DIMENSION
: CHANNEL_CHANGE_DIMENSION
: nbOfChannelChanges

Of course, the only signal source for which this measure is of any use is the standard TV input source, because changing channel when watching a DVD has no meaning: a press of the NEXT CHANNEL button would probably even change input source.

This is besides the point, however, as we have shown that the buttonDimension and the channelChangeDimension behave in the same way. They
represent the last tier; the useful measure that we can look at after both the context and the television status have been defined.

We have mentioned earlier the `switchOnDimension` which is quite particular, because it holds data which bypasses the `status` category of dimensions. Indeed, when a set is switched on its state is simply `OFF`.

### 4.3.2 Filling the database

Once the tables and fields of the database are all well-known, the following step should be mentioned: how will the parsing software proceed to fill the database row by row? Indeed, to understand how the values appear in the database is a key to the correct implementation of this environment. Not the only one, but not the least important one either. To illustrate this part, let us simplify the full schema, and take into account only three dimensions: the `absTimeDimension` (unchanged), the `featureDimension` (simplified to contain only the field `featureName`) and the `buttonDimension` (simplified to contain only the field `buttonName`). The facts that we will keep for this example are the following:

- `timeActive`
- `featureNbOfTimesActivated`
- `nbOfButtonPresses`

Thus, the present example (which, by the way, contains dimensions taken from each one of the three categories of dimensions mentioned on page 54) can help the analyst answer any of the following questions through specific querying of the database:

- *how much time has the television been active one specific day, or during a certain time frame?*
Figure 4.2: Schema of the DIMENSIONAL database
• How much time has a certain feature been active between 19h and 22h on Christmas eve (and how many times has it been activated)?

• When a certain feature is active, is button A more used than button B?

• More generally, what are the most used buttons?

But what should the data contained in the database look like, how should it be filled in? Let us suppose that a feature called featureA has been activated twice, on July 21, 2007, between 11h00 and 12h00, for a total of 27 minutes. While this feature was active the button called button 1 was pressed nine times, and button2 eight times. If all tables were empty to start with, they would look like the snapshot on figure 4.3 after the values were entered. This example can be seen as a template for all other dimensions and measures to be filled in.

4.4 The lossless database

As was mentioned in section 3.5.3.2 on page 39, the second database used for analysis should "have a structure copied on the one that was imagined"
for the project as a whole”. This means that, before being able to design the database, our understanding of the project must be flawless. Let us start by describing the way the logging is conceived globally.

- The highest-level entity in this project is the television user. When a user registers into the project he/she is asked some information that might be useful for user profiling.

- A user possesses a certain number of television sets that can all be used for logging purposes. One set, however, should not be shared by multiple users (or by multiple families), because otherwise user profiling becomes impossible. Each set is identified by a combination of its model ID and its serial number (a same serial can be used for two different products).

- The television sets produce a series of log files, whose naming is automated and explained on figure 4.4. The knowledge of the exact format can be used for extracting the user ID, the serial number of the set along with its model, and the log number.

- Each log file, after decryption, can be read as a certain number of records. To keep track of the order of the records within a given log file, they can be numbered. Each record contains a timestamp, but also an eventID and optionally some parameters, that can both be translated with the help of a configuration file. Thus, the records of a log file should be stored with an identifier of the configuration file that contains their definition. Indeed, in the worst case scenario a log file can be recorded by different software versions, should the software of the set be upgraded. This is the reason why the software version needs to be stored at record level, rather than at file level. It should also be noted that the parameters are not always integers. There can be some cases (for example when the eventID equals 0x3, which means that the parameters that follow represent the software version string) when the parameters should be read as characters representing a string in which case, of course, they must not be translated.

- Finally, there is the configuration file whose structure has already been explained thoroughly in section 3.1.2 on page 26, with its Release-
In the end, by translating the relationships between all these elements of the environment, we are able to define the complete *lossless* database. A very basic schema is provided on figure 4.5, and each part is detailed in the pages that follow.

### 4.5 The ETL process - Introduction

The Extraction, Transformation and Loading of the data from the OLTP source to the OLAP repositories is a key aspect of datawarehousing. The Extraction part is simply selecting the right information to be transferred (in our case, the correct log files, the ones that have not yet been integrated into the results). The Transformation and Loading are carried out by a C++ program that parses the files, converts them to a usable format, translates the data, filters and manipulates it, and then pushes it into the two long-term databases.
Figure 4.5: LOSSLESS database - basic schema
Figure 4.6: LOSSLESS database - user profile part
Figure 4.7: LOSSLESS database - configuration part
4.6 The ETL process - Extraction (multi-language interaction programming)

The Extraction part is to be carried out by a looping script that connects to a PHP file hosted on the remote server, thus querying the remote database for files to download. Files are then downloaded one by one, then have their MD5 checksums compared with the original files, and finally updates both the remote database (the isDownloaded flag is set to 1) and the local tracking database (with new entries for the new files).

4.7 The ETL process - Transformation

Once the file has been correctly downloaded to the local file system, it is ready to be parsed. This operation, however, is much more complicated than anything we’ve done so far. Also, as C is the most used language in the
software department at Philips, and because someone from the company will inherit all the code once it is done, standard C++ was chosen to implement this part. Note that the standard C++ library would not always be sufficient, particularly for communicating with the databases. The implementation discussed here is not directed towards any particular libraries, and although the open-source QT libraries were used for the coding, their use has been entirely restricted to its main scope: provide a link between the software and the databases. Let us start simple, by explaining the different steps of the file processing.

Firstly, the software connects to the tracking database (for the definition of all databases, see section 3.5 on page 33) in order to identify the files that have their isProcessed flag to zero. The batch of file names are then given as an argument to the call of the parsing software. As has already been exposed in section 3.1.1 on page 24, every log file needs to be decrypted before it can be read. The "filename.bin" file is read, and its decrypted version is saved as "filename.dec". Once it is decrypted, it can be parsed to a format that enables easy navigation through the data. The same needs to be done for all needed configuration files, since they are not in proper XML and therefore standard XML-parsing algorithms cannot be used. After all files have been parsed into more practical forms, they can be processed, with the configuration files providing the translation of the events and parameters. The aim of this whole sequence is to have the data stored in a format that is close to the way it will be stored in the database: as rows. Finally, all these rows can be inserted one by one into both analysis databases, and the tracking database can be updated by setting the isProcessed flag for the current file to 1.

The entire schema of the application is illustrated on figure 4.9, and each part of the program will now be explained more in detail. The colour code is the following:

<table>
<thead>
<tr>
<th>Colour</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>configuration file parsing</td>
</tr>
<tr>
<td>red</td>
<td>log file parsing</td>
</tr>
<tr>
<td>orange</td>
<td>post-processing of the data</td>
</tr>
<tr>
<td>green</td>
<td>databaseOutput</td>
</tr>
</tbody>
</table>
Figure 4.9: Schema of the Transformation and Loading application
4.7.1 Which log files to process?

The tracking database holds a record for each log file that has been downloaded from the remote server, with all downloaded-but-not-yet-processed files marked with a flag (isProcessed = zero). Thus, the parsing software must start by querying the database for the names of the files that need to be processed. This is done by the DATABASEREADER class, a read-only interface to the database. Its essential public methods are setConnectionParameters(), whose name rather explicitly explains its role: to provide the class with host name, user name, password and database name for the connection, and getFileInfo() which returns the file names of the files that will be processed, along with the attributes of the users that have provided these files and the dateOfFirstUse attribute that has been defined in all databases.

4.7.2 Parsing of the log files

A log file is very complicated to parse, because there are no lines as such (by contrast with a configuration file), only consecutive bytes that must be read according to a certain template. The LOGCLASS contains all methods for handling the encrypted log file, its decrypted version, and their storage format (the RECORDMAP and FILTEREDRECORDMAP). Its constructor receives as an input a file name, either a "*.bin" (in which case the file will automatically be decrypted) or directly a "*.dec", and its main method is, like for the configuration file, parse().

The parse() method fills in the RECORDMAP, which contains all records that appear in the log file (stored in lists called RECORDLIST) and indexed by software version. Whenever a new software version is detected, a new entry is created in the map and the following records will all be inserted into that new RECORDLIST. This makes it easy to separate records that must be interpreted with different configuration files. The intermediate steps are explained in what follows:

A record is read using a union, which in C++ is "a structure that assigns multiple variables to the same memory location".[13] In this case, we use
Figure 4.10: Parsing of the log file
a union that contains a character array and a structure that mimics the format of a complete record (the RECORDSTRUCT). The first ten bytes of the record are all read once to count the number of parameter bytes. The record is then read a second time, this time with the parameter bytes, and entered into the character array part of the union. Because the character array and the RECORDSTRUCT share the same memory location, this also fills in the attributes of the structure and voila! the record is stored. The RECORDSTRUCT is then copied into a RECORDCLASS with more functionalities, which in turn is added to the RECORDLIST that corresponds to the current software version (in the RECORDMAP).

4.7.2.1 Correcting the timestamps

Unfortunately it does not end here, because sometimes the recorded timestamps are not correct. Indeed, television sets have no internal battery, and thus cannot keep track of time when they are unplugged. This means that, whenever a set is unplugged, it starts incrementing its timestamp all over from zero, which is 01/01/1996, midnight. In order to build an algorithm capable of correcting at least some temporal incoherences, an understanding of the sequence that leads to time synchronisation is necessary. While it is on, the television software scans the teletext signal that it receives for occurrences of date and time. Whenever it searches, an event is generated (the SystemDateTime event). As soon as a correct time is detected in the signal, the hour of day and then the date are successively updated. This leads to consecutive records that read like figure 4.11.

Thus, by detecting such patterns, the timestamps can be corrected to a certain extent. In fact, the only case that can not be corrected is when the television user pulls the plug once more before the set can find the correct date and time. Figure 4.12 shows how the corrections can be made. Technically, the corrections that need to be applied to the elements of the RECORDMAP are first calculated and stored in the TIMESTAMPCORRECTIONLIST. Every single element of this list is a structure which contains two iterators that point to specific records (the boundaries of a series of records that have wrong timestamps), along with the value of the correction to apply (the
difference between the correct timestamp and the original wrong timestamp. By iterating on the elements of this list, the RECORDMAP ends up being corrected at least partially. Records that have not been corrected are later singled out thanks to their timestamps.

4.7.2.2 Filtering the recordMap

Now that we have a complete and mostly corrected RECORDMAP, the hardest part of parsing the log file is done. However, in order to ease further analysis of the data, an extra attribute can be used: the FILTEREDRECORDMAP, which is basically a copy of the RECORDMAP from which all records whose timestamp is lower than that of 01/01/2000, midnight, have been removed. The year 2000 is a threshold like any other that has been chosen arbitrarily to separate correct from wrong timestamps. This is just a way of filtering false and impossible-to-correct information that could potentially ruin all efforts put into the analysis.

In the end, the information contained in the log file can be accessed by using the public access methods:

- `getRecordMap()`
- `getFilteredRecordMap()`
Figure 4.12: Correction of the records’ timestamps
In order to do this once for all files whose names have been passed as argument to the software, all (pointers to) instances of LOGCLASS are contained in a LOGFILEMAP, referenced by LOGATTRIBUTESCLASS (a class that parses a fine name to determine the userId, the set’s serial number, its model, and the log number).

### 4.7.3 Parsing of the configuration file(s)

Now that all log files have been parsed, and the required configuration files are known, the latter can in turn be parsed.

A configuration file is handled by the CONFIGCLASS, whose constructor needs
only the file name. The initialise() method is then called to define, through the mapPossibleMarkers attribute, all flags that are used in the configuration file. Once this is done, the public parse() method can be called. This method, with the help of the DataInput class which represents an interface to a text file (with its methods that allow for navigating the file, like get-NextLine()), reads the file line by line, and parses each line by calling the appropriate method based on the active flag. For example, if the current line defines an event the parseEventLine() method will be called, and after a certain amount of cropping the information is entered into an instance of the EventClass, which in turn is added to the EventMap which contains all event descriptions, and indexes them by their eventId. In the same way, after parsing the file, the enumGroupsSet contains all enumGroups defined in the configuration file, the enumParametersMap, a three-dimensional map, contains all parameter descriptions indexed by enumGroup, by parameter position, and by parameter value (see section 3.1.3 on page 27 for information on the way this information is used for interpreting the log file), and the softwareVersion is also stored.

In the end, all needed configuration files are stored in the configurationFileMap, which contains (pointers to) instances of ConfigClass referenced by the software version that they represent. All information contained in the configuration file can be accessed by using the public access methods:

- getSoftwareVersion()
- getEnumGroupsSet()
- getEnumParametersMap()
- getEventMap()

4.7.4 Post-processing

Once all files have been parsed, they can be combined to form the aforementioned ”rows” that just need to be imported into the databases. In truth,
Figure 4.14: Post-processing of the log file
the lossless database could already be filled in now, without any post-
processing layer, although this will be mentioned in the next section. The
post-processing is needed to make sure that the correct information is entered
in the dimensional database. Therefore, a certain amount of information
filtering and aggregation is needed before the databaseWriter class can
be called upon. This is made possible by the postProcessingClass, which
takes as input one instance of logClass at a time, and the configurationFileMap. The public executePostProcessing() method should
then be called. It first invokes the initialise() private method, to deter-
mine what events must be looked for in the log file based on the multiple
eventMap that have been generated earlier and on the analysis that is
wanted. For example, with the current requirements that have been dis-
cussed multiple times in Chapter 2, the initialise() method will output an
eventIDMap which will contains all useful eventIDs (for example 0x30000
for a keypress), and a string that describes them (in this case, "button").
After this has been done, the following operations are successively carried
out:

- a call to setProcessedRecordList(), that generates a list of what
  are called processedRecordClass instances. The records of the
  log file are iterated through, and only the ones that are useful to the
  analysis (i.e. only those that are related to the information we need
  in the database) are processed. This processing translates the infor-
  mation contained in a given record into a format that is closer to the
database’s. In order to understand a bit better what this means, let
us we consider the example that was used in section 4.3.2 on page 56,
where the dimensional database was simplified to only provide three
dimensions (absolute time, features, buttons). In this case, the pro-
cessedRecordList would have only the following attributes:

  o (a pointer to) an instance of dateClass, which would provide
    the processing of the timestamp to a more readable format (with
    year, month, day of month, day of week, all the way down to hour
    of day, minute, and second).

  o (a pointer to) an instance of featureClass, which would only
    contain the feature name and a flag to know if the feature is being
    entered or exited.
○ (a pointer to) an instance of BUTTONCLASS, which would only contain the button name.

○ a character string flag (RECORDTYPE) indicating if the current processed record corresponds to a keypress (if RECORDTYPE=="button"), the activation/deactivation of a feature (if RECORDTYPE=="feature") or a shutdown of the set software (if RECORDTYPE=="off"). The value of the flag gives an indication of what information must be read from the current record: BUTTONCLASS and FEATURECLASS do not both carry information at the same time (a record either indicates a keypress or the (de)activation of a feature, but never both at the same time). Of course, this is only a simplified example. The final software includes these attributes and more: one for each dimension in the database. The end of the log is indicated by an extra processed record in the list, whose RECORDTYPE flag is set to "endOfLog".

- a call to COMPLETELIST(), a public method of the PROCESSEDRECORDLIST that, as its name indicates, completes the information it contains by inserting records at the hour, every hour. This is done to simplify further analysis.

- finally, a call to the most important method: COMPLETERECORDLIST-TOROWLIST(), that generates a list of items called ROWS that can afterwards be exported to the dimensional database. These ROWS, instances of ROWCLASS, must be seen as rows of the fact table, and as such contain the following attributes:

○ (pointers to) instances of ABSTIMEDIMENSIONCLASS, FEATUREDIMENSIONCLASS, BUTTONDIMENSIONCLASS, and all other classes that provide the same information than the dimensions of the database.

○ all measures, that represent the columns of the fact table in the database: NBOFBUTTONPRESSES, TIMEACTIVE, ...

After all this is done, the information can at last be entered in the databases. The following section explains how this is carried out.
Now that the information contained in the log files and the configuration files has been completely formatted, the final step is to import all this data into both databases. This part is carried out by the DIMENSIONALDATABASEWRITER and LOSSLESSDATABASEWRITER classes which, not unlike the aforementioned DATABASEREADER class, provide communication with one database at a time through the setConnectionParameters() public method. Their similarities end here, however.

Indeed, the LOSSLESS database’s structure so closely mimics that of the data itself that no post-processing is needed to aggregate, filter or interpret any
of the data. Therefore, the class devoted to writing to this database only needs to, first, fill in all information contained in the different configuration files that were used so far. After this is done it can add fields, if necessary, to the tables where users, television sets, models and log file names are stored. Finally, all records can be inserted into the database.

The dimensional database, on the other hand, only needs the result of the post-processing. It first fills in all data that relates to the dimensions, before being able to write facts.
Chapter 5

Witnessing the results

In order to demonstrate the capabilities of this whole framework, two test databases have been created (DIMENSIONAL and LOSSLESS), and a custom-built dimensional database analysis software was used to output some graphs after one single four-hour-long log file was processed. Note that the log file that was used for these tests seems to have been generated for the sole purpose of testing the logging television software, which would explain why we end up with so many button presses in so few time and other bizarre combinations.

5.1 Using the lossless database

We have made it clear that the lossless database is useful for questions that cannot be answered by querying the dimensional one. Indeed, a lot of information is lost in the process of transforming the data contained in the log files into dimensional database rows. In particular, the notion of "chain of events" that lead the television set from state A to state B has utterly disappeared. When such information is needed, however, the analyst can resort to querying the lossless database instead.

An example of a query that could be answered only with the lossless database is the following:
How often does it happen that the user enters a certain feature and then immediately exits?

This can be translated into counting the number of times that we see the following pattern in the loggings:

1. A feature is entered at time\( \text{Stamp} \ t \)
2. The feature is exited between time \( t + 1 \text{second} \) (an inferior threshold to rule out features that act as transitions between other ones) and time \( t+2\text{seconds} \) (the superior threshold).

This can in turn be translated to counting the number of lines returned by the following SQL query on the LOSSLESS database:

```
SELECT lr1.timeStamp AS startTime,
       lr2.timeStamp AS stopTime,
       eg1.enumGroup,
       e1.eventIDText
FROM logRecords AS lr1
     JOIN events AS e1 USING(events_id)
     JOIN enumGroups AS eg1 USING(enumGroups_id)
     JOIN records_enumParameters AS re1 USING(logRecords_id)
     JOIN enumParameters AS ep1 USING(enumParameters_id)
     JOIN logRecords AS lr2
     JOIN events AS e2 USING(events_id)
     JOIN enumGroups AS eg2 USING(enumGroups_id)
     JOIN records_enumParameters AS re2 USING(logRecords_id)
     JOIN enumParameters AS ep2 USING(enumParameters_id)
WHERE eg1.enumGroup = 'STATEGROUP'
     AND eg2.enumGroup = 'STATEGROUP'
     AND ep1.parameterDescription = 'LOADED'
     AND ep2.parameterDescription = 'UNLOADED'
     AND lr2.timeStamp BETWEEN lr1.timeStamp+1
                             AND lr1.timeStamp+2;
```
5.2 The dimensional database analysis software

To illustrate the way the dimensional database can be analysed, a software with a graphical user interface has been programmed with QT libraries. The objective was to have access to the results in a graphical environment until a full-blown database analysis software was set up (open-source solutions like Pentaho, BizGrez, OpenI, or commercial solutions like Microsoft SQL Server Analysis Services). This would also act as a verification step, to assess whether the database was correctly designed according the main aspect: permitting a dimensional analysis of the results.

This GUI enables the user to select a fact from the fact table, and a dimension from the various dimension tables, which are then plotted one against the other on a graph in a tabbed panel with the press of a button. The generated graphs can then be saved to a directory in PNG format, with the file names being copied on the graph titles.

This software is in fact an interface to the dimensional database, with the choice of fact and dimension modifying accordingly the query that is sent to the database. Data narrowing (limiting the results to certain values of the dimension) is not yet implemented in the software, but it could be done in two days’ work.
5.3 Results

Some outputs will be printed in this section, showing how the whole environment that has been designed in the current paper answers the initial requirements. Two facts (timeActive and nbOfButtonPresses) are plotted against relevant dimensions (the hourOfDay, the featureName, and the buttonName) to provide answers to questions such as:

1. At what time of day was the set more used?
2. While the set was on, what are the most used features?
3. Which buttons are most used?

Question 1 is answered on figure 5.3, where it can be seen that the set was used in the morning. In fact, examining the log file we see that the set was switched on at 8:30, switched off and then back on once or twice between 9 and 10, end then switched off again a little after 12:30. This is confirmed by the graph, which somehow also validates the transformation and loading of the data into the database. Note that the ”0” or ”NULL” bar on all graphs
corresponds to the summed total of all the other bars (which is why it is always the longest one).

Question 2 can be answered by looking at figure 5.4, which shows that App-StateOSD was the most used feature, but not by such a long shot considering that it is the standard feature for watching television ”normally”. Maybe the user was experimenting a little bit with the set while watching.

Figure 5.5 is the one that raised eyebrows and prompted recurring remarks about the accuracy of the results that were presented. We see that there were nearly 800 button presses between 9:00 and 10:00! This means one button press every 4.5 seconds. Analysis of the log file showed that these results are indeed correct, but allowed for another interesting interrogation: why would the ten digit keys (0, 1, 2, ...) be all pressed in sequence, in the correct order, in under five seconds, under normal use? This is the reason why this first analysed log file should be considered a simple test, with no meaningful information to be gathered from its analysis.

Finally, question 3 is answered on figure 5.6, which shows the number of button presses, during the test, for each button that was used at least once.

In the end, this graphical analysis of some results that can be obtained very easily by the analyst shows that the database was well designed, and that both its structure and the Extraction, Transformation and Loading process are correct with respect to the requirements that were formulated at the start of the project. We can now look forward to finally including real-life log files gathered from real users, and finding out about their behaviour.

### 5.4 A word on data mining

Data mining can be described as the *extraction of useful information from large data sets or databases.*[14] This term is used in the title of this paper, meaning that from the start Philips had thought about implementing data mining on the gathered loggings. This, however, shows in what way the initial requirements needed to be redefined before any of this could be done. Indeed, as mentioned in section 3.2, the project initially was to have text-file output, through C++-analysis of the log files. Of course, implementing
data mining on text files designed for Microsoft Excel would have been a daunting task. With the current status of the environment, however, with information stored amongst others in a dimensional database, the addition of a data mining layer becomes much easier.

The main application of data mining to our project would be user clustering: the possibility to group users in different profiles according to their behaviour. This would be of great value for the company for future product designs. Another application would be associative mining: like the famous beer-diaper example\(^1\), some characteristics of user behaviour could be related, and this could be shown by implementing data mining algorithms on top of our results sets.

In the end, while the data mining layer has not been implemented yet, the whole life-cycle of the log files and their possible use have been determined:

- Redefining the requirements
- Analysing the best format to output the results
- Designing the environment, from the user’s upload of the file to the dimensional analysis of the results set

Everything is in place to enable the installation of an additional analysis layer, which would not have been possible (or at least not as straightforward) had another storage format been chosen for the data, all had this whole environment not been designed.
Figure 5.3: Time active in function of the hour of day
Figure 5.4: Time active in function of the feature name, for the duration of the logging
Figure 5.5: Number of button presses in function of the hour of day
Figure 5.6: Number of button presses in function of the button name.
The beer-diaper example is a very common one. A study in the United States has shown that when men bought diapers in a grocery chain's shops on Thursdays and Saturdays, they also ended up buying beer. This is called associative mining, and the usefulness of such a discovery is immediately obvious: diapers got moved near the beer (or the opposite) to spur this tendency, and both products were always to be sold at full price on Thursdays to increase the income.
Chapter 6

Conclusion

This project has led us through a typical problem in today’s world: how to create information on television usage based on the simple logging data. After having examined the importance of event logging in various fields, the initial requirements were analysed in-depth. The aim of the project was to design and build a software solution that allows for proper analysis of the data that is sent by the television user. It soon appeared, however, that the requirement for flexibility was going to be a tricky one.

The first step was the storage of the information, which led us to the use of a certain number of databases. Of the three components of the ETL process, the transformation proved to be particularly difficult to implement, because of the post-processing that was needed before pushing data into the appropriate database. Another complex aspect to this was to make sure that the different parts of the environment worked well together, even though different programming languages were used.

Finally, with all components working together towards a common goal, a validation of the results was carried out with a custom-built program that enabled dimensional database analysis.

In the end, it is certified that the whole concept works as it should, with true results and important information soon to be discovered by Philips analysts as soon as the environment is completely coded and ported to a Philips
server. The results are very positive, and definitely provide a flexible solution to the problem that was originally described. Keywords for this project include multi-language programming, data warehousing, Extraction, Transformation and Loading, but most of all flexible analysis and understanding requirements. Also, let us not forget that perspectives for this whole post-processing environment should be evaluated on the long term, when a consistent user base for solid and statistically relevant input will be used. An eventual data mining layer could then also be added in order to dig deeper into the results sets.

From a personal point of view, it must be said that the experience one gathers from working on such a project, in a field as important as business intelligence and dimensional reporting, will probably prove invaluable for future projects. Interestingly, I had to learn software programming at this level from scratch, which in the end may constitute an advantage from a learning point of view. This whole project is a personal as well as a professional breakthrough.
Bibliography


93

