

Data Warehousing

The Layer between Raw Data Sources and Knowledge

Sivanujann Selliah

s093042

Kongens Lyngby
August 2014

Technical University of Denmark
DTU Compute
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, Building 324
DK-2800 Kgs. Lyngby, Denmark
Phone +45 45 25 30 31
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

Data warehousing is an essential part of delivering Business Intelligence (BI). It is the foundation which makes it possible to analyse and get answers for many different questions about large data sets.

The Technical University of Denmark (DTU) is for the second time participating in a competition to build an energy efficient house. The competition is called Solar Decathlon Europe. For the 2014 edition of this competition the entry from DTU is called EMBRACE.

This thesis will investigate how a data warehouse and BI solution can be made for the EMBRACE house. The principal research question of this thesis is how to give occupants and other subsystems of an EMBRACE house the insights of the functioning of the house, by measurements done by the control systems. The proposed solution is to make a data warehouse which is exposed through an OLAP cube, which is based on a star schema dimensional model. For reporting insights a BI application is also proposed.

The data warehouse was built using an ETL system which integrated four different operational source systems. These systems were the Hardware System Integration, Data Collection, Weather Data, and SDE Measurements. All the data from these systems were put into a star schema dimensional model, with multiple fact tables and dimension tables. The ETL system consisted of two types of ETL processing, one which was based on Entity Framework and another which was based on SSIS. The SSIS processing outperforms the EF-based ETL, but the EF-based ETL was more flexible.

The data warehouse was exposed through the OLAP system SSAS, which made query performance much better by pre-calculating aggregates and it provided hierarchical intelligence for the dimensions.

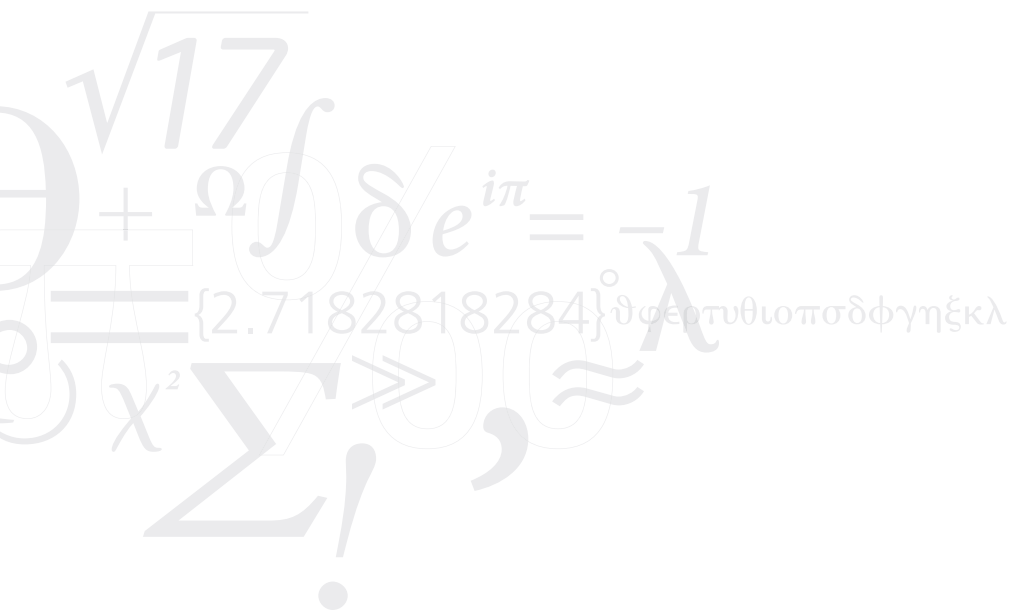
The final implementation was evaluated through a number of integration tests done through black-box testing. All the integration tests succeeded, which means that the components of the DW/BI solution and the integration between these works as intended. To test the functional requirements of the DW/BI solution a number of system tests was done through black-box testing. It was shown that all the functional requirements are met for the DW/BI solution implemented. Besides system testing, acceptance testing was also conducted, which included performance testing. These tests showed that the processing times set in the non-functional requirements are mostly met by the DW/BI solution, especially by the SSIS package.

Lastly to evaluate the DW/BI solution a number of BI reports were made, which showed that valuable knowledge can be gained from the insights provided by the DW/BI solution.

The thesis shows that because BI solutions are quite generic it is actually possible to build a data warehouse and BI solution for an intelligent house context.

Keywords: Data Warehouse, Business Intelligence, Dimensional Modelling, Cube, OLAP, Star Schema, Intelligent Housing, Home Control, Home Automation

This page intentionally left blank



Resumé

Data-warehousing er en nødvendig del af Business Intelligence (BI). Det er fundamentet som muliggør analyse og at få svar på mange forskellige spørgsmål omkring store datasæt.

Danmarks Tekniske Universitet (DTU) deltager for anden gang i en konkurrence for energi effektive huse. Konkurrencen hedder Solar Decathlon Europe. For 2014 udgaven af konkurrencen er DTUs bidrag et hus som hedder EMBRACE.

Dette speciale vil undersøge hvordan en data warehouse og BI løsning kan laves for EMBRACE huset. Specialets hovedproblemstilling er hvordan man kan give beboere og andre subsystemer af EMBRACE huset indsigt i husets funktion på baggrund af målinger lavet af kontrol systemer i huset.

Den foreslået løsning til dette er at lave et data warehouse som er udstillet igennem en OLAP kube, som er baseret på et stjerneschema, som er en dimensional data model. For at lave indsigtsrapportering er en BI applikation også foreslået.

Det færdige data warehouse blev bygget med et ETL system som integrerer fire forskellige operationelle kilde systemer. Disse systemer var Hardware System Integration, Data Collection, Vejr data og SDE målinger. Al data fra disse systemer blev sat ind i stjerneschemaet, med en del fakta- og dimensionstabeller. ETL systemet bestod af to forskellige typer af ETL databehandling, et som var baseret på Entity Framework og et som var baseret på SSIS. SSIS databehandlingen overgik ydeevnen på den EF-baseret ETL, men den EF-baseret ETL var mere fleksibel. OLAP systemet SSAS blev brugt til at udstille det færdige data warehouse, som gjorde at forespørgsels ydeevnen blev meget bedre ved at udregne akkumuleret tal på forhånd og det leverede også hierarkisk intelligens for dimensioner.

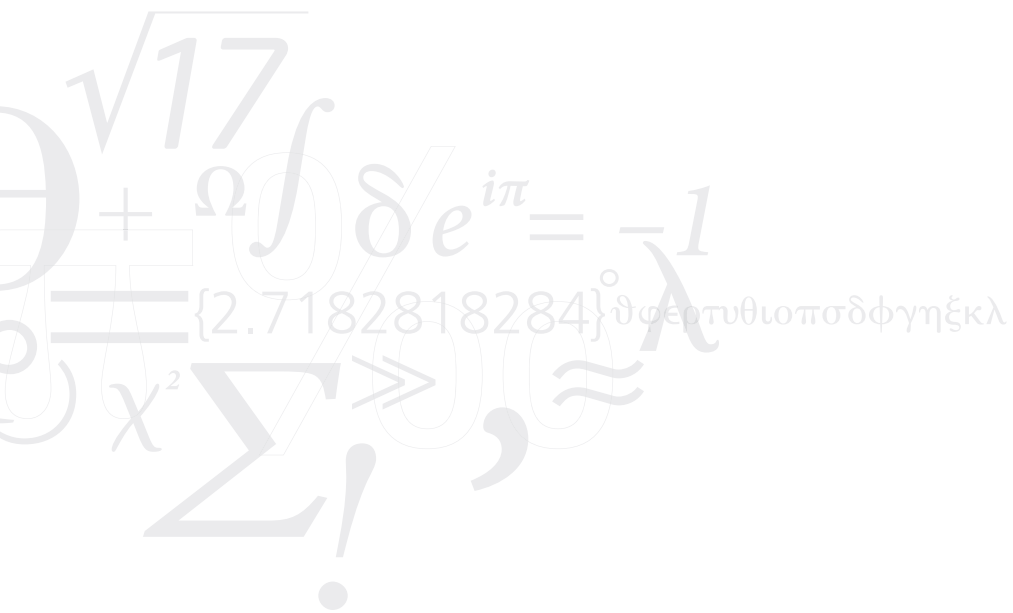
Implementeringen blev evalueret igennem et antal integration tests, som blev udført vha. black-box testing. Alle integration tests var succesfulde, hvilket betyder at komponenterne i DW/BI løsningen og integrationen mellem disse virker som forventet. For at teste de funktionelle krav af DW/BI løsningen blev et antal af system tests udført vha. black-box testing. Det viste sig at alle funktionelle krav var opfyldt for DW/BI løsningen. Ud over system tests blev der også udført acceptance tests, som inkluderede performance tests. Disse tests viste at behandlingstiden sat i de ikke-funktionelle krav til dels var opfyldt, især for SSIS pakken.

Til sidst blev der lavet nogle BI rapporter som viste at værdifuld viden kan tilegnes fra indsigter leveret af DW/BI løsningen.

Specialet viser hvordan generiske BI løsninger faktisk kan bruges til at bygge en data warehouse og BI løsning i en intelligent hus kontekst.

Stikord: Data Warehouse, Business Intelligence, Dimensional Modelling, Cube, OLAP, Star Schema, Intelligent Housing, Home Control, Home Automation

This page intentionally left blank



Preface

This master's thesis was prepared at DTU Compute, Department of Applied Mathematics and Computer Science at the Technical University of Denmark, DTU, in fulfilment of the requirements for acquiring a Master of Science in Engineering (Computer Science and Engineering).

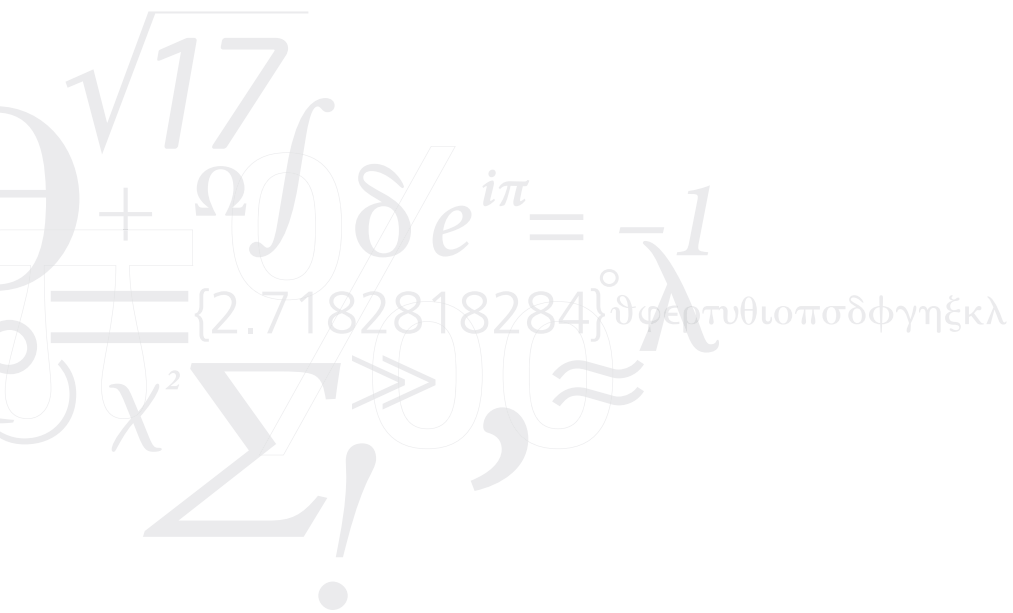
The thesis was done by Sivanujann Selliah (s093042). The thesis supervisor was Christian Damsgaard Jensen.

The data warehouse is available through <http://bi.epiccloud.solardecathlon.dk/>.

Kongens Lyngby, August 2014

Sivanujann Selliah

This page intentionally left blank



Contents

1	Introduction	1
2	Context	3
2.1	Intelligent Houses	3
2.2	Solar Decathlon Europe 2014	4
2.2.1	Team DTU	5
2.2.2	The ten contests	5
2.3	EMBRACE	6
2.3.1	Urban context	7
2.3.2	Competition	8
2.3.3	Save, smart and share	8
2.3.4	Appliances and control systems	8
2.3.5	Results from SDE14	9
2.3.6	After Versailles	9
2.4	EPIC Cloud	10
2.4.1	Logical components	11
2.4.2	Technology used	12
2.5	Business Intelligence	12
2.5.1	Motivation for using BI	14
2.6	Data Warehouse	14
2.6.1	Motivation for using DW	15
2.7	Scope	15
2.8	Summary	16
3	State of the Art	17
3.1	Third Normal Form	17
3.2	Dimensional Modelling	18
3.3	Facts	20
3.3.1	Structure	21
3.3.2	Keys	21
3.3.3	Additivity	22
3.3.4	Grain	23

3.3.5	Null values	24
3.3.6	Types of fact tables	24
3.3.7	Factless	25
3.3.8	Numeric values as dimensions	25
3.3.9	Units of measure	26
3.4	Dimensions	26
3.4.1	Structure	27
3.4.2	Flags, codes and indicators	28
3.4.3	Junk	28
3.4.4	Keys	28
3.4.5	Drilling down and other dimensional operations	29
3.4.6	Hierarchies	30
3.4.7	Multivalued	30
3.4.8	Date and time of day dimensions	31
3.4.9	Conformed dimensions	31
3.4.10	Drilling across	32
3.4.11	Slowly changing dimensions	32
3.5	Star Schema	33
3.6	OLAP Cubes	35
3.6.1	MDX query language	36
3.7	Design Process	39
3.8	Data Warehouse and Business Intelligence components	40
3.9	Summary	41
4	Goals	43
4.1	Stakeholders	43
4.2	Project Goals	44
4.2.1	Research Questions	44
4.2.2	Principal Research Question	45
4.3	Non-functional requirements	45
5	Analysis	47
5.1	Project goals from given context	47
5.2	Sub-problems	48
5.3	Solutions for sub-problems	50
5.3.1	Extending operational source systems	50
5.3.2	3NF relational model	52
5.3.3	Star schema model	54
5.3.4	Multidimensional system with an OLAP cube	56
5.4	The proposed solution	59

5.5	Business Intelligence Reporting	59
5.5.1	House measurement types	60
5.5.2	Weather measurement types	61
5.5.3	Competition measurement types	61
5.6	Operational Source Systems	61
5.6.1	Hardware System Integration	61
5.6.2	Data Collection	65
5.6.3	Weather Data	65
5.6.4	SDE Measurements	66
5.7	Requirements specification	66
5.7.1	Functional requirements	66
5.7.2	Non-functional requirements	68
6	Design	71
6.1	Architecture	71
6.2	Data Collection	74
6.2.1	Data model	74
6.2.2	Data Collection application layers	76
6.2.3	Helper service	78
6.3	Data Warehouse	81
6.3.1	Design process	81
6.3.2	Cube	87
6.4	ETL	89
6.4.1	Data processing	89
6.4.2	ETL application layers	91
6.5	BI Application	93
6.5.1	Data model	93
6.5.2	BI application layers	95
6.5.3	App Service	96
7	Implementation	99
7.1	Details on Data Collection	99
7.1.1	DataCollectionService	100
7.1.2	DataCollectionHelperService	102
7.2	Details on Data Warehouse	103
7.2.1	Data Warehouse database	103
7.2.2	SSAS cube	104
7.3	Details on ETL	112
7.3.1	ETL solution	112
7.3.2	SSIS ETL Package	117

7.4	Details on BI application	129
7.4.1	BI application	129
7.4.2	BI website	131
7.4.3	App service	132
8	Evaluation	141
8.1	Integration tests	141
8.1.1	Data Collection Service	141
8.1.2	Data Collection Helper Service	143
8.1.3	ETL	144
8.1.4	BI application	148
8.2	System tests	148
8.2.1	Gain knowledge about the functioning of the house	149
8.2.2	See how elements relate	149
8.2.3	See how elements of the house and outside conditions relate	149
8.2.4	Perceive the numbers as tangible elements	150
8.2.5	Drilling through data to analyse it	150
8.2.6	Slice and dice data cubes	150
8.2.7	Get relevant and valid data	151
8.3	Acceptance tests	151
8.3.1	Performance tests	151
8.3.2	User acceptance tests	155
8.4	BI reports	156
9	Conclusion	161
	Appendix	165
A	Third Normal Form	165
A.1	First normal form	165
A.2	Second normal form	165
A.3	Third normal form	166
B	Additional dimensional modelling techniques	169
B.1	Facts	169
B.1.1	Textual measures	169
B.1.2	Fact table types	169
B.1.3	Centipede	170
B.2	Dimension	170
B.2.1	Snowflake schema	171
B.2.2	Outriggers	171

Contents

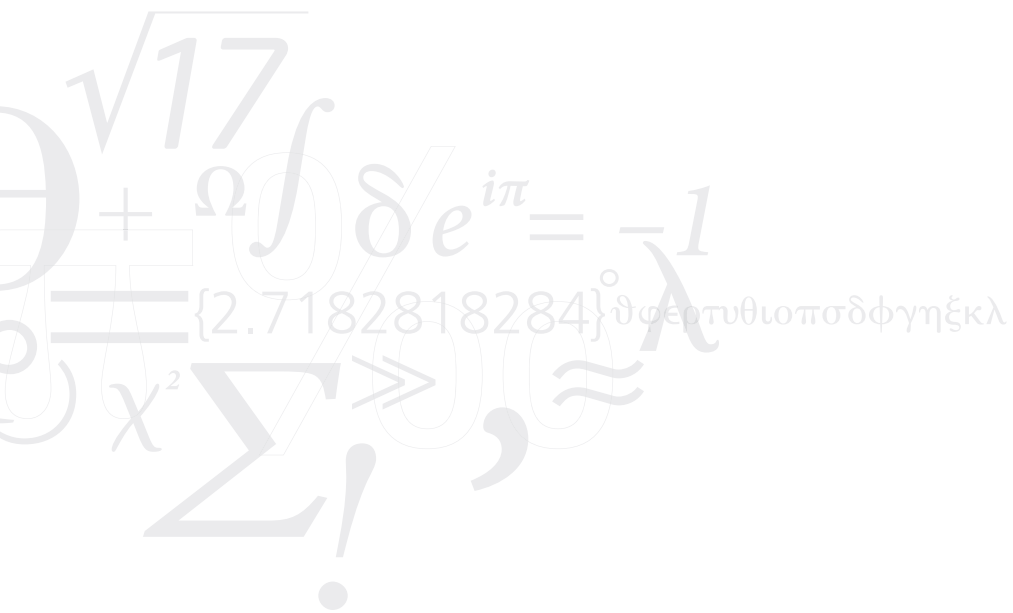
- B.2.3 Degenerate dimensions 171
- B.2.4 Role playing dimensions 172

- C SSAS Cube 173**
- C.1 Calculation script 173
- C.2 Dimension usage 174

- D ETL 177**
- D.1 Data Warehouse entities 177

- Bibliography 183**

This page intentionally left blank



List of Figures

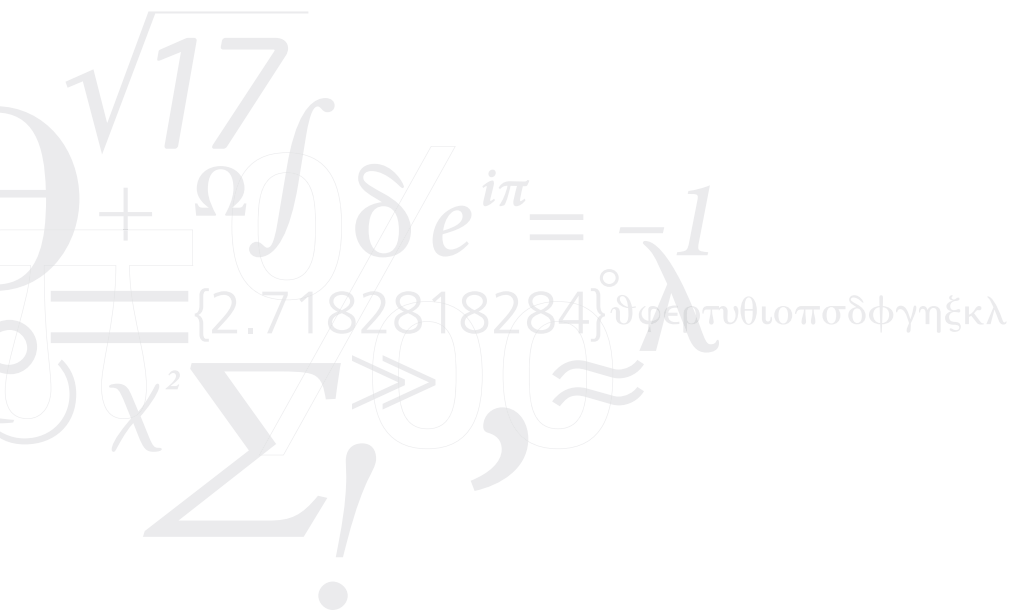
2.3.1 EMBRACE in Versailles	6
2.3.2 Ground floor and stairs.	7
2.3.3 Ground floor with kitchen and living room.	7
2.3.4 Ground floor with kitchen and living room.	7
2.3.5 First floor and door to private terrace.	7
2.3.6 Sheltered garden below the weather shield.	7
2.3.7 Sheltered garden with hobby room.	7
2.3.8 Final scores	10
2.4.1 Logical components of EPIC cloud	11
3.2.1 Example of cube with floor, device and time as dimensions	19
3.3.1 Example of a fact table	21
3.4.1 Example of a date dimension table	27
3.5.1 Example of a star-like structure for a Temperature fact table and five dimensions	34
4.2.1 Project goals	45
5.2.1 Use cases from the goals	48
5.6.1 Excerpt of HSI common interfaces and entities	63
6.1.1 Architecture of the DW/BI solution	72
6.2.1 Subcomponents of the Data Collection component	75
6.2.2 Data model for Data Collection	76
6.2.3 Data Collection packages	77
6.2.4 Sequence for Data Collection Helper service	79
6.2.5 Optional sequence for Data Collection Helper service	80
6.3.1 Subcomponents of the Data Warehouse component	81
6.3.2 Class diagrams for dimensions	83
6.3.3 Class diagram for DeviceGroupFact	84
6.3.4 Class diagram for records	85
6.3.5 Bool records	86
6.3.6 Numeric records	86
6.3.7 Numeric records with duration	87

6.3.8 Class diagram for WeatherFact	88
6.3.9 Classes for SDE measurement facts	88
6.4.1 Subcomponents of the ETL component	89
6.4.2 Processing flow for the ETL	90
6.4.3 Package diagram for ETL	91
6.5.1 Subcomponents of the BI component	93
6.5.2 Data model for BI application	97
6.5.3 Class diagram for BI application	98
7.1.1 Class diagram for Data Collection	100
7.2.1 Date dimension and hierarchies	110
7.2.2 Date dimension attribute relations	110
7.2.3 Device dimension and hierarchies	110
7.2.4 Device dimension attribute relations	111
7.2.5 Time dimension and hierarchies	111
7.2.6 Time dimension attribute relations	111
7.2.7 Weather dimension	111
7.3.1 Service related classes	114
7.3.2 Console class	114
7.3.3 Web related classes	118
7.3.4 ETL business logic classes	119
7.3.5 ETL common classes	120
7.3.6 Metadata classes	121
7.3.7 Destination classes	122
7.3.8 Sources classes	123
7.3.9 SSIS Package control flow	124
7.3.10 SSIS Package numeric data flow	125
7.3.11 SSIS Package bool data flow	126
7.4.1 Sequence of getting measures	134
7.4.2 Sequence of getting dimensions	135
7.4.3 Sequence of getting report	136
7.4.4 BI application classes	137
7.4.5 BI application data model	138
7.4.6 BI website classes	139
8.3.1 Processing time vs. number of rows - linear trendline	154
8.3.2 Processing time vs. number of rows - exponential trendline	154
8.4.1 Average temperature for the floor, the indoor and outside temperatures for Versailles during a full day	156
8.4.2 Average indoor, SDE and weather humidity in Versailles during week 27 for a full day	157

List of Figures

8.4.3 Average dimmable light duration in hours by percent level	157
8.4.4 Average dimmable light level by a full day and by floors	158
8.4.5 Energy consumption divided by the different groups of power meters	158
8.4.6 Total amount of kWh consumed and produced by weeks	159
8.4.7 Total amount of kWh consumed and produced by the hours of a full day	159
8.4.8 KPI value and status for energy production vs. energy consumption	160
C.2.1 SSAS cube dimension usage part 1	175
C.2.2 SSAS cube dimension usage part 2	175
C.2.3 SSAS cube dimension usage part 3	175
C.2.4 SSAS cube dimension usage part 3	176
D.1.1 Bool record entities	178
D.1.2 Numeric record entities	179
D.1.3 Numeric record with duration entities	180
D.1.4 Other fact entities	181
D.1.5 SDE fact entities	182

This page intentionally left blank



List of Tables

4.1.1 Stakeholders and their respective stakes	44
5.6.1 Boolean item types	64
5.6.2 Numeric item types	64
8.1.1 Integration tests for Data Collection Service	143
8.1.2 Integration tests for Data Collection Helper Service	144
8.1.3 Integration tests for ETL Data and Time	144
8.1.4 Integration tests for ETL HSI	145
8.1.5 Integration tests for ETL Weather	146
8.1.6 Integration tests for ETL SDE	146
8.1.7 Integration tests for ETL Data Collection	147
8.1.8 Integration tests for SSAS cube	147
8.1.9 Integration tests for BI application	148
8.2.1 System tests for gain knowledge about the functioning of the house	149
8.2.2 System tests for see how elements relate	149
8.2.3 System tests for see how elements of the house and outside conditions relate	150
8.2.4 System tests for perceive the numbers as tangible elements	150
8.2.5 System tests for drilling through data to analyse it	150
8.2.6 System tests for slice and dice data cubes	151
8.2.7 System tests for get relevant and valid data	151
8.3.1 Performance for ETL processing	153
8.3.2 Performance for BI application	155
A.1.1 Denormalised table	165
A.1.2 1NF normalised table	165
A.2.1 1NF normalised table for House entities	166
A.2.2 1NF normalised table for Device entities	166
A.2.3 2NF normalised table for House entities	166
A.2.4 2NF normalised table for Device entities	166
A.3.1 2NF normalised table for House entities	167
A.3.2 2NF normalised table for Device entities	167
A.3.3 3NF normalised table for House entities	167

A.3.4 3NF normalised table for Device entities 167

Acronyms

Acronym	Description
AD	Active Directory
BI	Business Intelligence
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
DAL	Data Access Layer
DW	Data Warehouse / Data Warehousing
DW/BI	Data Warehouse and Business Intelligence
EF	Entity Framework
ERP	Enterprise Resource Planning
ETL	Extract, Transform and Load
HSI	Hardware System Integration
HVAC	Heating, Ventilation, and Air Conditioning
IHC	Intelligent House Concept
IIS	Microsoft Internet Information Services
KPI	Key Performance Indicators
LINQ	Language-Integrated Query
MDX	Multidimensional Expression Language
NF	Normal Form
OLAP	Online Analytical Processing
PIR	Passive Infrared sensors
PLC	Programmable Logic Controller
POS	Point of Sales
PV	Photovoltaic Panels
RDBMS	Relational Database Management System
REST	Representational State Transfer
ROI	Return On Investment
SCD	Slowly Changing Dimension attributes
SDE	Solar Decathlon Europe
SQL	Structured Query Language

SSAS	Microsoft SQL Server Analysis Services
SSIS	Microsoft SQL Server Integration Services

Chapter 1

Introduction

This chapter introduces the thesis by giving an outline of data warehousing and Business Intelligence as well as the EMBRACE house and the competition in which it is competing.

Data warehousing is an essential part of delivering Business Intelligence (BI). It is the foundation which makes it possible to analyse and get answers for many different questions about large data sets.

BI is basically about getting knowledge out of large data sets and use it to help support decision making. This why it is being deployed more and more in many companies around world. By basing decisions on hard facts, it is possible for businesses to improve their competitive advantage and seek new business opportunities.

Nowadays BI solutions are quite generic, which makes it is possible to apply the same technology on many different areas. In this thesis BI and data warehousing will be used in an intelligent house context instead of a normal business context.

Intelligent houses can help occupants utilise a house more efficiently, this includes reducing the amount of energy consumed. These houses usually handle a lot of information, which is never shown to the occupants. But if awareness of ones energy consumption and overall usage of a house can be shown in a proper way, it might help utilise the house even more, and thereby e.g. reducing the energy consumption even more.

The Technical University of Denmark (DTU) is for the second time participating in a competition to build an energy efficient house. The competition is called Solar Decathlon Europe. For the 2014 edition of this competition the entry from DTU is called EMBRACE, which is a house split into two main parts. A thermal envelope, which is the dwelling and a sheltered garden.

The competition is between 20 houses from around the world. The houses are evaluated in ten separate contests, which are about different aspects of an energy efficient house.

For this house a cloud-based control system is made, which includes a data warehouse. The data warehouse will collect data and expose it through BI application, which can be used by the occupants to monitor and quickly react on the overall house functioning, e.g. to reduce the energy consumption.

This thesis will investigate how a data warehouse and BI solution can be made for the EMBRACE house. This requires an understanding of data warehousing and how data can be modelled to facilitate the BI ap-

plications to generate insights for occupants and subsystems of the cloud-based control system.

The principal research question of this thesis is how to give occupants and other subsystems of an EMBRACE house the insights of the functioning of the house, by measurements done by the control systems. The main goal of this thesis is inherited by the EMBRACE project, which is to save energy. This goal is decomposed into the goals of providing insights by analysing data through finding interesting relations, drilling through data, slicing and dicing data cubes and doing this in an intuitive and simple way for end-users.

The proposed solution is to make a data warehouse which is exposed through an OLAP cube, which is based on a star schema dimensional model. For reporting insights a BI application is also proposed. The data warehouse and BI solution will integrate several operational source systems, to be able to provide valuable insights about the house functioning.

The solution is implemented and is then evaluated by making integration tests, system tests and acceptance tests. Finally a number of example insight reports are made to see whether the solution can be used in an intelligent house context.

Because BI solutions are quite generic it is shown in this thesis that it is actually possible to make a data warehouse and BI solution for an intelligent house context, and use it for gaining insights into house functioning. This can help both occupants and subsystems to utilise an EMBRACE house better.

Chapter 2

Context

“One of the most important assets of any organization is its information”
—Ralph Kimball, Margy Ross, 2013

Contents

2.1	Intelligent Houses	3
2.2	Solar Decathlon Europe 2014	4
2.3	EMBRACE	6
2.4	EPIC Cloud	10
2.5	Business Intelligence	12
2.6	Data Warehouse	14
2.7	Scope	15
2.8	Summary	16

This chapter sets the context for the project by introducing intelligent houses, the Solar Decathlon competition, the EMBRACE house and the EPIC cloud.

2.1 Intelligent Houses

A house that knows when you are at home, and can open the shades and turn on the lights when a room is dark. A house which knows that when the windows are opened, it probably should not have full floor heating on. A house that can monitor the energy consumption and production, and give the occupants an instant view on how these are divided between the different installations and appliances. A house which can be controlled through a single and simple interface, e.g. where lighting and temperature settings can be set. These are just a few examples of houses which can be viewed as intelligent.

Most of these can be done using home automation, which is an installation where most electrical components and Heating, Ventilation, and Air Conditioning (HVAC) systems communicate with a central processing unit, which runs automation software.

Home automation is increasingly being deployed in newer houses as part of intelligent installations, because of the increased ease and comfort, as well as the energy saving aspects, which are provided by the

automatic functioning of the most energy consuming installations and appliances, like the HVAC systems and lighting.

Intelligent House Concept (IHC) and Programmable Logic Controller (PLC) hardware are examples of home automation hardware, these are also referred to as control systems.

Most home automation installations are hardware and software dependent, and the hardware can only be used by the proprietary software, this means that occupants will have to use several different devices to control their home, or use the same vendor for all installations. The automatic functioning is mostly done by checking for specific scenarios and setting set points on installations or separate systems. The monitoring of energy usage and the overall usage of the house is mostly used for the internal operation of the home automation software, and the information is rarely given to the occupants.

Home automation tries to make houses more intelligent and efficient, by automating house functioning of e.g. the HVAC systems, which are often just set once by the occupants and never adjusted, making these systems function inefficiently to the ever changing environment of a house. However it is still important for the occupants to be able to override the settings of the home automation so the indoor climate and general house functioning is to their preferences.

2.2 Solar Decathlon Europe 2014

An international competition is held every other year among universities to promote research and development of efficient houses. The competition is called Solar Decathlon Europe (SDE), which is a complementary competition to the U.S. Department of Energy Solar Decathlon. The two competitions are held in alternating years.

The U.S. Department of Energy Solar Decathlon was introduced in 2002, it tries to unite the best universities around the world to design, build and operate a solar-powered house which is full-scale and fully functional.

The Solar Decathlon Europe competition has taken place in 2010, and 2012. The 2014 edition of the Solar Decathlon Europe is held at Versailles, France, where 20 different proposals for efficient houses, from 16 countries, will be competing in ten separate contests, which is why it is a decathlon.

The objective of the competition is to design and build houses which are efficient. One aspect of this is to consume as few natural resources, e.g. energy, as possible. This can be done by producing enough energy by renewable sources, e.g. solar energy, to accommodate the total energy consumption of the houses. When houses can produce more energy than is needed by the houses themselves, the houses are called surplus-energy houses. Another aspect of efficiency is that the houses should produce a minimum of waste.

It is important that indoor climate and energy consumption is addressed as early in the design process of houses, so these can be fully integrated.

The teams assemble their houses for the competition in Versailles and they will be competing during July 2014. The competition site, a micro-city called Cité du Soleil, is open to the public, where they can get inspiration on how different teams around the world envision energy efficient houses.

The competition is supported by universities, institutions and companies, who provide technical know-how, products and financial subsidies. The proposals and projects are however mostly student-driven and are advised by faculty advisers.

2.2.1 Team DTU

The Technical University of Denmark (DTU) is participating in the Solar Decathlon Europe competition for the second time. The first house which competed in Madrid, Spain, in 2012 was called Fold. It ranked 10 out of the 18 competing houses. It received an award for the best photovoltaic panels (PV) integration. In the 2014 edition of the competition, the entry from DTU is called EMBRACE.

The main department responsible for the project is DTU Civil Engineering, while several other departments are also participating, including DTU Compute.

The project tries to set new standards in Denmark for low-energy houses. There is a huge potential for low-energy houses, because almost half of the energy used in Danish houses are consumed by HVAC and lighting. The EU has also put forward a requirement to make all new houses zero-energy by 2020.

The team from DTU consists of approximately 60 engineering students from various study backgrounds, ranging from architectural engineering, sustainable energy to computer science. Apart from the study background the team is also very diverse, it includes 13 different nationalities.

To make the EMBRACE project possible 31 sponsors have contributed with technical know-how, products and financial subsidies.

2.2.2 The ten contests

The ten contests are in Architecture, House Functioning, Engineering and Construction, Communication and Social Awareness, Energy Efficiency, Urban Design, Transportation and Affordability, Electrical Energy Balance, Innovation, Comfort Conditions, and Sustainability. The total number of points available from these contests are 1000 points.

The entire competition is evaluated through juries, monitoring and task completions.

- The Architecture contest is evaluated on the design coherence, the flexibility and maximisation of the space used by the house, the technologies and the bioclimatic strategies used.
- The House Functioning contest is evaluated on the functionality and efficiency of the appliances used in the house.
- The Engineering and Construction contest is evaluated on the structure of the house, the envelope, and the functionality of the electricity, plumbing and solar system.
- The Communication and Social Awareness contest is evaluated on how the teams have found creative, effect and efficient ways of communicating the project.
- The Energy Efficiency contest is evaluated on how the systems and the house is designed to reduce energy consumption.
- The Urban Design, Transportation and Affordability contest is evaluated on how the housing units are grouped in a social and urban context.

- The Electrical Energy Balance contest is evaluated on electrical energy self-sufficiency and efficiency of the energy balance.
- The Innovation contest is evaluated on the innovation made based on changes from previous competitions.
- The Comfort Conditions contest is evaluated on the interior comfort parameters like temperature, humidity, acoustics and lighting.
- The Sustainability contest is evaluated on the sustainability of the project and the reduction of impact on the environment.

To focus on as many aspects of each contest the DTU team is split into ten sub-teams, which all try to compromise with the other sub-teams to include as much of the important aspects from the individual sub-team. This approach will hopefully make the project better in all of the ten contests.

2.3 EMBRACE

The EMBRACE project is based on the main concepts and technologies from the 2012 house, Fold, with a new context of modern dense cities. It is designed for two-person families. The house is seen in Figure 2.3.1.

The reason it is called Embrace is because of the embracing envelope of the split of the house into two parts,



Figure 2.3.1: EMBRACE in Versailles

a thermal envelope (the dwelling) and a weather shield. The weather shield protects the building e.g. from rain, which means it can be used for most of the year.

The weather shield can be used for multiple dwellings when they are placed next to each other. The area under the weather shield is called the sheltered garden and is considered semi-private/semi-public areas, which makes it easier to get in contact with neighbours. To keep a private sphere for each dwelling, a private terrace is made on the first floor.

The weather shield integrates small photovoltaic panels (PV) in a special pattern, which provides solar energy as well as shading for the sheltered garden. The weather shield is seen in Figure 2.3.6, the black marks are the PV panels. It is projected that the large PV-panels can produce around 185% of the electrical



Figure 2.3.2: Ground floor and stairs.



Figure 2.3.3: Ground floor with kitchen and living room.



Figure 2.3.4: Ground floor with kitchen and living room.

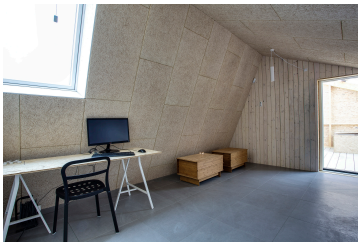


Figure 2.3.5: First floor and door to private terrace.



Figure 2.3.6: Sheltered garden below the weather shield.



Figure 2.3.7: Sheltered garden with hobby room.

consumption of the house.

Solar thermal collectors are placed above the thermal envelope, which provides thermal energy. A skylight is also integrated in this to get daylight into the dwelling.

The entire weather shield is opened to enforce better natural ventilation.

To nudge the occupants of the house to use the sheltered garden, the size of the thermal envelope has been reduced, so that it only includes a bathroom, a living-room, which includes the kitchen and the technical core on the ground floor. And a bedroom on the first floor. The ground floor is seen in Figures 2.3.2, 2.3.3 and 2.3.4. The first floor is seen in Figure 2.3.5, the door seen in the Figure leads to the private terrace.

In the sheltered garden a shed is also included, this room is both meant to be used as a hobby room and as a place for shareable appliances, which can be shared with neighbours, e.g. washing machines. The hobby room is situated right below the private terrace. The wall with the window, seen in Figure 2.3.7, is the hobby room.

The technical core, includes all the technical equipment used for the house, including the HVAC systems and the IHC/PLC systems.

2.3.1 Urban context

The house is designed to fit into the Copenhagen CO₂-neutral 2025 plan, which consists of a new smart-grid in a "dense/lowrise/mix-use" urban area around the northern harbour area of Copenhagen called Nordhavn. However the concept is open for adaptation.

Copenhagen is growing in population every year and to accommodate the denser population of this soon-to-be mega-city, the non-utilised space on the roof of existing buildings can be used. Here the EMBRACE house can easily fit, and with the introduction of smart-grids the surplus energy produced can be used by the existing buildings below. The project tries to put a suburban area atmosphere into the city center by creating a community of EMBRACE dwellings side-by-side on top of existing buildings in the city. Instead of having people live in the suburbs and work in the inner cities commuting back-and-forth, it will be more attractive for these people to live in an EMBRACE house and hence the need for a car is reduced and they will be more likely to use public transport and bicycles, and thereby reducing the overall pollution and the pressure on the existing infrastructure.

2.3.2 Competition

The house is being transported to France after the construction of it on DTU. This is the reason why it is built in easily prefabricated modules, making it easy to transport and assemble. The house is mainly built using wood, because this is more sustainable and it is lightweight, which is important since these houses should be put on top of existing buildings.

The rooms are strategically placed so that piping and wiring is kept to a minimum from the technical core. This also makes it easily maintainable.

2.3.3 Save, smart and share

The keywords for the project is life quality, sobriety and sustainability. Save, smart and share.

- The EMBRACE house is Smart because of its design where innovative systems and architectural design is combined, this is done by having all involved parties take part from the very start of the project. This makes the project more integrated. The smart part also refers to the usage and control of the house.
- The EMBRACE project tries to Save energy, space and money. Saving is a main parameter of the Solar Decathlon competition. The house is build by using sustainable materials, and the energy required for the construction and overall use throughout the life cycle of the house is reduced to a minimum. By building on top of existing buildings it also saves space. And by the use of solar energy it reduces the amount of energy taken from natural resources, and it can even send the surplus energy into the smart grid, and e.g. thereby to the host building.
- The EMBRACE house is designed for Sharing. The sheltered garden creates a community for neighbours, where they get to know each-other better and share common things. The house also includes the dwelling which makes it possible for the occupants to have their privacy of their own home.

2.3.4 Appliances and control systems

To reduce electrical consumption to a minimum, artificial lighting is kept to a minimum by using daylight and efficient LED lamps. Also only efficient appliances have been chosen. For heating and cooling an efficient air-to-water pump is used. When heating is required the pump circulates cold water in the pipes

and the hot air or surface e.g. at the middle of the day, when the sun is highest, will heat the water. When cooling is required the pump circulates hot water through the pipes and the cold air or surface, e.g. at night will cool the water. The hot and cold water is kept in storage tanks for later usage.

By fitting the EMBRACE house to an electrical smart-grid makes it possible to schedule energy consumption according to tariff-pricing of electricity. E.g. it is possible to turn on home appliances when the tariff-prices are lower. Since the house produces more energy than it consumes.

All the electrical components are interconnected to the control systems, the IHC together with the PLC. This makes it possible for the house to function without interaction from the occupants. But it also allows the occupants to control the house through tablets. The control systems can control the windows, shadings, heating, cooling, lights and sockets.

The systems are controlled through routers in the dwellings which sends information to a cloud service. The information gathered makes it possible for occupants to monitor their use of the house, their energy consumption and production and thereby take action on reducing the energy consumption on different appliances or devices used in the dwelling.

2.3.5 Results from SDE14

The competition in Versailles has finished and the final scores for each contest is given below.

- Architecture: 78 points (ranked #12)
- Engineering & Construction: 69.6 points (ranked #7)
- Energy Efficiency: 71.84 points (ranked #9)
- Electrical Energy Balance: 79.22 points (ranked #7)
- Comfort Conditions: 99.23 points (ranked #8)
- House Functioning: 90.66 points (ranked #11)
- Communication & Social Awareness: 64 points (ranked #8)
- Urban design, Transportation & Affordability: 101.65 points (ranked #4)
- Innovation: 59.81 points (ranked #9)
- Sustainability: 68 points (ranked #6)

Additionally the project was given 2 penalty points and no bonus points, because of a delay in the construction phase.

The total number of points was 780.01, which ranked the project #8 out of 20. So an advancement of 2 ranks compared to the 2012 house.

The final scores between all the competing houses is seen in Figure 2.3.8.

2.3.6 After Versailles

After the competition in Versailles, the Embrace house will be assembled and exhibited in Universe Experience Park, Denmark.

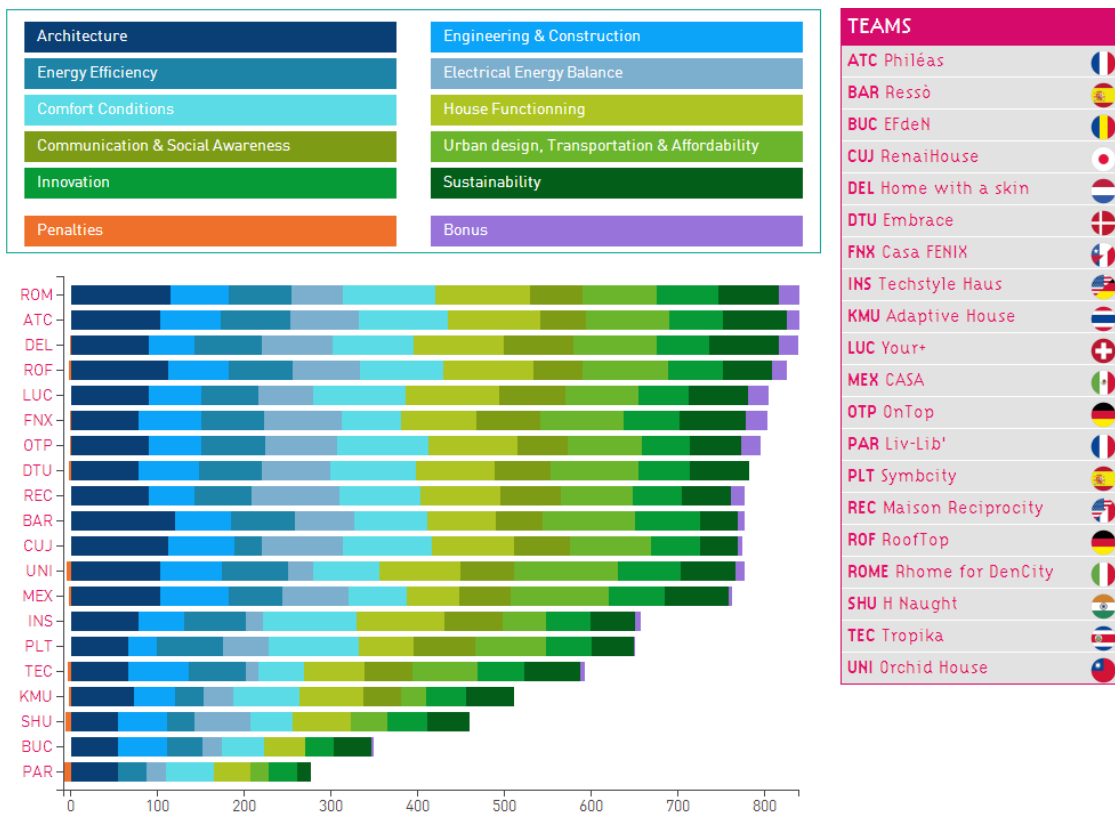


Figure 2.3.8: Final scores

2.4 EPIC Cloud

To be able to control and monitor the EMBRACE house a server needs to communicate with the home control systems.

In the Fold house for the 2012 competition, a server was situated in the house, which gave several significant problems, such as unavailable access from the internet, and the energy consumption was so high that most of the energy produced by the house was consumed by the server, which is not very convenient when participating in an energy saving competition.

The problems faced in 2012 means that a new solution must be found. One solution is to locate the server in the cloud, and with the rise of cloud computing and services, this makes a lot of sense. However since most of these services are relatively cheap, the costs are not justified for a student project.

Another solution which is also cloud based, is to have a server on-premises to act as a cloud, this solution requires a server to be located at DTU, and being accessible from the internet, on a select range of ports.

This solution is chosen for this project.

Since the cloud needs to be on the same network as the home control systems, a router is connected to the home control systems in the house and the IP traffic is relayed through port forwarding, from the WAN

IP. This makes it possible for the cloud to connect to the home control systems. Another solution is to run a VPN server on the router, this will also encrypt all traffic, however this option is not possible for the competition, since the traffic is relayed through several NATs.

The cloud is called EPIC cloud. EPIC is an acronym for Energy Preservation and Information Computation. The name represents the desire to preserve energy while being able to compute as much information as possible. It is able to preserve energy by not consuming any energy from the house, and since multiple houses can be connected to the same cloud, the amount of energy consumed is shared by these. And hence making the cloud more efficient. If a cloud provider is chosen, these also pool all the servers in big server farms, where they usually use a green source of energy, e.g. from hydroelectric power. And the cooling is usually provided from the cool air outside the farms, which are located up in the colder northern regions.

2.4.1 Logical components

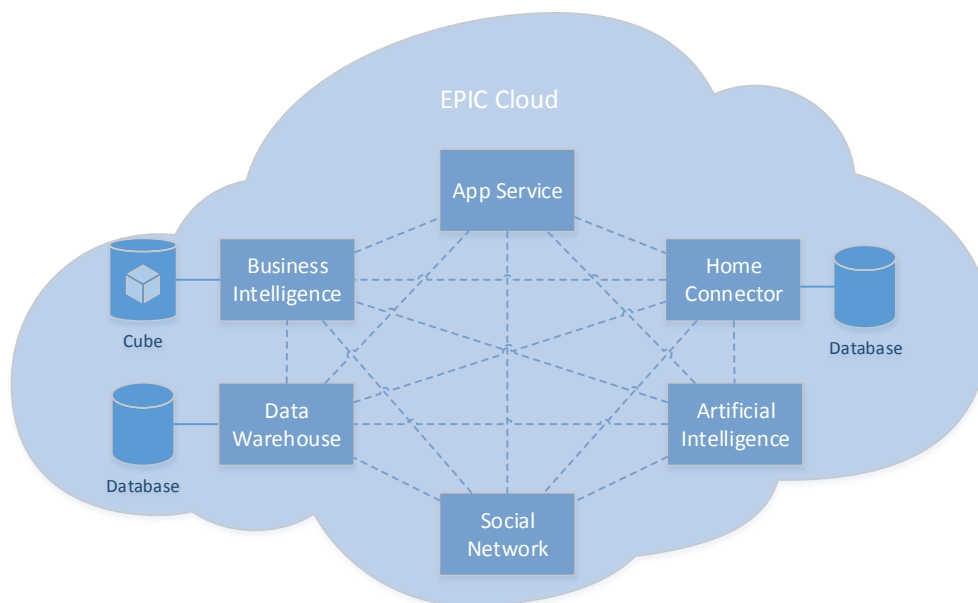


Figure 2.4.1: Logical components of EPIC cloud

The cloud, seen in Figure 2.4.1, is split into the following logical components: Kademia P2P network, Home Connector, App Service, Artificial Intelligence, Social Network and Data Warehouse/Business Intelligence.

- The Kademia P2P network is the backbone of the communication between the logical components, it is based on the Kademia DHT system, and provides a publish subscribe system for messaging in the network of components in the cloud.
- The Home Connector component enables the cloud to communicate with the home control systems, like the IHC, the PLC and the Uponor system (an HVAC system). This component also includes a logger feature which sends out current values of the devices in the home control systems to the

network.

- The App Service is the single source for the applications, e.g. a tablet application, to communicate with the different components in the P2P network over the internet. E.g. it is used to communicate with the Home Connector to control the lights and the Business Intelligence to get insights.
- The Artificial Intelligence component makes models of the usage patterns of e.g. the lighting using motion detectors and tries to optimise, when lighting should be turned on and off.
- The Social Network component enables the occupants of the EMBRACE houses to share events, favours and produced products.
- The Business Intelligence component enables the occupants and researchers to gain insights to e.g. how the energy consumption is consumed, and how the houses are being used in terms of the measurements made from home control systems.
- The Data Warehouse component enables the Business Intelligence component to obtain information necessary to produce reporting (insights) for different measures and dimensions. This component also collects all the available information from the network as well as other data sources.

2.4.2 Technology used

The technologies used for this thesis is Microsoft .NET Framework, C#, Microsoft SQL Server, Microsoft SQL Server Analysis Services, Microsoft SQL Server Integration Services.

2.5 Business Intelligence

The field of Business Intelligence (BI) comprises several activities, applications, theories, architectures and best practices. It primarily concerns analysis of raw data from operational source systems of a company, e.g. Customer Relationship Management (CRM), Enterprise Resource Planning (ERP) and Point of Sales (POS) systems.

These systems usually contain a vast amount of business critical information, which is typically "hidden" by application layers. This makes it harder for individual employees of a company to analyse the data.

The most significant activities related to BI is online analytical processing (OLAP), querying and reporting of data.

Today companies use BI mostly for decision making support, which can help to cut costs and find new business opportunities, and thereby strengthening the competitive advantage of a company.

Decisions in a company range from strategic, which are made by top-management, to tactical, which are made by middle-management, to operational which are made by common employees. BI applications should be able to support decision making for all levels of employees in a company.

By giving employees access to quickly and easily analyse and make sense of business critical information stored in the operational source systems, makes them better prepared to handle even the smallest decisions, which will improve the overall performance of an entire company.

BI should not just be used for e.g. annual reporting, the reporting facilities should be the basis which is used to help making decisions with hard facts, i.e. the numbers from the reports. Having hard facts can im-

prove any decisions made by employees, and help persuade management to be more confident about taking necessary decisions. But BI is not just about showing reports with hard facts, it is also about analysing the numbers and gain valuable insights into the business. And thereby improve business processes.

Numbers alone does not give much insight or value, the context of the numbers give more insights. E.g. sale figures alone from the two previous months, where the amount is larger one month than the other, might not give much insight into *why* this happened. To gain more insight the BI application should be able to identify why, e.g. what was changed, and thereby giving context to the numbers.

The basis for implementing BI solutions in a company is the raw data sources, these should be extracted, cleaned and transformed to fit the needs and loaded into a proper analytical data format, which ensures the integrity and quality of the numbers, that employees can get out e.g. from querying and reporting. This point is very important, because if the basis or foundation of the numbers are not reliable, then the reporting and querying facilities can not be used for anything. Another point in this area is to make sure that the business actually has the data that is needed to record the individual business processes, otherwise the BI applications might not be able to give the correct and complete picture of these business processes. A proper BI solution should also be able to handle change, because business environments change often and quickly to maintain competitive advantage. If the BI applications cannot handle this, then it is not to much use. A perfect BI solution however does not exist. And instead of making one, the focus should be on the business needs and goals, and thereby try to help support decision making as much as possible, e.g. by analysing the current decision making and how management currently uses numbers.

Return On Investment (ROI) can be significantly improved by focusing on the business needs and goals instead of making a perfect BI solution. The ROI on BI solutions range from smaller to very large figures, which is why it is being deployed in more and more companies around the world.

BI applications can help in several ways to optimize a business like making sure the entire business is aligned with the overall strategy of the company, this can e.g. be done by setting up Key Performance Indicators (KPI) which measure the performance based on the overall strategy, this makes it possible to quickly identify areas which are not aligned with the overall strategy.

It can also help in benchmarking, from overall to specific, areas of a company, e.g. how the supply chains are performing. This information can even be used for leverage and make better deals with suppliers. In addition to this, BI applications can help identify more profitable clients or customers and find ways to keep them satisfied by the services provided, e.g. by analysing areas where the company can give some goodwill, i.e. better deals for the client but also for the company.

The field of Business Intelligence is constantly moving, especially in the techniques for analysing data and giving proper feedback about this information e.g. to advice and help decision making. The field of modelling the data to fit the requirements needed by BI, has however been stable for many years now. The most commonly used data modelling for BI is dimensional modelling. One of the most acclaimed books on dimensional modelling is the Data Warehouse Toolkit. The first edition of which came out in 1996. And not much of the techniques and best practices have changed since then.

2.5.1 Motivation for using BI

Nowadays most BI solutions are so generic, it is also possible to use these applications for much more than just gaining company insights. E.g. gaining insights from an intelligent house like the EMBRACE house should be possible by using BI applications. These insights could be about the energy consumption but also about the overall usage and functioning of the house.

Business Intelligence brings together the world of business and the world of data, this is what makes this topic interesting. The fact that the business needs of getting answers to many different kinds of questions can be served by the vast amounts of data, often hidden away in operational source systems.

2.6 Data Warehouse

To be able to provide business intelligence a data warehouse (DW) is often essential, this provides the analytical data to be processed by the BI applications. Data warehousing and business intelligence (DW/BI) systems are often delivered together as a BI solution.

The word data warehousing is composed of data and warehousing, i.e. the concept of having products available from all shelves in a warehouse. This metaphor illustrates quite well the intention of building a data warehouse. It is providing data in an easy way for end-consumers to take down from shelves as they desire.

The biggest difference between operational source systems and data warehouses is that while operational source systems does operations, e.g. update and add, on records, data warehouses does operations on large sets of records. Operational systems are fully optimised to use their databases for themselves, meaning it might result in poor performance if a BI application uses this data source directly to execute large queries. Operational source systems are used by the employees in a company for the individual business processes. While data warehouses are used by the employees in a company to evaluate and test performance of both the individual and subsets of the business processes.

Operational source systems are hence built to handle individual transactional records efficiently, these systems usually updates the state of the processes, by records, without recording the individual transactional step. While for the DW it is almost required that steps in the processes can be traced using transactional records. However DW users rarely use the individual records, they just use them for summarising the processes executed. E.g. to see how long a process has taken, from and to a specific step in the process.

Some operational source systems do not keep records for long, this is however essential for DW systems since this provides historical context of the entire organisation.

From the above it should be clear that using a BI application on top of existing operational source systems without a data warehouse can result in unreliable data, poor performance and hence poor usability of the DW/BI system.

2.6.1 Motivation for using DW

Data Warehousing is the glue that binds the data from operational source systems and the BI applications together, without it, the BI applications are not as useful as they can be. This is why Data Warehousing is an interesting topic to look into.

And the fact that it can be built upon operational source systems that control an intelligent house makes it a completely new and innovative way of looking at the data being collected. Also since all houses in Denmark over the coming years will have digital electrical meters, this collected data can help researchers and even the occupants of these houses to easily gain insight into what their energy consumption is being consumed upon and if they have control systems, they can even gain insights into overall functioning of the house.

2.7 Scope

This thesis will investigate the use of data warehousing to deliver Business Intelligence reporting for data collected from the EMBRACE house, which is the entry from DTU at the Solar Decathlon 2014 competition.

BI reports can be used for analysing and benchmarking the EMBRACE house on different measurements and parameters.

Business Intelligence is being deployed more and more in corporations around the world to make better decisions on business strategy and management. In this thesis BI will be used to help occupants and researchers of the EMBRACE house to save energy and use the house smarter and thereby making it a more intelligent house.

A large part of the process is to make an extract, transform and load (ETL) system, which can retrieve data from different data sources, transforming or cleaning the data, so it fits the needs of doing BI analysis, and lastly put the data into a data warehouse, which is consumed by BI applications. The data will originate from different data sources i.e. the operational source systems, which are the home control systems.

An essential part of BI reporting is the ability to analyse and interpret the data, which should give the analyst knowledge on the current and historic facts on the functioning of the house. And thereby make recommendations e.g. on how/where to reduce the energy consumption.

Data warehousing is a part of the umbrella term Business Intelligence, which covers theory, methods, processes, architectures, and technologies.

To be able to build a data warehouse, the data should use dimensional modelling, which consists of facts and dimensions.

The reporting can then be done through OLAP cubes, which consume the facts and dimensions, and lets end-users slice and dice cubes for relevant data in BI analysis tools.

One way of evaluating the performance of a house is the use of key performance indicators. This thesis will only cover how to create a key performance indicator and how they can be used.

A data warehouse is a good foundation for doing e.g. data mining and the use of machine learning can help with this. This thesis will however not consider these techniques.

2.8 Summary

Intelligent houses use control systems for home automation to control e.g. HVAC, appliances and lighting. These can be integrated to make houses more intelligent and thereby help occupants use the houses more efficiently e.g. to reduce the overall energy consumption.

The Solar Decathlon Europe competition is an international competition, where the overall goal is to make energy efficient houses. The competition consists of ten contests, which evaluates how well the houses are as an energy efficient houses and how well they would be to live in.

DTU is participating for the second time in the Solar Decathlon Europe competition. In 2012 the house, Fold, was ranked #10. For the 2014 edition the entry from DTU is EMBRACE, which consists of a thermal envelope and a sheltered garden. The house was ranked #8 in the competition.

For the EMBRACE house a cloud-based control system is made. The cloud consists of multiple logical components including a Data Warehouse and a Business Intelligence component.

Business Intelligence is mainly used for decision making support in larger companies, where it is built upon operational source systems. Common usage of BI is reporting and querying, which is the basis for the decision making support analysis.

Data Warehouses are often an essential part of delivering BI solutions, because it is the data foundation. Data Warehouses consume data from operational source systems, cleanses it and makes it available for BI analysis tools.

The reason for focusing this project on DW and BI is that BI solutions and applications are now so generic that they can be used for multiple purposes, e.g. for getting insights into the functioning of the EMBRACE house. Delivering a proper BI solution includes a DW, that can contain a vast amount of data and expose it in a way, such that it can easily be used by end-users.

Chapter 3

State of the Art

"Innovation is taking two things that already exist and putting them together in a new way."

—Tom Freston, 2012

Contents

3.1	Third Normal Form	17
3.2	Dimensional Modelling	18
3.3	Facts	20
3.4	Dimensions	26
3.5	Star Schema	33
3.6	OLAP Cubes	35
3.7	Design Process	39
3.8	Data Warehouse and Business Intelligence components	40
3.9	Summary	41

This chapter provides an insight into the state of the art in data warehousing.

3.1 Third Normal Form

Relational database design often use normalisation and the third normal form. The point of using this is to normalise the data as much as possible, and thereby removing redundancies and dependencies, which affect the amount of operations required to insert (create), read, update and delete (CRUD) data. E.g. if redundant data is present, these will have to be updated as well when an update is done on one part of the data. This also wastes disk space.

Inconsistent dependencies can be made if data in one table is missing a link or relationship to another table, hence the path to find the right data, can sometimes be impossible, especially if several tables exists. Links or relationships can also be broken, e.g. if updating of the data is not done right.

Normalisation can be achieved by creating more tables and create relationships between them. Normalising a database happens in accordance to rules, these rules are called normal forms (NF). If a database

design is compliant with the first rule, it is a database in first normal form (1NF). The rules build upon each other, which means that if a database is in third normal form (3NF) it is also in first and second normal form (2NF).

The first three normal forms are the most widely used in most applications. However there exists a few more normal forms other than these first three. These are however not used very often, mainly because they do not hinder any functionality of the database, even though they would provide for a better database design.

It is important to note that even though most relational database designs use the third normal form, they are not necessary for an application to run, however if they are not used, the application should be prepared to handle the redundant data and inconsistent dependencies.

More detail about the three first normal forms is given in Appendix A.

3.2 Dimensional Modelling

Even though complying fully with the third normal form rule is considered best practice for database design, the data is not optimally designed for fast querying of many records e.g. required by BI applications, this can however be done by using dimensional modelling.

Dimensional modelling tries to help the end-users easily understand the content of the data. Dimensional modelling is considered the best and preferred way of storing analytical data to be processed by BI applications.

The dimensional structure of tables makes the data much more simple, in the way that it is presented, however because of this simplicity, it cannot be normalised to the same degree as with 3NF. This is the most significant difference between dimensional models and 3NF compliant models.

Dimensional modelling is mostly used for the presentation area for end-users, i.e. in a data warehouse.

The basics of dimensional modelling is the concept of cubes, where edges are different dimensions and when this cube is sliced and diced, the set of facts or measurements for these defined dimensions are revealed.

An example of this could be a house where temperature and humidity measurements are done by devices located on different floors, and the measurements are made every hour. In this description the dimensions for the cube reveal themselves as one for devices, one for floors and one for time. The facts are the temperature and humidity measures, which are recorded on single dots inside a cube along the dimensions. The cube is seen in Figure 3.2.1. Now if the cube is sliced and diced so that the measurements made by a specific device is selected, two floors are selected and a range of time starting from morning to noon, then the set of measurements revealed by the cube will be the temperature and humidity for these specified dimensions. The above example shows how tangible and simple the idea of dimensional modelling is, this is the main reason why it is so often used for BI applications. And because measurements are recorded along all dimensions/axes of the cube the data is quickly retrievable.

Dimensional models can be stored in any relational database management system (RDBMS) because they still use the same data structures as traditional relational database designed structures. As discussed above

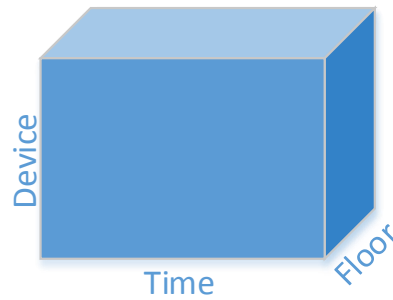


Figure 3.2.1: Example of cube with floor, device and time as dimensions

3NF tries to remove as many redundancies as possible, by making different tables for each separate entity, this model performs well for operational source systems because they require less operations to maintain the data, e.g. when CRUD operations are performed.

This model is however not optimal when it comes to dimensional modelling because of the simplicity required by the model. The simplicity ensures that users can get the data that is needed without having to navigate and joining through a lot of foreign tables. Even though the dimensional model also uses foreign tables, it is kept to a minimum by only having foreign tables for each dimension. Where a traditional entity based relational model, would have several tables for each values stored in each dimension. The data stored in both models can model exactly the same data, it is only the format or structure of the data that is different for these.

3NF relational database models are sometimes called entity relationship models, where entity relationship diagrams are often used to represent the design. However since dimensional models also consists of joined/linked relational tables, like traditional entity relationship models, entity relationship diagrams can also be used for these models.

Most BI applications will execute queries defined by the end-user, and these can be very unpredictable. This unpredictability makes it nearly impossible for RDBMS systems to perform efficiently on a fully 3NF compliant relational table model. However when the data is modelled using dimensional modelling, it can perform better because of the minimal usage of navigation and joining of foreign tables.

Dimensional models are very flexible, which makes it possible to modify the model during its lifetime, without causing major modifications to the existing infrastructure or applications.

- New measures can be added to a fact table, this is done by adding a new column with the new measure. The column should be nullable, if there exists fact rows which will not contain the new measure. It is important when adding a new measure that this complies with the existing grain, since it would violate the rule that grains should never be mixed, otherwise. If the new measures are at a different grain, it belongs in another fact table. Grain defines the granularity of rows in a fact table. These are discussed more in detail later.
- New dimensions can be added to fact tables as long as they do not change the grain of the fact table. The new dimension is added to a column in the fact table and a reference (foreign key) for the dimension row is populated to the fact table. If not all facts need the new dimension these should be

populated with a foreign key to a default dimension row, which is an "unknown" or "not applicable" dimension row.

- New attributes/columns can be added to a dimension table. If the value of the new attribute/column is only applicable for a subset of the dimension rows, these should be populated with a "not applicable" text string instead of a null value.

Once the new changes have been checked and validated, they can be introduced in the BI applications, which will then need to be modified and reloaded to accommodate the new changes.

3.3 Facts

Facts or fact tables contains the measurements in a dimensional model. This is also why facts are sometimes referred to as measures. These measurements comes from the operational source systems. A fact could e.g. be the temperature or humidity of a house.

Since these measurements will be done quite often, it is important not to store these in many different places, e.g. for each floor. Instead the measurements should only be contained in the operational source system and a single data warehouse. This ensures that data is consistent and it reduces the amount of space required.

There should only be one row for each measurement event. I.e. the same measurement should not be replicated or otherwise modified, because this might cause incorrect calculations later. This also includes "zero"-measurements, e.g. measurements where nothing has happened yet, these measurements should not be added as a new row in the fact table, since these will cause incorrect calculations as well as consume a large amount of space, since a "nothing-has-happened-yet" measurement will occur more often than an actual measurement.

One of the main principals in dimensional modelling is that a row in a fact table corresponds to a single measurement event. And hence a single row in a fact table has a one-to-one relationship with the grain of the fact table, which represents a single actual measurement event.

Measurement events in a business context is usually bound to a business process. Where the measurements comes from different steps in a business process. The measurement is a performance metric which can be analysed. Since there can be made several measurements before, during and after a step in a business process, these measures can spawn into several fact tables depending on the grain and intervals that they are measured by the operational source systems. However only the key metrics should be considered first, these are usually related to the key steps in a business process.

Measurements are bound by the source data realities, if the measurements are not done e.g. by an operational source system, then they cannot be analysed. So if it needs to be analysed the data needs to be clearly identified in the operational source systems.

Measurements in general can range from numeric values to textual string values. The most useful measurements in dimensional modelling is numeric values, preferably the ones which are additive. Most fact tables only use numeric values.

An example of a fact table is seen in Figure 3.3.1. More detail on the above points are given below. Addi-

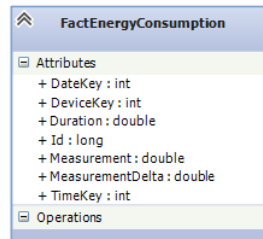


Figure 3.3.1: Example of a fact table

tional dimensional modelling techniques regarding fact tables are given in Appendix B.1.

3.3.1 Structure

The structure of fact tables is a combination of numeric measured values together with foreign keys to dimension table rows. The numeric values in a single row (i.e. at the lowest grain) should correspond to a single event that happened, e.g. a measuring of the temperature.

The purpose of the fact tables are to be used during calculations which are used by the queries that end-users request.

Derived facts can either be put into a fact table or they can be added to a view. It is important that all end-users then use the view instead of the physical fact table, otherwise this might confuse end-users, and they might try to do the derived calculations themselves which can be error-prone.

It is possible to have measurements in multiple or single fact tables, usually if the grain is different and/or the dimensionality is different, i.e. that some facts have relations to more dimensions than others, then separate fact tables should be used. If they have the same grain and/or dimensionality it is up to how the end-users use it. If the contents/measurements have different contextual meaning they should be split into separate fact tables. The reasoning for having measurements with the same grain and/or dimensionality in the same fact table is that it is possible to add a dimension which separates the measurements. E.g. a type dimension, which can be used to separate the measurements.

Fact tables are often named with a "Fact" prefix.

3.3.2 Keys

Fact tables should not have primary keys, because they are not suppose to be browsed though these. However an ETL system might assign the fact table rows with surrogate keys to quickly identify rows, e.g. if the rows needs to be updated or removed.

Fact tables consists of several foreign keys to primary keys of dimension table. These dimension table primary keys are used to browse the fact tables. Facts include a foreign key for each of the dimensions, which have a context with the fact. This could e.g. be a foreign key for a row in the time dimension, a foreign key for the sensor device dimension and a foreign key for the floor dimension. The facts can then be found by using e.g. the time dimension, where a row can give give multiple results (a set of measures) from the the

fact table.

When all foreign keys in a fact table match the primary keys of a dimension table, the table satisfies referential integrity, this integrity is usually handled by the RDBMS system.

When specific facts are needed in a retrieval from a fact table, a composite key is usually used. The composite key combines the foreign keys to a single key, which should make it possible to get the individual facts, if multiple facts are found by the full composite key, then duplicates might exist. I.e. if a fact table is missing a relation to a dimension, e.g. the device dimension, then facts could be misinterpreted as duplicates, even though they might not be.

Keys are discussed more in detail in the dimension section.

3.3.3 Additivity

Additive means that the numbers can be added or summed when a set of these numbers are used for a calculation. When slicing and dicing a cube, large sets of numbers are usually retrieved and one of the most simple calculations to do with these large amounts of numbers is to sum them. This requires that the numbers can be added. And the resulting value is as valid as the single measurement. There exists three types or categories of measurement additivity, the additive, the semi-additive and the non-additive.

The additive are the ones that can be summed across all dimensions. The semi-additive are the ones that can be summed across some dimensions. Most semi-additive measurements cannot be summed across the time dimension, but they can be summed across others. The non-additive are the ones which cannot be added across any dimension.

The semi- and non-additive measurements can usually be used with counts and averages. Non-additive measurements are usually ratios, because they are "pre-calculated" they cannot be summed. Since these ratios might be pre-calculated they should, if possible, be calculated by the BI applications instead and the base measures for the calculated ratio should be stored in the fact tables. This approach might not always be possible if the actual measurements give ratios.

In a house context an example of an additive measure is the amount of energy consumed in kWh, but only when the amount is reset every time the measure is read. This gives a measurement in a transaction. These measurements are additive, since they can be summed across all dimensions.

An example of a semi-additive measure is the amount of energy consumed in kWh, when the amount is not reset every time the measurement is read. I.e. since this measurement grows for each unit of time, the measurement cannot be added across e.g. the time dimension, since this would result in invalid values. E.g. if the amount of energy consumed at 1 o'clock is 12 kWh and the amount of energy consumed at 2 o'clock is 14 kWh. These two numbers cannot be added to get a valid total across the time dimension. The measurements can however be summed across e.g. the metering devices dimension. So if the amount of energy consumed is 12 kWh for the lights meter and 12 kWh for the HVAC meter, then the total amount can be summed to 24 kWh, which is valid. For the time dimension for this measure a count could be calculated instead, so that it can be calculated that two measurements were recorded from 1 o'clock to 2 o'clock.

An example of a non-additive measure could be temperature measurements. These cannot be added across

any dimension, because the temperature is a fixed range. It does not make any sense to sum across the time dimension or a temperature sensor device dimension. E.g. for a time dimension a measurement of 20 degrees Celsius at 1 o'clock and a measurement of 21 degrees Celsius at 2 o'clock. Summing these measurements would give a total of 41 degrees Celsius, and this temperature never occurred. The same goes for a device dimension, if one temperature sensor measures 20 degrees Celsius and another temperature sensor measures 21 degrees Celsius, a summed total of 41 degrees Celsius does not make any sense. However it is possible to calculate an average instead for non-additive measures, so that for both the time dimension and the device dimension an average of 20.5 degrees Celsius is calculated, which is a valid value.

Measures which record a static level e.g. a measure of intensity like room temperature are inherently non-additive for some dimensions, most likely the date and time dimension. These measures can however be used together with the date dimension by using averages and counts.

It should be noted that when calculating averages over a date or time dimension, this should not necessarily be done by using the SQL AVG function, since this averages the values from the resulting fact table rows and not against the number of rows that a dimension table has. E.g. if a weekly average of energy consumption is needed, and the energy consumption is measured twice, then the AVG function will give an average by summing these values and dividing it by two, however a week has seven days (seven rows in a date dimension) so the average should have been divided by seven not two. Hence it should be noted that there are two different kinds of averages and both can be used, but for different measures.

From above it is seen that the most flexible measurements are the fully additive because they can easily be summed and used across all dimensions related to a fact table.

3.3.4 Grain

The specific level of detail in which measurements are done is called the grain of a fact table. E.g. if temperature measurements are done every minute, the grain of the fact table containing these measurements is said to have a one minute time grain. Note that this grain is related to the time dimension. The grain can also refer to other dimensions or more likely several dimensions, e.g. the devices which do the measurements, which defines the grain of the fact table. Hence the grain defines what a single row in a fact table corresponds to.

It is important that the grain is never changed during the lifetime of the fact table. E.g. going from a time grain of one minute to a "summed" grain of ten minutes. This could cause multiple measurement events to be calculated more than once, and would hence give incorrect calculations. If the grain is to be changed, the new grained measurements should be placed in another fact table. It is also important that grains are not mixed in a single fact table. Each declared grain should be in different fact tables. E.g. should measurements done every hour not be mixed with measurements done every seconds, especially if the hourly measurement is an accumulating measurement. Also measurements done for e.g. the temperature should not be mixed with the energy consumption, because these are from different contexts.

In general when considering the grain of a fact table, three different types of grain exists, transactional, periodic snapshot and accumulating snapshot. Transactional is the most common, since the facts are usually retrieved from the operational source systems, which mostly use transactional data.

The lowest level of granularity is always preferred when choosing the grain. These atomic measurements can be used for much more than accumulated measurements, because they can always be rolled up to the accumulated measurements, but it is not possible to "drill down" into more detail with an accumulated grain.

The grain is dependent on the data realities of the operational source systems, if the desired grain is not available from the measurements done by the operational source system, the grain should be redefined to what is possible.

3.3.5 Null values

Non-value measurements for the numeric values might occur, these should be saved as null values. Most RDBMS systems can gracefully handle these null values without miscalculations. E.g. sum, count and others either ignore these values or they can be set to ignore the values.

It is important to tell the BI application that there might be null values in a fact table, so that these can be handled correctly, i.e. so that it can set only to get non-null values from the fact tables.

Non-measurements i.e. inserting a row in a fact table when nothing has happened, should not be done however, because this would fill up most fact tables with useless information (for transactional fact tables). Null values should never occur for foreign keys to dimensions, if a "unknown", "missing" or "not applicable" dimension is needed, this should be added to the dimension table as a separate row and the key for this dimension row should be set in the fact table instead. For RDBMS systems these key columns should have a not null constraint.

3.3.6 Types of fact tables

There are three types of fact tables, transaction fact tables, periodic snapshot fact tables and accumulating snapshot fact tables. Each of them can be used in separate fact tables, they should however never be combined, because this would violate the grain. The periodic and accumulating fact tables are described in Appendix B.1.2.

3.3.6.1 Transaction based

For transaction fact tables a row corresponds to a transaction, this transaction could be an event that occurred during or after a process has been done, e.g. it could be from a temperature measuring event. Transactions usually come directly from the operational source systems, which are mostly designed to handle transactional records. The transactions might also be from calculations done in the ETL system.

The best transaction facts or measurements are the ones that are at the most atomic level possible, these are usually the transactional grain. These measurements are more dimensional and robust in the sense that they can be sliced and diced the most, i.e. an end-user can drill all the way down to the actual records, and not just some aggregate measure.

Transaction fact tables often have the same amount of rows as the transactional records of the operational

source systems, i.e. no rows are usually added beside these, which means that if a transaction event occurred in the operational source system, one row would be inserted in the fact table.

Transaction fact tables always have all the dimension foreign keys filled, because an event could not occur if the context from the dimensions did not have that context dimension row. I.e. since the fact row was inserted a number of context dimension rows where true at that event, these dimension row foreign keys will hence always be filled.

Transaction fact tables are usually the largest fact tables, because they include a row for every transactional event, so considerations about storage capacity should be considered, e.g. by estimating the number of rows which will be put into the fact table over a defined time period.

3.3.7 Factless

A special type of fact tables is the fact tables referred to as factless fact tables. These tables record that an event occurred, but no numeric values, measures or facts were recorded. It could e.g. be that the alarm went off in a house context. Then the fact table would include what dimensions were true when the event occurred, but since there are no measurements, this is a factless fact table.

Factless fact tables are good to use when analysing what did not happen. This could e.g. be modelled by having two factless fact tables, one for the possible events, and one for the events that occurred, taking the set difference for these two tables would give the events that did not happen.

Factless fact tables are also used for many-to-many relations between dimensions. E.g. a device grouping in a house, where a number of devices can be in a number of groups, and a number of groups can contain a number of devices. The devices and groups would then be dimensions and a factless fact table could be used for the relationship between these two.

3.3.8 Numeric values as dimensions

Numeric values are not only facts, sometimes they can be used as dimensions. This is often determined by whether the values are being used for calculations or for filtering/grouping. If they are used for calculations the numeric values belong in a fact table. And if they are used for filtering and grouping they belong in a dimension table. If the values fall into intervals/ranges these can also be modelled as dimensions, where e.g. a range between 12-24 degrees Celsius can be modelled as a single dimension row. When numeric values are used for dimension rows, they should be accompanied with a text string which includes the value, which gives the numbers more context.

Numeric values can also both be used as dimensions and facts, and hence should be placed in both if the numbers are both used for calculations as well as for filtering/grouping in reporting.

Usually numeric values which are continuous values belong in a fact table, while numeric values which are discrete should be made as dimension rows.

3.3.9 Units of measure

When multiple units of measure is, can or should be used, they should not be inserted as separate measures (columns) in a fact table, instead a specific unit of measure should be used, and then this can be calculated into the other unit of measures by using defined conversion factors, which are stored in the fact table. The conversion factors should not be stored in dimension tables, e.g. since the dimension table might not be used for a specific reporting, where different units of measure is needed.

The fact tables can then be made into views which includes the different units of measure which the end-users use. This could e.g. be done for power meters which are counters, instead of storing the same information in a multitude of different units of measures in different columns, only the count and the duration between the counts could be recorded in a fact table, and the power in kWh, Wh and kW, W could then be calculated and shown using a view. When using a view the conversion factors might not be needed in the fact tables, since the measures are already shown to the end-users with the specific units of measure. It is important that different units of measure are clearly labelled for the columns so end-users can be sure about which columns to choose.

3.4 Dimensions

Dimensions or dimension tables are the descriptive context labels for the measurements from the fact tables. They describe the "wh"-questions, e.g. "when", "where", etc. of a measurement event. The descriptive context labels are used for filtering, querying and grouping the measurements from the fact tables. The columns are also called "by" words, since they describe the behaviour of how an end-user can see measurements by a dimension table. E.g. if a user wants to be able to see energy consumption by different devices, the device labels would have to be in a dimension column. This makes the dimension columns essential to dimensional modelling and the BI applications, since they provide the constraints and labels that are available in the BI applications. Because of this, it is important to give proper descriptions to the column names and values so they make sense for the end-users, otherwise they might confuse the content of their query. The more verbose the labels are, the more understandable they are for the end-users. Hence codes and specific numbers should not be used as descriptive labels. If the codes have special meaning their verbose meaning should be available as a label, and if needed the code can be added as a separate column together with the verbose labels.

These points should be held in mind when building any DW/BI solution, since these dimension labels will be the foundation of any analysis done by the BI application. They are the only way to access the measurements in a fact table.

Dimension tables usually have several attributes or columns, and usually more than fact tables. These columns are mostly textual string values. Dimension tables usually have fewer rows than fact tables.

The dimension rows have a primary key associated with them, this primary key is used for the foreign keys in the fact tables.

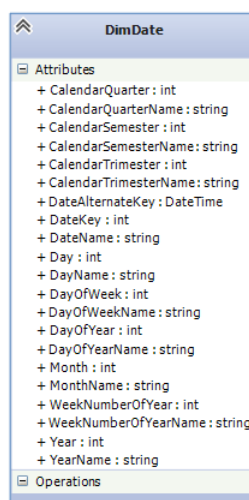
When identifying dimensions, all the discrete textual strings will be attributes/columns of a dimension, these text strings should be consistent with end-users expectations.

A reference from a fact table should only be to one dimension row, where the attributes only have a single value. This makes it possible to identify the context of a measurement in a fact table row. For many-to-one relationships the attribute value might occur for several dimension rows, but only once in a single row, i.e. it should not be repeated in another column in the dimension table.

Dimensions have a tendency to describe hierarchical relationships, they contain a many-to-one relationship in a single dimension table, thereby increasing the verbosity of the columns. This is done to improve query performance and to increase the usability and understandability of the dimensions.

Striving for dimensions which are conformed, i.e. be able to be used across fact tables from different operational source systems is key in a DW/BI solution which integrates these different systems.

An example of a date dimension table is seen in Figure 3.4.1. More detail on the above points are given



DimDate	
Attributes	
+ CalendarQuarter	: int
+ CalendarQuarterName	: string
+ CalendarSemester	: int
+ CalendarSemesterName	: string
+ CalendarTrimester	: int
+ CalendarTrimesterName	: string
+ DateAlternateKey	: DateTime
+ DateKey	: int
+ DateName	: string
+ Day	: int
+ DayName	: string
+ DayOfWeek	: int
+ DayOfWeekName	: string
+ DayOfYear	: int
+ DayOfYearName	: string
+ Month	: int
+ MonthName	: string
+ WeekNumberOfYear	: int
+ WeekNumberOfYearName	: string
+ Year	: int
+ YearName	: string
Operations	

Figure 3.4.1: Example of a date dimension table

below. Additional dimensional modelling techniques regarding dimension tables are given in Appendix B.2.

3.4.1 Structure

The structure of dimension tables consists of a primary key column. The primary key is used as a foreign key for fact tables. A fact table row, will have a foreign key to the dimension table row for which the context was true, thereby enabling a descriptive context to a measurement from the fact table.

Dimension tables are denormalised and includes verbose text strings. Indicators and flags like enumerations and boolean values as well as codes can also be included as columns in dimension tables, these do not belong in fact tables, however giving verbose descriptions to these indicators, flags and codes are preferred. The attributes of a dimension table row is used as labels, constrains, i.e. filtering and grouping, hence giving the attributes proper verbose descriptions help the end-users.

Most dimensions can be filled with all/most possible dimension rows from the start, since these hold the

context that measurements can/will be in when they are inserted in a fact table. E.g. the date dimension can often be filled with rows for the next 10-20 years at the start of the usage of the DW/BI solution. More rows can always be added to accommodate the next 10 years. The same goes for e.g. device dimensions in a house context, since most of the devices are known from start, these can all be loaded into the dimension table, before they are actually used in the fact table. There are some situations where it is not advisable to put all possible values into a dimension table, e.g. when they will never be used or has not been used yet, then these rows can be added when the value is actually used in a fact table. This is decided by the ETL system.

Dimension tables are often named with a "Dim" prefix.

3.4.2 Flags, codes and indicators

Flags and indicator values in dimensions should be filled with proper text, instead of a boolean value or a yes/no value. E.g. if a weekday indicator is used in a date dimension, then the words "weekday" and "weekend" should be used instead of "true" and "false". This gives the end-users much better reporting capabilities. Flags, codes and indicators usually comes from the operational source systems, where they are often used. In addition to this null values should never be used in dimension tables, if it is necessary to have it, a more descriptive text should be used e.g. "not applicable" or "unknown".

3.4.3 Junk

The above flags and indicators might be used a lot in the operational source systems, and instead of creating a separate dimension table for each type of flag and indicator a junk dimension can be used. A junk dimension includes all the indicators and flags that are in the data from the source systems. Not all possible values should however be loaded, only the ones that are used. Junk dimensions can give significantly higher query performance than having these rows in different dimension tables. Each type of indicator is then made into a separate column in the dimension table. When indicators can be used together, the columns for these are also populated, otherwise the text "not applicable" or "unknown" is inserted into these columns. The junk dimension is never referred to as a junk dimension to end-users, for whom it is usually called a transaction indicator dimension or transaction profile dimension.

3.4.4 Keys

The primary key for a dimension table should never use the natural key from an operational source system. These keys might be used for several dimension rows, maybe even in the same dimension table. Since the natural keys are from the operational source systems, they might also occur in several systems, making it impossible to know which system the key was from. And since these natural keys are administered by the operational source systems they can easily be changed without the DW/BI system knowing, e.g. these keys might be reassigned to other entities in these operational source systems.

Instead a surrogate key should be used which is a simple continuous integer value. These keys are much more robust since they are simple and maintained by the DW/BI system. Using these keys also makes the

foreign keys much more simple in the fact tables as well, which makes a difference in the amount of space needed for the fact tables, which is the biggest consumer of space in a dimensional model.

The natural keys will most likely be used by the ETL system to assign the fact tables with foreign keys to the dimension table rows. Hence the natural keys will usually be saved either in the dimension table or inside metadata kept by the ETL system. The natural key is also referred to as an alternate key.

The date and time of day dimensions however usually use a slightly different key numbering scheme, since these dimensions are fully known from the start and almost never changed, these dimensions are given keys in the format *DDMMYYYY* for date and *PHHMMSS* for time, the *P* is a prefix number which is used to be able to keep the keys as integer values instead of string values.

Besides the natural key which is the key used from the operational source systems, surrogate keys which are continuous sequentially assigned integer values, durable keys are sometimes needed. These keys will never change even if the surrogate key is changed. This could e.g. be used to identify a specific device in a house. When changes are made to this device in the home control system, the data warehouse might give the device a new surrogate key, however to keep record of the specific device a durable key can be used to identify this exact device even though it has been given a new surrogate key. These keys are mostly used when there is a need to keep history records, and e.g. a dimension row is kept for the old device information and a new row is inserted for the new device information. The keys used in joins should be on surrogate keys, never on natural keys.

Using surrogate keys helps when multiple systems are integrated and it improves the performance. It makes it possible to uniquely identify not applicable dimension rows, and they make it possible to have several rows with the same natural key.

3.4.5 Drilling down and other dimensional operations

Drilling down using a dimension is one of the core features of a dimensional model. Drilling down means to get further into the measurement details. This is done by adding another dimension to a query, and thereby getting closer to the atomic measurement. When adding another dimension to the query, the dimension is added to the group by SQL statement for the query. The fact that drilling down to get closer to the atomic measurements is so simple is one of the most significant features of dimensional modelling. Note that this feature is done without any special predefined drill-down paths or programming, it works "out-of-the-box" with dimensional modelling. However to give the end-users a better experience these summarised queries should be pre-calculated to give a fast query response.

Drilling down means adding a row header from an attribute in a dimension, while drilling up means removing a row header from the query. Hence by drill down with all dimension attributes would give the most atomic data, at the most granular level, and hence show all details of the measurements in a fact table. While rolling all the way up to no row headers would result in the most summarised values of the measurements from the fact table.

Drilling down and rolling up aggregates the data within a single dimension attribute not for all the attributes in a dimension.

Other dimensional operations include dimensional reduction, pivoting, slicing and dicing. These opera-

tions usually operate with a whole dimension i.e. for all attributes.

When aggregates are rolled up to the highest level, this is a type of dimensionality reduction, i.e. multiple dimensions are reduced to a single base dimension.

When aggregates are rolled up to two dimensions the view is referred to as pivoting.

Slicing is selecting data for a defined dimension, while dicing is selecting the measurements that is wanted.

3.4.6 Hierarchies

Dimension tables should be denormalised such that hierarchies are flatten inside the same table. Dimensions can have several hierarchies built into the same dimension table, this could e.g. be a date dimension, which could have a week calendar, where the hierarchy would be year to week to date, as well as a month calendar, where the hierarchy would be year to month to date, these could even be put together to create a third hierarchy, which would be year to month to week to date. These different hierarchies should not be put into different dimension tables, they can coexist in the same dimension table. Descriptive names for the levels of the hierarchy is essential, because this helps end-users use the proper levels.

When a many-to-one relationships needs to be modelled as a hierarchy in a flatten denormalised dimension table, the levels are made into separate columns in the table. E.g. for a year to month to day hierarchy to be modelled, the year, the month and the day would all be separate columns in the dimension table. A row is then made for each day, which includes the month as a column and the year.

3.4.7 Multivalued

There exists two main methods to model many-to-many relationships e.g. for when a dimension should be multivalued. The two design methods are called positional and bridge table. The easiest is to model the multivalued attribute with several columns, one for each possibility, i.e. using a positional design. E.g. if it is possible to have devices in groups, then each group name would have a column. This design is not very scalable or flexible, since new columns would have to be added as more possibilities arise.

The bridge table design uses a bridge table which is a factless fact table, which has foreign keys to both sides of the dimensions. E.g. see the grouping example under factless fact table. This design is much more flexible and scalable, since new combinations can be made by adding new rows. It has the disadvantage though that, it is not possible to get null values for non-existing combinations, e.g. it is possible to find all devices without a specific group with the positional design, since this is just a query to find all the devices with the specified group that has a null value. However this is not directly possible when using a bridge table design, because only the groups that are used exists in the bridge table. Using a bridge table has the advantage that it is possible to add weight to each combination by adding another column. This could e.g. be used for device groups, where some groups have more devices than other groups, and hence the probability of an event occurring needs a weighing factor to be calculated when this measure is reviewed by device groups.

3.4.8 Date and time of day dimensions

One of the most significant dimensions is the date dimension, this dimension is used in almost every DW/BI solution, because it enables end-users to navigate through dates. The date dimension has most of the descriptive contexts for all days in the years that the data warehouse is used. It includes e.g. the month, the week number, the day of week, and maybe a holiday indicator to signify that a date is a holiday. As mentioned above with the keys, date is one of the dimensions which do not use a surrogate key, instead it uses an integer value which is a combination of the year, month and day. This is typically done because of the nature of the date dimension, because it is almost never changed, and thereby these durable keys can be used as the primary key instead of a surrogate key.

The number of rows for this dimension table is relatively small, considering a row for each day, e.g. for 25 years only $25 \cdot 365 = 9125$ rows are needed.

The date dimension should be made as a physical dimension table and not with a view, since many of the attributes cannot be easily derived using SQL, e.g. getting the week number for a specific date is not directly possible in all RDBMS systems. The date dimension should be used instead of using `datetime` types through SQL, even though these give the capabilities of filtering on specific dates. It is however much easier to join a number of date dimension rows given from a hierarchy than specifying these sets of dates in SQL. E.g. filtering on weekdays, in alternating years for a range between two years, is much more cumbersome to do in SQL than just joining on these from the date dimension.

Another date and time related dimension is the time of day dimension, which is used to give end-users the ability to drill down using the time as well as the date. This violates the above mentioned rule about flattening hierarchies, because the time of day is closely related to the date dimension, however the time of day dimension contains $60 \cdot 60 \cdot 24 = 86400$ rows and even more if milliseconds are used as the smallest grain. Appending 86400 rows for each day in the date dimension would not be advisable, even just for one year the number of rows to be loaded would be $365 \cdot 86400 = 31536000$. This is a waste of space, since the same information and capabilities are possible with much less using two separate dimension tables. The time of day dimension includes columns like hour, minute, second and it could also contain half hour and quarter hour.

3.4.9 Conformed dimensions

The best dimensions are the conformed dimensions, these dimensions can be used across multiple fact tables, and sometimes across data from different operational source systems. These dimensions can often be found by the use of the same name for columns in different dimensions and the same contents of rows. If the domain context is the same for these dimensions, they should be consolidated to a conformed dimension. The most basic conformed dimension is a date dimension, because this dimension can be used with nearly every fact table that has a reference to a date. Conformed dimensions can either exist physically as one dimension table in one database, or they can exist in different dimension tables, maybe in other databases, however to ensure that they can be used together the keys and contents of the dimension tables needs to be exactly the same. This makes it possible to use them as conformed dimensions. Con-

formed dimensions makes it possible to drill-across. These dimensions show the true integration between operational source systems that is provided by a DW/BI solution.

3.4.10 Drilling across

Drilling across is the ability to query different fact tables using the same dimensions. This is only possible if conformed dimensions are used, otherwise the query would result in different headings/labels for each fact table. It is important that this operation is not done using joins between the different fact tables, because this would cause performance problems and it might even cause miscalculations, instead a merge function should be used to give the result after receiving the result of separate queries to the fact tables. Most BI applications support these types of queries, and they are essential to give the end-users the ability to easily compare different facts from e.g. different operational source systems. This could e.g. be used to see the trends for the energy production vs. the energy consumption with a date dimension, if these facts are in separate fact tables and both use the date dimension.

3.4.11 Slowly changing dimensions

Dimension attributes in rows can often change, due to changes in the operational source systems. These changing attributes are called slowly changing dimension (SCD) attributes. There are several types of ways to handle this. The first five types are standalone/basic types, while several others are composites of these and builds upon these. These are called hybrid types.

3.4.11.1 Retain original

Type 0 is called retain original. When using this type, the attribute is never changed, even if it is changed in the operational source systems. Often this type is used together with the word original to describe the column name, because the original value is always kept. Durable keys always use type 0 for changes, since these keys are never changed after they have been set.

3.4.11.2 Overwrite

Type 1 is called overwrite. When using this type, the attribute value is changed to the newest and current value every time it is changed in the operational source system. This makes it possible for end-users always to know the value, because it corresponds to the current value in the operational source system. When overwriting attribute values BI applications should be reloaded because the old value cannot be used e.g. as a filter any more. This type has the disadvantage that history is not kept, because the value is just overwritten. This type should only be used when keeping history of the changes are not necessary.

3.4.11.3 Add new row

Type 2 is called add new row. When using this type, a new dimension row is inserted with new value(s) together with the old value for the unchanged values. When facts are loaded from this point on, they

will use the new primary key as the foreign key to the dimension row. This type may require three new attributes or columns to be added to the dimension table. The timestamps for when the new row should be used and when it is not to be used and a boolean which indicates whether it is the current row, that must be used when facts are loaded from this point on. This makes it possible to e.g. filter on changes in dimension row values, i.e. see if there are differences between before and after the change, since there are now two or more (if several changes have occurred) dimension rows. Hence it is possible to see differences by history based on these changes. This type is mostly used, since it keeps history of changes in a flexible way.

3.4.11.4 Add new attribute

Type 3 is called add new attribute. When using this type a new column is added to the dimension table, which contains the old value, and the new value is overwritten for the attribute, i.e. a type 1 change. This type is not very flexible if all the history of changes needs to be kept, since this would mean a new column for each change. And if only a subset or single rows have their attributes changed, this would cause a lot of not applicable or unknown fields. This type is not used very much, mostly because of the inflexible nature of it, especially for frequently changing attributes.

3.4.11.5 Add mini-dimension

Type 4 is called add mini-dimension. This type is mostly used for high frequency changes. When using this type a new "mini"-dimension table is added which contains the dimension rows for the frequently changed attributes and they are referred to by a separate surrogate primary key. This type is only useful when many changes are done at a time, and when they are done frequently. The fact table would then need to have foreign key to the normal dimension and a foreign key to the "mini"-dimension. This type does not keep any history of the changes, it is purely done for performance.

3.5 Star Schema

When data is represented in a dimensional model and deployed in a RDBMS system. And hence uses a relational table structure it is referred to as a star schema. It is called a star schema because of the relationship between fact tables and multiple dimension tables, which form a star-like structure. An example of the star-like structure is seen in Figure 3.5.1, where a Temperature fact table has multiple relations with dimensions. Each relation is a one-to-many relation, i.e. multiple fact table rows can be retrieved by querying for a single dimension row. The basis for the star schema is the fact tables and dimension tables discussed above.

The core of dimensional modelling is the simplicity and symmetry, which is the basis for its widespread use in data warehousing. It makes it easy to navigate to data, and it helps describe the context. The simplicity also helps the overall query performance because less joins across multiple tables are needed to get a full picture of the data.

An example of an SQL query to get data out of the star schema and thereby dimensional model is seen in Listing 3.5.1.

```

1  SELECT SUM([ MeasurementDelta ]) AS [ Delta ], [ DayName ]
2  FROM [ FactIndoorTemperature ]
3  JOIN [ DimDate ] ON [ DimDate ].[ DateKey ] = [ FactIndoorTemperature ].[
   DateKey ]
4  WHERE [ WeekNumberOfYearName ] = 'Week 27, 2014 '
5  GROUP BY [ DayName ]

```

Listing 3.5.1: SQL to query a star schema

Here the sum of the MeasurementDelta from the fact table is retrieved along with the dimension table context header DayName, for which the data is grouped by. The data is also filtered to only include the measurements for week 27 of 2014. This would result in two columns a Delta and a DayName column and a row for each day in week 27 for which there is data.

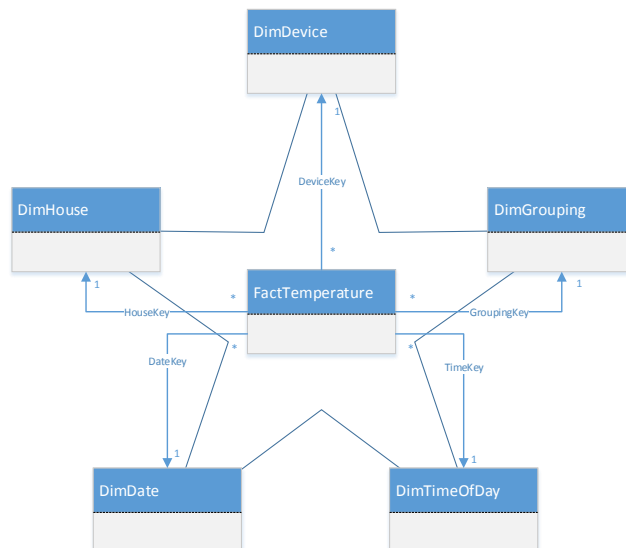


Figure 3.5.1: Example of a star-like structure for a Temperature fact table and five dimensions

Note how simple the data has been retrieved. If this data was only in an operational source system with a transactional record for the temperature measurements hidden away in multiple tables because of a 3NF model which is used by the operational application, multiple joins and obscure table names would properly have to be navigated. And defining the week of 27 of 2014 would involve much more than just a single where clause with the string "Week 27, 2014". – This is dimensional modelling.

The star schema and dimensional model is easily extendible because a row is just added to add new measurements or dimension labels or levels of a hierarchy. And since most BI applications just use the same query, no data needs to be reloaded or applications adjusted to accommodate these changes. In addition to this it is relatively simple to add new columns to accommodate new measurements and new dimension

columns, since these can be done using SQL.

An important note to the simplicity is to keep the model as simple as possible and not trying to optimise it against a specific report or report type or by trying to apply 3NF. The end-users will always have new different questions which they want to query upon, so optimising the model against one type of querying, could mean a decrease in performance for new types of queries.

To give the end-users the best possible end-result, measurements should be done at the most granular level, i.e. as atomic the data can be measured, the better. Since accumulated data can only be used to get to a certain level of detail, and thereby inhibit the ability to drill through the data.

Note that once the measurements have been made it is always possible to roll it up to a lower detail level, but it is not possible to drill down to a higher detail level.

3.6 OLAP Cubes

When data is represented in a dimensional model and deployed in a multidimensional database system it is referred to as an OLAP cube. It is called online analytical processing, because of the implementation of the cube, which requires index and calculation preprocessing to be done when data is loaded and is cached in the system. This preprocessing requires more time when the data is loaded, which depending on the amount of data, can potentially mean that data is a little slower updated, i.e. because of the additional processing time. Load of new data might not be done as often as with a regular star schema. However because of the preprocessing it delivers exceptionally better query performance for the end-users than a regular star schema which is dependent on the performance from the RDBMS system.

The number of dimensions and measures does not have to be equal as the name "cube" implies, and a cube can easily consist of several more dimensions than a three dimensional cube. The cube name just makes it more tangible for end-users.

Data cubes have been around for several years, it is actually a simplification of a statistical cross-tabulation. The inherent nature of OLAP cubes makes them better at doing more complex calculations as well as manipulating with the data in each row.

One of the biggest advantages to OLAP products is the fact that they can handle parent-child relationships in a much better way than e.g. using a star schema, where this relationship needs more modelling. By using an OLAP cube it is sufficient to have a single value/key, which can roll up the parent-child hierarchy. Another advantage to OLAP cubes is that since precalculations are done, it is much faster to build a solution which can analyse events that "did not happen". Since the cube calculates the entire facts and dimensions, it is possible to select all the dimension and fact rows where an event did not happen, i.e. where the opposite context of the specified dimension rows were true.

A multidimensional system like an OLAP cube presents data in a multidimensional way, i.e. the cube, which makes them a good tool when modelling a multidimensional model, it lets end-users easily slice and dice the cube of data.

By defining the relationships between tables explicitly and by precalculating summaries they outperform all RDBMS systems. And they can be used for much more analytical processing because they have many

built-in calculation capabilities, which does not exist for SQL and RDBMS systems.

Even though the multidimensional model consumed by the cube can be made directly from operational source systems, the OLAP system prefers that the source is already a multidimensional model e.g. a star schema, since it does not have to handle any ETL processing, which is done for the star schema. This also makes the processing of the OLAP cube much faster. It should however be noted that OLAP cubes need to be reloaded and processed when new dimension rows are added or changed, i.e. when slowly changing dimensions are used in a multidimensional design.

Star schema and OLAP cubes can be used together to create a better performing DW/BI solution. The star schema can thereby be used as the foundation for the OLAP cube. The content of a cube is often equal to the content of a star schema, it often just includes pre-calculated summary data for faster querying.

An advantage with OLAP cubes is added security which in most OLAP systems can be defined by the level of detail in which end-users have access to data. This is not possible for a regular star schemas in RDBMS systems, where security can usually only be made on tables and databases.

OLAP cubes has the disadvantage that porting it from one vendor to another, requires much more effort than porting a star schema, since the star schema can be implemented in any RDBMS system.

OLAP cubes are not queried through SQL like a star schema, instead MDX and sometimes XMLA is used. MDX is discussed further below. XMLA is an XML based query language, which can e.g. be used to send processing requests to an OLAP system.

The preprocessing of the cube can include the creation of aggregate fact tables, these fact tables include summarised values for the facts. This makes query response much quicker, because the aggregates are already calculated. These summaries are then indexed to include the foreign keys for the dimensions which they are most likely to be queried from. Hence a cube consists of a multidimensional model and the aggregates for (nearly) all possible combinations.

OLAP systems which internally are based on relational models are referred to as ROLAP, while systems which are internally based on multidimensional models are referred to as MOLAP. Note that it is not dependent on whether a star schema is used or not, it is how the data is represented internally in the system.

3.6.1 MDX query language

The Multidimensional Expression Language (MDX), resembles the SQL query language, however MDX has more analytical capabilities, through the way it queries the cube.

The basis for MDX queries is the SELECT statement which resembles the the SELECT statement from SQL. In SQL the SELECT statement returns a number of rows and columns. In MDX the SELECT statement returns a dimensional model, which can e.g. be in two dimensions and thereby be returned as rows and columns.

A basic MDX SELECT statement is given in Listing 3.6.1.

```
1 SELECT FROM [Name of Cube]
```

Listing 3.6.1: Example of a basic MDX SELECT query

When the cube is processing a SELECT statement it automatically defines a tuple with all dimensions including a "Measures" dimension which is the measures or facts. When nothing is specified for this tuple, the default values for the dimensions are used. The values for the dimensions are called members. Members are the "children" of the attribute hierarchies.

A cube named "DW Cube", which has three dimensions date, time and device, will be used in the following examples. Including these three dimensions, the special "Measures" dimension is also included in the tuples used to query the cube.

The SELECT statement in Listing 3.6.1 would result in a query where the following tuple would be used:

```
([Date].[Date Hierarchy].DefaultMember, [Time].[Time Hierarchy].DefaultMember,
 [Device].[Device Hierarchy].DefaultMember, [Measures].DefaultMember)
```

The [Date Hierarchy] is the default attribute hierarchy for the date dimension. The DefaultMember is the default member of that hierarchy. Most default members for dimensions is the [(All)] member, which includes data for all the members in the dimension, i.e. all data. The default member of the measure is usually the first measure that was defined. For this example the default measure will be "Average Temperature". The default members can be explicitly defined in the cube. The above tuple is hence translated to the MDX statement given in Listing 3.6.2.

```
1 SELECT {[Date].[Date Hierarchy].[All]}, [Time].[Time Hierarchy].[All], [Device].[Device Hierarchy].[All], [Measures].[Average Temperature]} ON COLUMNS FROM [DW Cube]
```

Listing 3.6.2: Example of a basic MDX SELECT query with all tuple values defined

A set of dimensions is defined by the following syntax:

```
{[Dimension 1], [Dimension 2]}
```

The ON COLUMNS from the statement in Listing 3.6.2 means that the resulting set should be placed on the columns axis. The Listing also shows how to define the query dimensions in a SELECT statement. An example without the whole tuple is given in Listing 3.6.3.

```
1 SELECT {[Date].[Date Hierarchy].[Year]} ON COLUMNS FROM [DW Cube]
```

Listing 3.6.3: Example of SELECT query

This would result in the following tuple being sent to the cube:

```
([Date].[Date Hierarchy].[Year].Members, [Time].[Time Hierarchy].[All],
 [Device].[Device Hierarchy].[All], [Measures].[Average Temperature])
```

As seen in the tuple the Members is not needed when querying the cube. The Members will result in showing the explicit members of the year, e.g. 2014, 2015 etc. on the columns and the average temperature will be shown in the first row. Another example query is given in Listing 3.6.4.

```

1 SELECT {[ Device ].[ Device Hierarchy ].[ Sensors ]} ON COLUMNS, {[ Date ].[
  Date Hierarchy ].[ Year ]} ON ROWS FROM [DW Cube]

```

Listing 3.6.4: Example of MDX query with multiple members on columns and rows

This would result in the following tuples:

```

([Date].[Date Hierarchy].[Year].&[2014], [Time].[Time Hierarchy].[All],
 [Device].[Device Hierarchy].[Sensor].Members, [Measures].[Average Temperature])
([Date].[Date Hierarchy].[Year].&[2015], [Time].[Time Hierarchy].[All],
 [Device].[Device Hierarchy].[Sensor].Members, [Measures].[Average Temperature])

```

And etc. for each row. The result will be the average temperature for each year on the rows and in the columns the different sensor devices will be, e.g. first floor temperature sensor, ground floor temperature sensor. Notice the "&", which is used to uniquely identify a member.

A "slicer" can be added to a query e.g. to get different measures. The slicer is a WHERE clause in the SELECT statement. An example is given in Listing 3.6.5.

```

1 SELECT {[ Device ].[ Device Hierarchy ].[ Sensors ]} ON COLUMNS, {[ Date ].[
  Date Hierarchy ].[ Year ]} ON ROWS FROM [DW Cube]
2 WHERE ([ Measures ].[ Average Energy Consumption ])

```

Listing 3.6.5: Example of MDX with a WHERE slicer

This would give the same result as in the previous example but with the average energy consumption shown. The slicer can also be used to constrain the query, e.g. if only a specific device is needed then the query could look like the one given in Listing 3.6.6.

```

1 SELECT {[ Device ].[ Device Hierarchy ].[ Sensors ]} ON COLUMNS, {[ Date ].[
  Date Hierarchy ].[ Year ]} ON ROWS FROM [DW Cube]
2 WHERE ([ Measures ].[ Average Energy Consumption ], [ Device ].[ Device
  Hierarchy ].[ Devices ].&[HVAC Meter ])

```

Listing 3.6.6: Example of WHERE slicer for constrains

This will result in the same result as above but it will only show the measure for the "HVAC Meter" device. And hence it will only have one column.

When using e.g. the "Average Energy Consumption" measure some devices, e.g. the temperature sensors will result in (*null*) values in the result view. To get the proper values and discarding these (*null*) values. The NON EMPTY clause can be used in the SELECT statement as given in Listing 3.6.7.

```

1 SELECT NON EMPTY {[ Device ].[ Device Hierarchy ].[ Sensors ]} ON COLUMNS,
  {[ Date ].[ Date Hierarchy ].[ Year ]} ON ROWS FROM [DW Cube] WHERE ([
  Measures ].[ Average Energy Consumption ])

```

Listing 3.6.7: Example of NON EMPTY query

To get more dimensions out in the result, cross join of the dimensions can be used for a set of dimensions. The set operator "*" (asterisk) can be used as given in Listing 3.6.8.

```

1 SELECT NON EMPTY {[ Device ].[ Device Hierarchy ].[ Sensors ]} ON COLUMNS,
   ([ Date ].[ Date Hierarchy ].[ Year ] * [ Time ].[ Time Hierarchy ].[ Hour ])
   ON ROWS FROM [DW Cube] WHERE ([ Measures ].[ Average Energy
   Consumption ])

```

Listing 3.6.8: Example of cross join of dimensions

This query will result in a three dimensional output, where the dimensions are the "Year" and "Hour" as well as the "Devices". Since the cross join was done on the rows axis, the result will be flattened to fit into the rows, this e.g. results in rows of "2014 - Hour 1", "2014 - Hour 2", and etc. for each year.

When defining sets the WITH statement can be used to make temporary set "variables". An example is given in Listing 3.6.9.

```

1 WITH SET [Important Devices] AS
2 {[ Device ].[ Device Hierarchy ].[ Devices ].&[HVAC Meter] , [ Device ].[
   Device Hierarchy ].[ Devices ].&[ Lights Meter ]}
3 SELECT [Important Devices] ON COLUMNS
4 FROM [DW Cube]

```

Listing 3.6.9: Example of WITH statement for a set

This query would then use the set defined in the WITH statement. The WITH statement can also be used to define calculations as given in Listing 3.6.10.

```

1 WITH MEMBER [Measures].[Average Dimmable Light Level] AS
2 [Measures].[Dimmable Light Level Sum] / [Measures].[Dimmable Light
   Level Count]
3 SELECT [Measures].[Average Dimmable Light Level] ON COLUMNS
4 FROM [DW Cube]

```

Listing 3.6.10: Example of WITH statement for a calculation

Note that the measures can also be defined as query dimensions on columns as well as rows.

The following set operations are valid in MDX:

- - (except) for set difference
- * (cross join) for Cartesian product
- + (union) for set union
- : (range) for ranges, e.g. [Year].&[2014] : [Year].&[2016], which will give the set containing the members 2014, 2015 and 2016.

Other SQL-like statements are also valid like comments, arithmetic operations, string concatenation and logical expressions.

3.7 Design Process

The design process and methodology of dimensional modelling recommended by [Kimball and Ross, 2013] is a four-step design process.

The process starts by considering the business processes that needs to be in the dimensional model. Next the grain is clearly defined. Next the dimensions are identified. And lastly the facts are identified.

1. Understanding the business processes and what performance metrics can be extracted from these is key.
2. Clearly defining the grain helps to establish a firm foundation for the dimensions and facts. The grain should be defined by business terms, e.g. one row per sales transaction line. This makes it general enough not to have premature commitments about the dimensions. However it might be easier to define the grain clearly by dimensions after the design has been made complete.
3. The dimensions are identified by looking at the context for the business processes and the steps involved when the metrics are measured. When the grain is clearly defined, the dimensions are usually the answers to the "wh"-questions, e.g. what, when, who, where etc., these should be clear otherwise the grain must be redefined.
4. The facts are identified by looking at the metrics needed and the metrics available from the business processes. The metrics might not be available because they are not measured in the operational source systems at the desired grain, but maybe they might exist at another grain, which means that the grain must be redefined.

It is should be clear from the above that the design process can be an iterative one until the right grain is defined from the business processes.

3.8 Data Warehouse and Business Intelligence components

The components of a DW/BI solution is frequently composed of four components: the access to a number of operational source systems, the ETL system, the data warehouse, also called the presentation area, and the BI applications.

- The operational source systems are the systems which contains the, often, transactional measurement records from systems used for operational purposes. This could e.g. be the home control systems. The DW/BI solution needs access to these.
- The ETL system is a component which handles the data transition from the operational source systems to the data warehouse.
 - The extract process handles the extraction of data from the operational source systems, this includes extracting it through various data access layers. This could e.g. be by database operations, or by web service calls.
 - The transformation process handles the cleansing of data, this involves e.g. handling of duplicates and conflicts, and conforming the data to a certain data format or standard, so that the data is comparable.
 - Lastly the data is loaded into the data warehouse, where e.g. surrogate keys are given to dimensions. Natural keys are usually the keys used to identify a row in the operational source system. Using this key makes it possible for the ETL to find and lookup the right dimension for facts extracted from the operational source system.

- The data warehouse is also called the presentation area, because it is where the data is presented to the BI applications, and sometimes directly to the end-users. This is where the dimension tables and fact tables are stored. This can either be in a star schema and/or a cube, which also contains pre-calculated measures. The data in the data warehouse should always contain the most atomic data possible, i.e. at the most granular level possible, so end-users have the ability to drill down to the actual measurements and not have to go to an operational source system to get this information.
- The last component in a DW/BI solution is the BI application, which consumes the data warehouse and makes it possible for end-users to query and make calculations on the measures from the fact tables. The querying and filtering of measures is done by using the descriptive context dimensions.

3.9 Summary

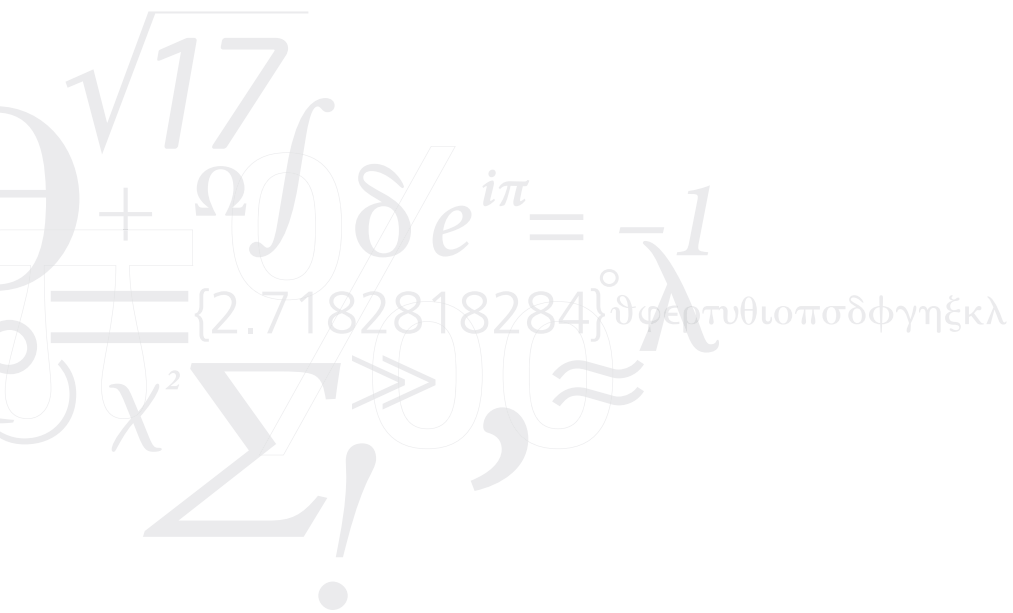
Third normal form (3NF) is considered best practice for database designs. However for BI and data warehousing, dimensional modelling is considered best practice, which is a denormalised data structure. The reason why dimensional modelling is denormalised, is because it increases query performance and it makes the model much easier to understand for end-users. A dimensional model in a data warehouse can be based on a 3NF model from operational source systems, which often are modelled using 3NF. Dimensional modelling consists of two types of tables, facts and dimensions. Fact tables contain measurements (also called metrics and facts) and foreign keys to dimension table rows. Dimension tables contain a primary key and descriptive context with textual strings, which often describe the "wh"-questions like "when", "where" and etc. of a fact measurement. Fact and dimension tables form a cube, where the inner points of the cube are facts and the edges are dimensions. This model makes data more tangible for end-users.

Additivity for facts categorises whether these can be summed across all dimensions, called fully additive, a subset of dimensions, called semi-additive, or no dimensions, called non-additive. The grain of a fact table describes the level of detail of each row in the table. Defining the grain is important, because the grain should never be mixed in the same fact table. Most fact tables are transactional, because of the transactional data from the operational source systems.

Dimension tables use surrogate keys e.g. for the primary key, instead of the natural key from operational source system. This makes it possible to the same natural key for multiple rows in a dimension table. Durable keys are used when history of changed dimension rows is needed. Using dimensions it is possible to do dimensional operations on fact tables, like drilling down, which is done by adding a dimension row header. Rolling up summarised data is done by removing a dimension row header. Slice and dice is done by defining dimensions and measures. Dimension tables often represent hierarchical structure inside the dimension table. Since dimensions are often dependent on operational source systems, slowly changing dimension attributes are commonly used, there are several ways to handle these changes.

A dimensional model deployed in a RDBMS system, is referred to as a star schema. A dimensional model deployed in a multidimensional system, is referred to as an OLAP cube. OLAP cubes use MDX queries. A DW/BI solution consists of access to a number of operational source systems, an ETL system, a data warehouse and BI applications.

This page intentionally left blank



Chapter 4

Goals

"Simplicity is prerequisite for reliability"
—Edsger W. Dijkstra, 1975

Contents

4.1	Stakeholders	43
4.2	Project Goals	44
4.3	Non-functional requirements	45

This chapter defines the goals and research questions for the project.

4.1 Stakeholders

To be able to identify the right goals for this project, it is important to consider the stakeholders and their respective stakes, in the context presented in the preceding chapters.

The identified main stakeholders for the Data Warehouse project for the Embrace house are: the occupants of the house, a tablet application developer, the students from the Embrace project, the researchers, the sponsors and the public visiting the Embrace house. As described in the context the ten contests of the Solar Decathlon Europe competition will evaluate how well a house will be to live in for the people (occupants) of the house. Hence the focus on the stakeholders is on the competition. The stakes for the respective stakeholders is given in Table 4.1.1.

Stakeholder	Stake	Importance
Occupants (External)	Get relevant data by easy to understand business intelligence reports (insight reports)	High
Tablet application developer (Internal)	Get relevant data and ability to show graphs from these	High
Students (Internal)	Get relevant data for theses and analyse the performance for specific parts of the house	High

Researchers (Internal and External)	Get relevant data to analyse the overall performance of the house and ability to drill through to get specific data	Medium
Sponsors (External)	Be able to showcase the capabilities of a DW/BI solution which can show the distribution of energy consumption	Medium
Public (External)	Be able to see the innovation in using data warehousing and business intelligence in intelligent houses	Low

Table 4.1.1: Stakeholders and their respective stakes

4.2 Project Goals

From the stakeholders and the context the following goals are identified.

The main goal for the EMBRACE project is to save energy.

This goal can be further refined to the following:

For the occupants of the EMBRACE house to consume less energy and gain awareness of their energy consumption.

The gain awareness goal can be refined further:

To provide and gain insights into the house and thereby getting a complete picture of the house functioning and usage.

This should be done by making sure that the data is valid and the ability to provide analysis of the data.

The analyse data goal can be further refined to these specific goals:

Help occupants to find relations using the data, give them the ability to drill through data and offer ways to slice and dice the data. The analysis should be done using a simple interface for reporting which is intuitive to use.

The non-project goal is to experiment with data warehousing technologies like ETL and OLAP cubes.

Figure 4.2.1 shows the hierarchical order of the project goals.

4.2.1 Research Questions

From the above goals the following main problems will be investigated in this thesis.

What is data warehousing? And how does it relate to Business Intelligence? What is possible with a data warehouse? And is it possible to use it in an intelligent house context, instead of a business context?

Which components are needed for a business intelligence solution using a data warehouse and how do the components interact? How can data be presented in an efficient and intuitive way for end-users? What is the best way to model a data warehouse? What is needed for an efficient ETL system? How can business intelligence reporting be used to gain knowledge about the performance of a house?



Figure 4.2.1: Project goals

4.2.2 Principal Research Question

The project goals and research questions propose the principal research question which this project needs to address which is how to give the occupants and other subsystems of an EMBRACE house the insights of the functioning of the house, by the measurements done by the control systems.

4.3 Non-functional requirements

For the data warehouse project to succeed a number of non-functional requirements must be considered together with the goals.

The performance must be adequate, it does not help the end-users to wait too long for queries to execute. Hence the response times and refresh times should not be longer than one second. Beside the query response time, the data warehouse will process new data, the processing time should be minimised as much as possible to serve newer data to the end-users as fast as possible. Before the data warehouse can process new data it must extract, transform and load data into the data warehouse, the time used on the first loading should not be longer than one hour depending on the amount of data in the sources. Because this is dependent on the amount of data, a processing time for each row should be defined. This should be maximum 100 ms per row. Subsequent ETL operations should be less than 15 minutes. Again for each rows this should be maximum 100 ms per row.

If it takes approximately an hour to process the data for the first processing, a requirement to have a fall-

back data warehouse is needed. This fall-back should be used during maintenance, where a first-run processing is only needed.

The system must be extendible to be able to accommodate new data sources as these arrive. This also means that maintainability of the data warehouse system should be easy.

The interface and data presented to the end-users must be intuitive and easy to use, hence the usability of the system must adhere to this.

The data should be secured in the cloud and only available to authenticated users. Authorisation can also be considered for the access, since the data warehouse is able to serve several EMBRACE houses.

The integrity and validity of the data must be adequate enough to give a complete picture, this means that data which is invalid in terms of e.g. being above a set maximum or below a set minimum, should be discarded. It also means that data which cannot be mapped properly should be discarded. All the discarded data should still be logged in order to do auditing at a later stage.

The data must be consistent, meaning labels and descriptions should be the same for the same data. I.e. if two labels are same, then the data should be the same.

The storage needed for the data warehouse will grow as more and more data is loaded into it. Hence the storage capacity should be extendible. The initial storage capacity for the data warehouse should not exceed 250 GB.

The amount of transactions the data warehouse should be able to handle should accommodate the usage from occupants, researchers, students and the public, if they use the data warehouse for 25 queries per hour on average and the amount of people/devices using it on average every hour is about 250, then the system should handle 6250 queries per hour on average. However this should also be scalable to accommodate more queries and more people/devices.

The availability should be as high as possible, however since the service is an added service for the occupants of the EMBRACE house, it should be enough with 95% availability, meaning $365.242 \text{ days} \cdot (1 - 0.95) = 18.3$ days of unavailability a year.

Recoverability of the data warehouse should be established by ensuring the data warehouse can be regenerated from the source data at any point in time, and as long as the source data is intact, the data warehouse should be able to recover. Hence backup of the data warehouse is not required, it is however required for the source data. The time estimates for the regeneration of the data warehouse should adhere to the initial ETL loading times described above, and if possible the fall-back data warehouse should be available, during the first load.

Compatibility is limited to a Windows environment with the .NET Framework, since C# is being used by the other components in the system. To be able to communicate with other devices, the system will have to use REST web services and the HTTP application layer. To communicate with other components of the system the Kademia P2P Publish/Subscribe network will be used.

Chapter 5

Analysis

"The purpose of computing is insight, not numbers"
—Richard Hamming, 1962

Contents

5.1	Project goals from given context	47
5.2	Sub-problems	48
5.3	Solutions for sub-problems	50
5.4	The proposed solution	59
5.5	Business Intelligence Reporting	59
5.6	Operational Source Systems	61
5.7	Requirements specification	66

This chapter analyses the problem space, the different operational source systems and the contents of the business intelligence reports. This leads to a requirement specification at the end of the chapter.

5.1 Project goals from given context

The project goals are defined from the top-most goal which is inherited from the EMBRACE project, which is to save energy. This goal is part of the three main concepts of EMBRACE, which are Save, Smart and Share.

This thesis will try to focus on the defined goals, which are highly influenced by the Solar Decathlon Europe 2014 competition. However since the main goals for the competition is the ten contests which tries to evaluate the best house on different parameters, these goals are also valid for a house built for occupants who wants to live in this house. Because of this correlation between competition goals and goals for occupants, the focus of this thesis will be on the competition goals.

As defined in the project goals, the top-most goal can be combined with the Smart and Share concepts as well which give rise to this Data Warehouse project. Because saving energy in a smart way, which makes

it possible to share the saved energy with other houses, can be made possible by making the occupants more aware of their energy consumption. Awareness about consumption patterns might contribute to less energy consumption, which is another goal for this project.

Drilling further down these energy consumption goals only gets this far. Going further into how this can be achieved however results in more goals for this project:

Gaining awareness of the energy consumption can hence be further decomposed into providing insights into the house functioning and getting a complete picture of the house usage.

The insights should be presented and manipulated intuitively and should be easy to use.

The data which is the foundation of these insights should be valid.

Taking these goals further into solution-oriented goals the following goals emerge:

Ability to drill through and analyse the data. Ability to find relations using data.

These goals can be realised by making it possible to slice and dice the data cubes into specific areas of interest.

5.2 Sub-problems

The idea of insights from the principal research question can be further decomposed into the following sub-problems. The sub-problems can also be viewed as use cases for the project, which is shown in figure 5.2.1.

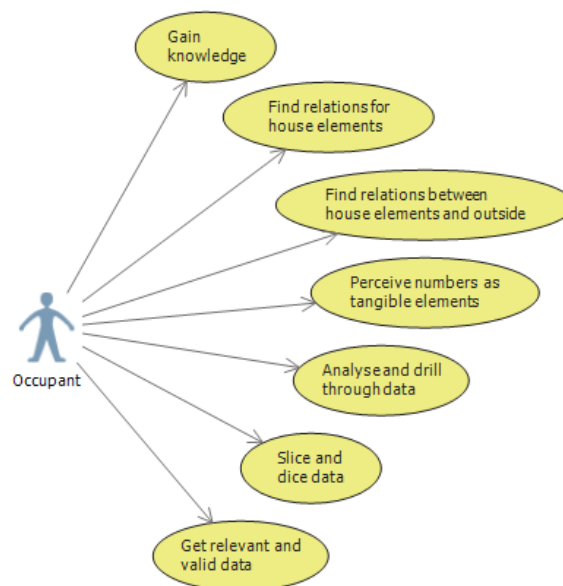


Figure 5.2.1: Use cases from the goals

1. Gain knowledge about the functioning of the house

The idea of gaining knowledge for the functioning of the house can be further decomposed into the

following components:

- (a) Gain knowledge about energy consumption and the context in which it is consumed
- (b) Gain knowledge about energy production and the context in which it is produced
- (c) Gain knowledge about how the devices in the house are used and how long they are in different states
- (d) Gain knowledge about the temperature, humidity and other indoor comfort conditions which might affect the usage of the house and the context for these

2. See how elements relate

The idea of seeing how elements relate can be further decomposed into:

- (a) Seeing differences in curves in a graph for elements in a house
- (b) Seeing similarities in curves in a graph for elements in a house
- (c) Ability to compare graphs to further analyse relationships

3. See how elements of the house and outside conditions relate

The idea of seeing how elements of the house and outside conditions relate can be further decomposed into:

- (a) Ability to see outside conditions like temperature and cloud cover
- (b) Ability to compare graphs from outside conditions with indoor comfort conditions

4. Perceive the numbers as tangible elements

The idea of perceiving numbers as tangible elements can be further decomposed into:

- (a) Ability to easily add graphs to be generated in reports, instead of showing raw numbers
- (b) Ability to summarise as many numbers as possible, so no additional calculations are needed to get the full picture
- (c) Ability to change formatting of numbers so they seem familiar to the occupants
- (d) Ensure understandability of the data model, e.g. by using a tangible object like a cube.

5. Drilling through data to analyse it

The idea of drilling through data to analyse it can be further decomposed into:

- (a) Ability to summarise data from a hierarchy level above the current summarised view (Roll up)
- (b) Ability to summarise data from a hierarchy level below the current summarised view (Drill down)
- (c) Ability to summarise data from different measurements (Drill across)
- (d) Ability to drill all the way down to the raw data (Drill through)

6. Slice and dice data cubes

The idea of slicing and dicing data cubes can be further decomposed into:

- (a) Ability to specify the view for specific facts
- (b) Ability to specify the view for specific dimension to see the facts
- (c) Ability to give numbers context by means of dimensions

7. Get relevant and valid data

The idea of getting relevant and valid data can be further decomposed into:

- (a) Ability to calculate with known metrics

(b) Having valid data

5.3 Solutions for sub-problems

The above identified sub-problems can be solved in several ways.

- The first solution is to use the operational source systems and modifying or extending them to accommodate the capabilities necessary to get an overview of the data.
- The second solution is to use an ordinary 3NF relational model for the data. The data is already stored in 3NF relational models in the operational source systems, so these can even be used, without the need to "relocate" the data.
- The third solution is to use a star schema model for the data. This requires that the data is "relocated" from the operational source systems, where the data is stored in a 3NF relational model.
- The fourth solution is to use a multidimensional system with an OLAP cube to present the data. This can even be done using either the raw data from the operational source systems, the 3NF relational model or the star schema model.

5.3.1 Extending operational source systems

Extending the operational source systems is one solution to the given sub-problems.

5.3.1.1 Pros

Since the data is already located in these systems, it makes sense to let these systems generate overviews. An advantage is that no overhead is produced e.g. by having additional systems transferring data. And hence the overviews will also be up-to-date with the current data in these systems.

There is no need to increase storage capacity either, since all the data is already in the databases.

5.3.1.2 Cons

This solution would however require the systems to handle the transactional data in a very different way, than they already do. Since most operational source systems handle transactional data one by one, which they are designed for, and can handle quite well. This does not work well for analysis though, so these systems would have to be extended to handle and calculate hundreds of thousands, even millions of records to get summarised data.

In addition to handle the vast amounts of records, these systems are not integrated, which means that the overviews that can be generated from these systems, will not give the complete picture, it will only show the summarised data for the particular systems. These integrations could also be done, which would however cost more in overall development.

Access rights to the data can be limited to the same authorisation levels as the existing systems use. This is both good and bad. Since most systems use very restricted access rights it will be impossible for unauthorised people to get any data, which is good. But this also means that people who only wants the summarised

data will need to get access to the systems, which also includes the raw data, which is bad.

Another problem with this solution is the fact that these systems are typically business critical in the sense that making large queries all the time to these databases might interfere in the operational use of these systems. This also means scalability to accommodate an increase in users or queries made by users of the DW/BI system will have an adverse effect on the operational use of the systems, and hence scalability is almost non-existent for this solution.

Also just extending these systems to handle analysis does not make them very extendible when new data sources or types of queries are needed.

The query performance of this solution is not good, because these systems are not designed to analyse the data, they are designed to handle normal operation, which is most likely to handle single transaction rows. With this solution data quality and validity must be defined for each system, which can make comparison of raw and summarised data between systems impossible or worse, incorrect, if users think the data can be compared.

5.3.1.3 Sub-problems covered

The sub-problems covered with this solution includes (1) and to some degree (4) and (5).

- Sub-problem 1 is covered because the solution makes it possible to gain knowledge about the functioning of the house, by being able to summarise the data and thereby give indications to what can be done to e.g. reduce energy consumption.
- Sub-problems 2 and 3 are not covered because for each of the operational systems only the data for this particular system is available to be analysed. And hence comparisons cannot be made to e.g. see relations.
- Sub-problem 4 is covered to some degree because it could include the functionality to show graphs and format the numbers to seem familiar. However many calculations would probably have to be done after the reports/summarised data is generated, because of the usage by occupants wanting answers to very different questions, hence these cannot be predefined. This can make the system very tedious and cumbersome to use and will hence not give the full picture required.
- Sub-problem 5 is covered to some degree because the raw data is available in these systems, making it possible to retrieve these. But this retrieval might not be as intuitive, if this is not done with the same filters, which are used to drill down to the current summarised views.
- Sub-problem 6 is not covered because using the transactional data means that it is not modelled as a cube, and hence dimensions and facts cannot be defined to slice and dice the data to the needs of the analysis.
- Sub-problem 7 is not covered because the data validity needs to be defined for each operational source system. And hence data validity cannot be ensured.

From the above it is seen that this solution only covers very few of the sub-problems. Hence this solution is not the proposed solution for this project.

5.3.2 3NF relational model

Using a 3NF relational model is another solution to the given sub-problems. This solution can be split into two solutions, one where the databases of the operational source systems are used and another where the data is replicated and relocated to another database.

5.3.2.1 Pros

The pros for a 3NF solution using the databases of the operational source systems are basically the same as the above solution. However since the operational source systems are not used, but only the data model from them, it allows for the use of better analysis and database tools which can be used for the data model. This can e.g. make it possible for end-users to query the specific data that they need. The data is also always up to date, with the operational source systems.

The most significant pros for a 3NF solution which replicates the data into another 3NF (standalone) data model is that when requesting large queries, the operational source systems will still be able to perform as they usually do, since the data is queried elsewhere. Again for this solution the fact that the operational source systems are not used, means that better analysis and database tools can be used.

5.3.2.2 Cons

The cons for a 3NF solution using the databases of the operational source systems are also basically the same as the above solution. However there will also be security and usability issues with this solution. Most RDBMS systems give access based on databases and tables, this means that when an end-user is given access to the database, all data is available, even the raw data, i.e. not just summarised data. The usability of this solution is very low, since end-users must learn the underlying data model/structure, which for a highly optimised operational source system will include dozens, maybe hundreds of tables and links between these. This will overwhelm most end-users, especially non-technical users. The end-users will not have any tangible feel for the tables or the numbers, since they are in 3NF.

Another problem is that the databases for different operational source systems will be in different databases, which can make it impossible to get a complete picture. A significant problem with this solution is that the queries are handled by the same RDBMS systems, which also handle the operational source systems usage of the data, which can lead to decreased performance for these. Note that these systems are often business critical so decreased performance is not good. Another issue with the query performance is when calculating summaries, which still needs to be calculated based on single transaction rows, for each query, which means that query performance will be low, especially if multiple tables are joined.

Integration between operational source systems is non-existent, which means end-users will have to manually load the data into an analysis tool to be able to drill across. This also means that data quality and validity is defined for each system, which means the numbers might not be comparable.

For the 3NF solution which replicates data, all of the above cons also apply, however since the data is physically located in other tables/databases, the querying does not affect the performance of the operational

source systems. However since the data needs to be replicated, the data might not always be up-to-date with the latest data from the operational source systems. This solution also requires that data storage capacity needs to be increased.

This solution also requires separate components which handle the data transition, i.e. ETL components to be implemented.

5.3.2.3 Sub-problems covered

The sub-problems covered with these solutions includes (1) and to some degree (2), (4) and (5).

- Sub-problem 1 is mostly covered since it should be possible to get some measured numbers out of the 3NF data models. However the context for the numbers, might not be easily retrieved through these models, since this data is probably hidden away in the data model, because of the application layers.
- Sub-problem 2 can be covered by the use of proper analysis and database tools, however the data might need to be loaded into these tools manually, if the analysis tools are independent of the data source. These tools will be able to show visualisations of the data, however to get the most out of the data, it may need to be transformed or calculated before they can be used directly by the visualisation tools. Hence the data model needs some work before it can be properly used by the analysis tools.
- Sub-problem 3 is not covered by this solution because this data will most likely come from two operational source systems and getting a complete picture will be almost impossible.
- Sub-problem 4 is somewhat covered because it should be possible to add graphs and summarise some of the metrics in the 3NF data model. This is however likely to be done by the analysis tools and not by the data model, which can affect query performance a lot, since it is the analysis tool that needs to do the calculations. The data model will most likely not be easy to understand for end-users, which will make the usage of these very tedious and cumbersome, since most of the data needs to be processed manually to fit the data to the analysis tools. Also if end-users do not clearly understand the data model, they will most likely have a very hard time figuring out e.g. which tables to get specific metrics from.
- Sub-problem 5 is nearly covered since it should be possible to get summarised data, and the raw data is easily available. However it is not possible to drill across, because the data might not be comparable and the metrics will most likely come from different operational source systems, making it even harder to drill across. Also the data, directly from the tables, might not be in hierarchical order, which means that the data is not easily rolled up for the summarised views. This will make it very hard to drill down and roll up.
- Sub-problem 6 is not covered since the data is not in a dimensional model, meaning it is impossible to slice and dice the data right from the data model using dimensions and facts. And since dimensions are not used, the context for the measurements might be harder to get.
- Sub-problem 7 is not covered since data validity needs to be defined for each data model in different databases. And since the 3NF model does not necessarily use known metrics it will be hard to get relevant data out.

From the above it is seen that this solution only covers very few of the sub-problems. Hence this solution is not the proposed solution for this project.

5.3.3 Star schema model

Another solution to the sub-problems is to model the data as a dimensional model in a relational database, also known as a star schema.

5.3.3.1 Pros

When the data is in a dimensional model, like a star schema, it is optimised to handle various querying of large sets of data. This means that the overall query performance is significantly better than when using a 3NF model.

The star schema enables better integration between operational source systems in a single database, even in single tables. This makes it possible for end-users to query exactly the data they want, no matter which operational source system the data comes from. I.e. it makes it possible to drill across, given that conformed dimensions are defined in the dimensional model. This can give a more complete picture of the house functioning.

Extendibility is also easily possible when using a star schema, since new fact rows and dimension rows can be added. If changes to the dimensional model is needed, e.g. to accommodate a new fact/metric or dimension foreign key in a fact table, this can usually be done without affecting the existing BI applications. The same goes for adding new descriptive attributes to dimensions, these can also be added without affecting existing BI applications. However when these changes needs to be added to the BI application, these will need to be modified. Also when removing fact/metric columns in a fact table or attributes/columns in a dimension table, this will need modifications in the BI applications.

Since the data is located in another database/other tables than the databases/tables that the operational source systems use, the various, sometimes large and computational heavy, querying will not affect the operational source system. This makes it possible for these systems to keep their level of reliability and availability.

Data quality and validity in a star schema can be enforced and ensured across operational source systems, which makes it possible to easily compare raw and summarised data across these systems, if they are comparable. This can also be indicated to the end-users, through labelling and naming.

The relations between dimension tables and fact tables are ensured by referential integrity done by the RDBMS system, which means that foreign keys will always resolve to a proper dimension row.

More and better analysis and database tools can handle a dimensional model, because they do not need to alter the way the data is modelled before using it. This also makes it easier for end-users to use these tools, because they do not need to modify the data before using it.

The way the data is modelled also gives users a more tangible feel for the data, since they can imagine it as a cube. The data is also modelled for ease of use, so no confusing joins are needed, and given that the labelling and naming is done for usage by end-user and not by operational source systems, makes it possi-

ble for end-users to easily identify the desired tables and attributes which they need for their analysis. This means an increased usability.

The scalability is much better when using a star schema for the amount of users and their queries, since these are handled separately from the operational source systems.

5.3.3.2 Cons

Since the data needs to be loaded into the star schema, and hence be relocated, the data will not always be up-to-date with the operational source systems. However this can be done more frequently to get the data up-to-date at a certain level.

A dimensional model in general will always need more storage capacity than a 3NF model, since the data is denormalised.

Since the data is in a RDBMS system the query performance for the summarised data is not optimised, because the summaries needs to be (re-)calculated during querying, and these will most likely not be cached, so the next queries will also perform badly.

The attribute hierarchies of the dimension tables, do not have a strong relationship, this means that rolling up summarised data is done manually by the end-users or the BI applications. This both decreases performance, but it also means that end-users will not be able to fully utilise the intelligence provided by the attribute hierarchies.

Building a proper dimensional model from the data models used by the operational source systems will take longer to implement, than using a 3NF model or using the existing systems. Because this requires a completely new structure for the data as well as building several other components which handle the transition of the data, e.g. ETL components.

Having a star schema in a RDBMS system does not provide any proper authorisation other than being able to give access rights to certain tables or databases, which means that end-users given access will not only have access to summarised data, they will also have access to the raw data (the most granular level defined for a fact table).

5.3.3.3 Sub-problems covered

The sub-problems covered with this solution includes (1), (2), (3), (4), (7) and to some degree (5) and (6).

- Sub-problem 1 is covered since the data is modelled for easy retrieval of these facts (metrics) as well as their context through dimensions.
- Sub-problem 2 is covered since the data model allows for quickly getting summarised data to generate these graphs e.g. using an analysis tool.
- Sub-problem 3 is covered since conformed dimensions allow for comparing across different metrics, e.g. from different operational source systems.
- Sub-problem 4 is covered since the graphs should be easily added without too much data processing, e.g. manually, since the data is modelled for most analysis tools. Which also means that summaries can be quickly generated by these analysis tools. The model supports understandability in a high

degree since it can be imagined as a tangible object like a cube with the edges as dimensions and the inner points of the cube as facts.

- Sub-problem 5 is mostly covered, however since there are no strong relations between the attributes, and thereby attribute hierarchies in the dimension tables. It is up to the end-users to use the attributes properly as hierarchies. Drilling across is available in a dimensional model, as long as conformed dimensions are used. The raw data will also be available as long as the grain is defined to include the raw data, otherwise only the most granular level is available. The model supports drilling through nevertheless.
- Sub-problem 6 is also mostly covered since the star schema models a dimensional model, where it is hence possible to specify the dimensions and facts for the desired view of summarised data, i.e. slicing and dicing. The model includes dimensions to give context. However since the model needs to calculate all the summaries at query time, the performance can be worsen by making large queries.
- Sub-problem 7 is also covered because data validity is defined for the star schema, which includes data from multiple sources. And the known metrics for end-users are defined clearly in the star schema through proper labelling, so these are easily available to them.

From the above it is seen that this solution covers all of the sub-problems. Hence this solution is a viable solution for the proposed solution for this project. However because of the query performance another solution might be better.

5.3.4 Multidimensional system with an OLAP cube

The last solution for the sub-problems is to use an OLAP cube, which is deployed in a multidimensional system. This solution can be split into two solutions, because the cube needs to get the data from a data source, this data source could either be a 3NF model or a star schema.

5.3.4.1 Pros

The pros for a cube based on a 3NF model, in addition to the pros given in the 3NF solution, are listed in the following:

It transforms the 3NF model into a dimensional model, thereby providing dimension tables and fact tables based on the model. The dimensional model is highly optimised for querying, making this solution much better than the ordinary 3NF model solutions. Most OLAP systems can access multiple operational source systems, which enables easy integration between these systems. This makes it possible for end-users to query across the 3NF models, by using the dimensions and facts of the provided dimensional model through the cube.

Since the data is provided in a dimensional model, the data does not need to be remodelled or prepared before it can be loaded into analysis tools.

The data is always up-to-date if the 3NF model and databases of the operational source systems are used. Also using the existing databases, means that increased storage capacity is almost not needed, however the cached data as well as the preprocessed data needs to be saved.

The end-users will not have to learn a complex data model which is used by the operational source systems, but can be given a subset of the data model, easing the usage.

The pros for a cube based on a star schema, in addition to the pros given in the previous solution are listed in the following: Since the star schema is already a dimensional model the data is accessed even quicker by the OLAP system.

Implementing and deploying the OLAP cube is very easy when it is based on a dimensional model like a star schema, since every dimension and fact in the cube has a one-to-one relation with the underlying model.

Referential integrity is ensured by the underlying data source, ensuring correct behaviour and calculations done by the cube.

Data quality and validity is ensured by the ETL system which populates the star schema, hence the data quality and validity of the cube is better than for a 3NF based solution.

Extending the star schema makes no difference in the cube provided by the OLAP system, when the model in the OLAP cube needs to be modified, these changes are easily loaded into the model, because of the one-to-one relation.

The operational source systems can perform as designed, since the querying and pre-calculations are done elsewhere, than in the databases used by these systems.

The pros for both solutions are listed in the following:

Using an OLAP cube makes it possible to use some of the best analysis tools, making it easier to query the data and hence make better reporting.

The model provided by the cube is a dimensional model, making it more tangible for end-users.

Since the summarised data is pre-calculated and cached the query performance is much better than the other solutions.

The attribute hierarchies are predefined in an OLAP system, meaning the hierarchies can be used properly and as intended by the end-users.

Authorisation can be managed on several levels of detail, e.g. it is possible to give access right only for summarised data, while access to the most granular (raw) data can be given to a select number of end-users.

The dimensional model means that data can be compared across the fact tables, if conformed dimensions are available.

5.3.4.2 Cons

The cons for a cube based on a 3NF model are listed in the following:

Even though the query performance is optimised, basing the cube on a 3NF model decreases the processing performance a lot, since the data needs to be loaded through different data sources, and the model does not have a one-to-one relation with the dimensional model, which means that this needs to be processed as well. Query performance is also decreased because the data needs to be retrieved e.g. from several operational source systems, and depending on the complexity of the model, expensive joins are necessary, to

make a dimensional model with the right measures out of the 3NF model.

No real referential integrity is guaranteed since facts and dimensions are retrieved separately, which might result in unexpected behaviour and calculations made by the cube.

Extendibility is also a problem with this solution, because extending the model to include new fact tables and dimension tables requires the OLAP system to process even more joins and navigations to get the proper data out of the operational source systems.

If the same databases and tables are used by the OLAP cube and the operational source systems, querying will affect the performance of these systems negatively, since the cube will make various and often large queries to the database, especially during pre-processing and under end-user querying.

Data quality and validity cannot be ensured to a very high level, the OLAP system can only filter rows out based on basic SQL, if more advanced filtering and cleansing is needed, this has to be done in an ETL system. This is a significant problem when using the databases used by operational source systems as the base for an OLAP cube.

When using this solution, what is essentially being done is making an ETL component inside the OLAP system, this should never be done, because the lack of flexibility makes it almost impossible to make a proper ETL system. It also delivers an overall inferior performance because of the tight coupling between the data model of the operational source systems and the dimensional model provided by the cube. The OLAP system should be kept as a presentation area for a data warehouse.

The cons for a cube based on a star schema are listed in the following:

The most significant cons is that the data is not always up-to-date with the operational source systems because of the loose coupling between these systems and the data source for the cube.

The data is relocated into a denormalised format, which requires an increased storage capacity.

The cons for both solutions are listed in the following:

Most OLAP providers provide proprietary analysis tools, which means these tools are often only available on certain platforms.

Using an OLAP system requires the data to be pre-processed, this means that it takes longer before the data is available for end-users.

5.3.4.3 Sub-problems covered

The sub-problems covered with these solutions includes (1), (2), (3), (4), (5), (6) and (7).

- Sub-problem 1 is covered since all the data/metrics from the facts are easily available through the dimensional model provided by the cube. The context is provided by the descriptive dimensions.
- Sub-problem 2 is covered since OLAP cubes can be used by advanced analysis tools, which provide easy use of graphs and other visualisation techniques.
- Sub-problem 3 is covered when using conformed dimensions, since these enable seeing relations between fact tables of different operational source systems.
- Sub-problem 4 is covered since most of this is possible using analysis tools that can be used with

OLAP cubes. The cube design also makes it easy for end-users to understand the data and hence makes it more tangible for them. Calculations are done by the OLAP system, which means that the analysis tools do not have to calculate anything to give summarised data. And since analysis tools works directly with OLAP cubes, no manual effort is needed before the tools can be used.

- Sub-problem 5 is covered since OLAP cubes have strongly defined hierarchies, which makes it possible to utilise the intelligence of hierarchies fully to, both, roll up and drill down summarised data. If conformed dimensions are used, drilling across is also possible. Using an OLAP cube also gives the ability to drill all the way through to the most granular data. And because the OLAP system pre-calculates the summarised data, query performance is much more superior than any of the other solutions. If a star schema is used as the foundation for the OLAP cube, the query performance is even better, especially when drilling down in the data.
- Sub-problem 6 is also covered since the data is provided in a dimensional model, where facts and dimensions can be specified to slice and dice a cube. And the context is given by the dimensions. Since the most data is pre-calculated, the query performance, even for larger queries is better than when using a star schema alone.
- Sub-problem 7 is also covered because the data validity can both be defined in the underlying data model, if a star schema is used and in the OLAP cube itself. The known metrics for end-users can exist in a star schema, and even more known metrics can be defined in the OLAP cube.

From the above it is seen that this solution covers all of the sub-problems. The pros and cons show that an OLAP solution which is based on a star schema is the most viable and should hence be considered as a proposed solution.

5.4 The proposed solution

From the above it is clear that the best and most viable solution is a combination of a star schema and a multidimensional system with an OLAP cube.

This solution will require an ETL system for the star schema and an OLAP system for the data cube. This will require more development, but the outcome should be significantly better, than any of the other solutions, in terms of processing, querying and overall performance, as well as better data quality and validity. And because of the underlying dimensional model end-users will understand the data much easier. The solution also makes it possible to use some of the best analysis and database tools, making reporting, querying and decision support much easier for end-users.

5.5 Business Intelligence Reporting

One of the purposes for the DW/BI solution made for this thesis, is to provide reports, which is also known as Business Intelligence Reporting. These reports are often called "Flash-reports" in a business context, because they give a quick overview of the state and performance of a business. These reports are usually directed to top- and middle management. In this thesis these reports will be called "Insight Reports"

(Insights), because they will be able to provide occupants and other stakeholders quick overviews and insights into the overall house functioning of the EMBRACE house.

The overviews are made by selecting dimensions and facts. These will be identified and determined in the design chapter. The following are the minimum required measurement types that should be available in the insight reports.

5.5.1 House measurement types

The following measurement types should be available for the insight reports in the DW/BI solution to be able to provide proper insights into the house functioning. The list is prioritised, with the most important measurement types first.

1. Energy consumption divided by the different types of consumption in the EMBRACE house. This includes the energy consumed by the following appliances: Dishwasher, Dryer, Fridge, Lights, Oven, Stove and Washing machine. It also includes the energy consumed by the following devices: Heat pump, Nilan system, Control systems (IHC and PLC) and other pumps.
2. Total energy consumption. This measures the total energy consumed from the power grid, i.e. it does not include the power sent to the power grid from the energy produced by the PV panels.
3. Energy production from the PV panels. It should be noted that, during competition, the energy produced is not consumed by the EMBRACE house itself, instead all the energy is sent to the power grid.
4. Dimmable light levels and the duration for each level, the duration is calculated based on the transition between levels.
5. Indoor and floor temperature measurements from different places in the house.
6. Humidity measurements.
7. Light switch states and the duration for each state, the duration is calculated based on the transition between states.
8. Water flow/pressure for the solar panels.
9. Water temperature and general temperature for the solar panels and storage tanks.
10. Door and window open states and the duration for these states, the duration is calculated based on the transition between states.
11. Motion detected by Passive Infrared (PIR) sensors, this is recorded as states, which means that the duration can also be calculated. There are two types of PIRs: alarm PIRs and regular motion detectors. The alarm PIRs are less sensitive and more accurate.
12. Power outlet states and the duration for these states. It should be noted that the power outlets of the EMBRACE house are double outlets, where one is constantly turned on.
13. Brightness or daylight sensor, which captures whether there is light outside. This sensor is state based, hence the duration can also be calculated.
14. Maintenance of the solar panels, which is state based, so the duration can be calculated.

5.5.2 Weather measurement types

In addition to the above house measurement types, the following forecast weather measurement types should be available for the insight reports. The list is prioritised, with the most important measurement types first.

1. Weather description, e.g. clear sky, raining, etc., which gives a rough description of the weather.
2. Cloud cover, which is the percentage of cloud cover. This can e.g. be used to see why energy production is low or high.
3. Humidity, for the outside humidity.
4. Pressure, which can be used to determine whether stable or unstable weather is expected.
5. Amount of precipitation, e.g. rain or snow.
6. Temperature, expected actual, high and low temperatures.
7. Wind speed and direction.

The above measurements will most likely be expected measurements from forecasts, however if the current forecasts for the current hour is used, they should be of value to the insight reports.

5.5.3 Competition measurement types

During the competition the Solar Decathlon Europe organisation will also be making measurements. The following measurement types should be available for the insight reports. The list is prioritised, with the most important measurement types first.

1. Indoor air quality.
2. Temperatures in rooms, dishwasher, washing machine, freezer, fridge and oven.
3. Indoor humidity.
4. Power measurements including PV, consumption for different elements of the house and the total power taken from the grid. The total power taken from the grid, includes the power produced, i.e. a negative value means that more energy was produced than was taken from the grid.

These measurements will mostly be used by researchers and students after the competition, however these measurements can also be used later when comparing the current situations with the situations during the competition. Hence this provides better insight reports.

5.6 Operational Source Systems

The following operational source systems have been identified as the primary raw data sources.

5.6.1 Hardware System Integration

The hardware system integration (HSI) service integrates the control systems in the EMBRACE house(s). It enables control of these control systems by setting set points and reading values.

It provides access to items, which are generic devices. The integration makes it possible to manipulate with these items through three basic operations: UpdateEvent, Read and Update.

- The UpdateEvent are for event handlers, which are fired when a new value is available for the item.
- The Read operation is to read the current value from the control system.
- The Update operation is used for setting a set point in the control system.

The service can integrate multiple control systems, called hardware systems. For the EMBRACE house, an Intelligent House Concept (IHC) integration, Programmable Logic Controller (PLC) integration and Uponor integration is built into it.

The items exposed from the service originates from these control systems, which is why the items are generic, and are used using the interface `IItem`.

The relation between an item and the control system is based on item types. Item types are also generic, and hence interfaces are used when using these objects.

The integrations also provide a number of grouping objects, one for houses, one for floors and one for groups.

5.6.1.1 Interfaces and Entities

The generic item type, `IItemType`, consists of a name, description, type, as well as whether the type is an interval or has states, if so these are defined in a list in the item type.

There exists three types of item types, these are the entities which implements the generic item type. One for numeric values, which is called `ItemTypeNumeric`. One for boolean values, which is called `ItemTypeBool`. And the last one is for a custom number of states, which is called `ItemTypeCustom`.

- `ItemTypeNumeric` objects contains a maximum and minimum value. The item type numeric is an interval, which is set for the generic item type.
- `ItemTypeBool` objects contains a true and false state. States are objects which have a string value. Using states makes it possible to give item values a user-friendly label. E.g. "Window Opened"/"Window Closed" instead of "True"/"False". The item type bool is a simplified item type custom, which only have two allowed states for true and false.
- `ItemTypeCustom` objects contains a list of states, which can be used for this item type.

The different entities from the HSI system is described below:

- The generic item `IItem` consists of a name, a description, a list of groups, the item type and a value which is a generic object.
- An item object, `Item`, contains a unique id, a name, a description, both the generic item type and the actual item type, a system variable (which is used to identify the item in the originating control system), a value, a list of groups and the floor.
- A group object contains an id, a name, a description and a reference to a parent group. Groups are made as trees of groups and items. Groups are independent from items, floors and houses, hence they could group items between houses and floors.
- A floor object contains an id, a name, a description and a list of items and a reference to a house object.
- A house object contains an id, a name, a description and a list of floors.

5.6. Operational Source Systems

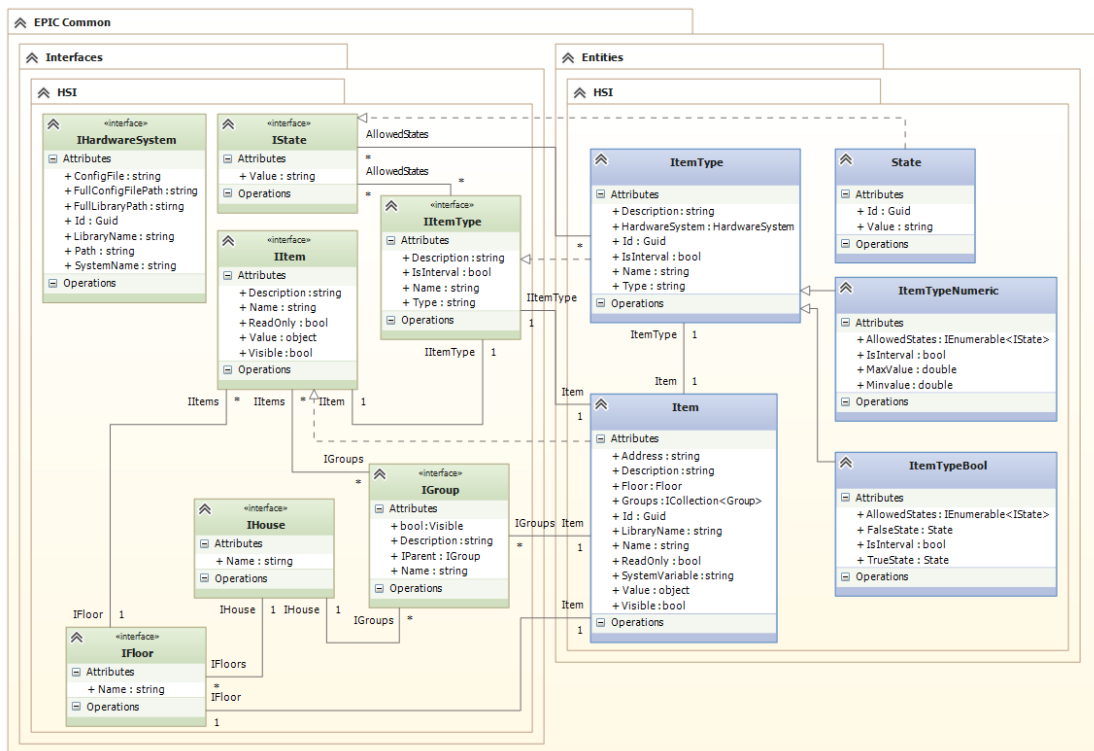


Figure 5.6.1: Excerpt of HSI common interfaces and entities

The most essential interfaces and entities from the HSI system used for the DW/BI solution can be seen in figure 5.6.1.

5.6.1.2 Item types

The following item types are available from the integration service. The boolean item types are seen in Table 5.6.1. The numeric item types are seen in Table 5.6.2. There are no custom item types available for the EMBRACE house, so this type will not be supported in the DW/BI solution.

Name	Hardware system	True label	False label
Alarm noise	IHC	Activated	Passive
Alarm PIR	IHC	Activated	Passive
Darkness sensor	IHC	Dark	Light
Diode	IHC	On	Off
Door open sensor	IHC	Open	Closed
Light	IHC	On	Off
On/off	PLC	On	Off
PIR	IHC	Activated	Passive

Power plug	IHC	On	Off
Power sensor pulse	IHC	Activated	Passive
Pulse counter reset	IHC	On	Off
Push button	IHC	Activated	Passive
Window open sensor	IHC	Open	Closed

Table 5.6.1: Boolean item types

Name	Hardware system	Min value	Max value
CO2	PLC	0	100
Dimmable light	IHC	0	100
Floor temperature	Uponor	10	30
Humidity	PLC	0	100
Indoor Temperature Sensor	IHC	-5	40
Lux sensor	IHC	0	100
Outdoor Temperature Sensor	IHC	-40	40
Power sensor accumulated	IHC	0	9999
Pulse counter	IHC	0	999999
Technical Temperature Sensor	IHC	-100	999
Water pressure	PLC	0	20
Water temperature	PLC	0	200
Window	IHC	0	100

Table 5.6.2: Numeric item types

Note the "Pulse counter" and "Pulse counter reset" item types. These are the power meters which counts pulses for each 1/100 and 1/1000 kWh passed though the meter. These are used to measure the energy consumption and production from the house. To calculate the additive measure kWh and the non-additive measure kW from these power meters the following equations are needed.

For the normal power meters, which counts for each 1/100:

$$\begin{aligned} n/100 &= x \text{ in kWh} \\ (n/100)/(s/3600) &= x \text{ in kW} \end{aligned} \quad (5.6.1)$$

For the main power meter, which counts for each 1/1000:

$$\begin{aligned} n/1000 &= x \text{ in kWh} \\ (n/1000)/(s/3600) &= x \text{ in kW} \end{aligned} \quad (5.6.2)$$

The n is number of pulses, s is the duration between readings in seconds, and the x is the final measure. Note that to get the measures the counters needs to be reset, which is done by sending a true and false flag to the "Pulse counter reset" item.

5.6.1.3 Values

The values inside the item objects contains the interesting metrics. These are the values which are read from the control systems.

The value field is set on response messages to `UpdateRequest` and `ReadRequest`. The value is also in `UpdateEvent` messages. These response messages are sent out in the P2P network, when other system components requests updates or reads. And when a new value is available the `UpdateEvent` messages are sent.

5.6.2 Data Collection

The data collection service collects the data from the control systems which are integrated in the HSI. This system does not exist as part of the HSI system, hence this system should be made as part of the Data Warehouse solution.

As indicated above the HSI system sends response messages for the requests sent by other system components in the P2P network. By subscribing to these response and update event messages, it is possible to get a copy of these messages. This makes it possible to save the measurements in a operational 3NF model. By using a 3NF model the service can be optimised to handle each message as a transaction, and thereby handle multiple messages one by one, ensuring all data sent in the P2P network is collected.

This system is essential, because there are no other data sources for these measurements after the competition.

5.6.3 Weather Data

Weather data can be provided from any web service based weather service.

Two alternatives have been found, `OpenWeatherMap.org` and `yr.no`. Both provide XML based weather data for the required locations, which are Kongens Lyngby, Denmark (where the house was built), Versailles, France (where the competition is held) and Nordborg, Denmark (where the house will stay after the competition).

The `yr.no` service provides most of the weather information required for the insight reports, including weather description, precipitation, wind direction and speed, as well as current temperature and pressure. However the cloud cover and humidity are not available. The cloud cover is quite essential to be able to make comparisons of cloud cover and the energy produced by the PV panels.

The `OpenWeatherMap.org` provides all of the weather information required for the insight reports. Also the cloud cover and humidity. It also provides expected high and low temperatures in addition to the current temperature.

Another difference between the two services is that `yr.no` forecasts weather in periods of six hours, while

OpenWeatherMap.org makes forecasts for every hour.

The OpenWeatherMap.org should be used for this DW/BI solution.

5.6.4 SDE Measurements

During the competition in Versailles the organisation will collect measurements from the houses, these measurements are:

- Indoor air quality in ppm.
- Temperatures for two rooms, dishwasher, washing machine, freezer, fridge and oven in degrees Celsius.
- Indoor humidity in percent.
- Power measurements including PV, consumption for appliances, lighting and home automation, consumption for home electronics and the total power taken from the grid. The total power taken from the grid, includes the power produced, i.e. a negative value means that more energy was produced than was taken from the grid. The power measurements are in W.

The measurements are available through the official website for the competition. The website provides the raw data through JSON, which is data usually used by JavaScripts on websites.

The data is separated in several files for each type of measurement. The JSON contains data for all houses. The EMBRACE house is given a specific id, which needs to be used when retrieving the data.

The general format of the JSON is an array with an element for a timestamp starting from July 30th, 2014 at midnight, the timestamp is minutes since this date and time.

For each element in the time array is another array with an element for each house. The EMBRACE house is identified as number 16. For the house element a third array is defined which contains the individual measurements. If a measurement is not available it is filled with a null value.

5.7 Requirements specification

The above analysis give rise to the following summary of the functional and non-functional requirements for the DW/BI solution.

5.7.1 Functional requirements

The DW/BI solution should be made using an OLAP system based on a star schema dimensional model.

The functional requirements for the DW/BI solution are given below:

1. Gain knowledge about the functioning of the house
 - (a) Gain knowledge about energy consumption and the context in which it is consumed
 - (b) Gain knowledge about energy production and the context in which is it produced
 - (c) Gain knowledge about how the devices in the house are used and how long they are in different states

- (d) Gain knowledge about the temperature, humidity and other indoor comfort conditions which might affect the usage of the house and the context for these
- 2. See how elements relate
 - (a) Seeing differences in curves in a graph for elements in a house
 - (b) Seeing similarities in curves in a graph for elements in a house
 - (c) Ability to compare graphs to further analyse relationships
- 3. See how elements of the house and outside conditions relate
 - (a) Ability to see outside conditions like temperature and cloud cover
 - (b) Ability to compare graphs from outside conditions with indoor comfort conditions
- 4. Perceive the numbers as tangible elements
 - (a) Ability to easily add graphs to be generated in reports, instead of showing raw numbers
 - (b) Ability to summarise as many numbers as possible, so no additional calculations are needed to get the full picture
 - (c) Ability to change formatting of numbers so they seem familiar to the occupants
 - (d) Ensure understandability of the data model, e.g. by using a tangible object like a cube.
- 5. Drilling through data to analyse it
 - (a) Ability to summarise data from a hierarchy level above the current summarised view (Roll up)
 - (b) Ability to summarise data from a hierarchy level below the current summarised view (Drill down)
 - (c) Ability to summarise data from different measurements (Drill across)
 - (d) Ability to drill all the way down to the raw data (Drill through)
- 6. Slice and dice data cubes
 - (a) Ability to specify the view for specific facts
 - (b) Ability to specify the view for specific dimension to see the facts
 - (c) Ability to give numbers context by means of dimensions
- 7. Get relevant and valid data
 - (a) Ability to calculate with known metrics
 - (b) Having valid data

5.7.1.1 Measurement types

The measurement types that should be available for insight reports are given below:

- 1. House measurement types
 - (a) Energy consumption divided by the different types of consumption in the EMBRACE house. This includes the energy consumed by the following appliances: Dishwasher, Dryer, Fridge, Lights, Oven, Stove and Washing machine. It also includes the energy consumed by the following devices: Heat pump, Nilan system, Control systems (IHC and PLC) and other pumps.
 - (b) Total energy consumption
 - (c) Energy production from the PV panels
 - (d) Dimmable light levels and the duration for each level

- (e) Indoor and floor temperature measurements from different places in the house
 - (f) Humidity measurements
 - (g) Light switch states and the duration for each state
 - (h) Water flow/pressure for the solar panels
 - (i) Water temperature and general temperature for the solar panels and storage tanks
 - (j) Door and window open states and the duration for these states
 - (k) Motion detected by Passive Infrared (PIR) sensors
 - (l) Power outlet states and the duration for these states
 - (m) Brightness or daylight sensor
 - (n) Maintenance of the solar panels
2. Weather measurement types
 - (a) Weather description
 - (b) Cloud cover
 - (c) Humidity
 - (d) Pressure
 - (e) Amount of precipitation, e.g. rain or snow
 - (f) Temperature, expected actual, high and low temperatures
 - (g) Wind speed and direction
 3. Competition measurement types
 - (a) Indoor air quality
 - (b) Temperatures in rooms, dishwasher, washing machine, freezer, fridge and oven
 - (c) Indoor humidity
 - (d) Power measurements including PV, consumption for different elements of the house and the total power taken from the grid

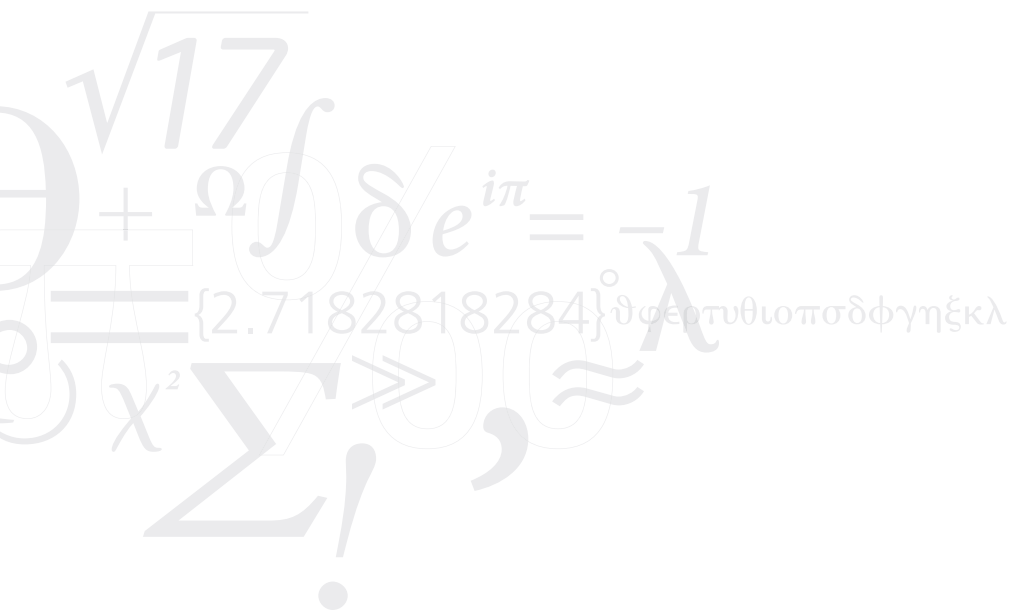
5.7.2 Non-functional requirements

The non-functional requirements for the DW/BI solution is given below:

1. The response times and refresh times of queries made to the DW should not be longer than one second.
2. The time used on the first loading and processing of the DW should not be longer than one hour depending on the amount of data in the sources.
3. Subsequent ETL operations should be less than 15 minutes.
4. The processing time per row should not exceed 100 ms.
5. A fall-back data warehouse should be used during maintenance.
6. The system must be extendible to be able to accommodate new data sources.
7. The interface and data presented to the end-users must be intuitive and easy to use.
8. The data should be secured in the cloud and only available to authenticated users.
9. The integrity and validity of the data must be adequate enough to give a complete picture.
10. All discarded data should still be logged in order to do auditing at a later stage.

11. The data must be consistent.
12. The storage capacity for the DW should be extendible. The initial storage capacity for the data warehouse should not exceed 250 GB.
13. The system should initially handle 6250 queries per hour on average.
14. The availability should be 95%, meaning $365.242 \text{ days} \cdot (1 - 0.95) = 18.3$ days of unavailability a year.
15. Recoverability of the data warehouse should be established.
16. Compatibility is limited to a Windows environment with the .NET Framework.
17. Communication with other components of the cloud is done through Kademia P2P and REST web services.

This page intentionally left blank



Chapter 6

Design

”There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”

—Charles Antony Richard Hoare, 1980

Contents

6.1	Architecture	71
6.2	Data Collection	74
6.3	Data Warehouse	81
6.4	ETL	89
6.5	BI Application	93

This chapter provides the overall design and architecture of the data warehouse solution.

6.1 Architecture

Based on the requirements defined in the previous chapter, the overall project must be split into at least three service applications. One service application which can collect the data which is sent in the P2P network, and is thereby an operational source system in DW/BI solution context. This application will be called Data Collection. Another service application which can handle the data from the different operational source systems and do overall ETL processing. This application will be called ETL. And finally a BI service application, which exposes the data from the data warehouse in cubes and enables querying of these. This application will be called BI. These three applications should work as standalone applications, hence no tight coupling should be made between them. The data collection and BI applications will have to be connected to the P2P network to be able to communicate with the other systems in the network, notably the Hardware Systems Integration from the home connector system, and the App Service which handles communication outside the P2P network and thereby e.g. with the tablet application. The overall architecture of the DW/BI solution can be seen in Figure 6.1.1. The figure shows the operational source

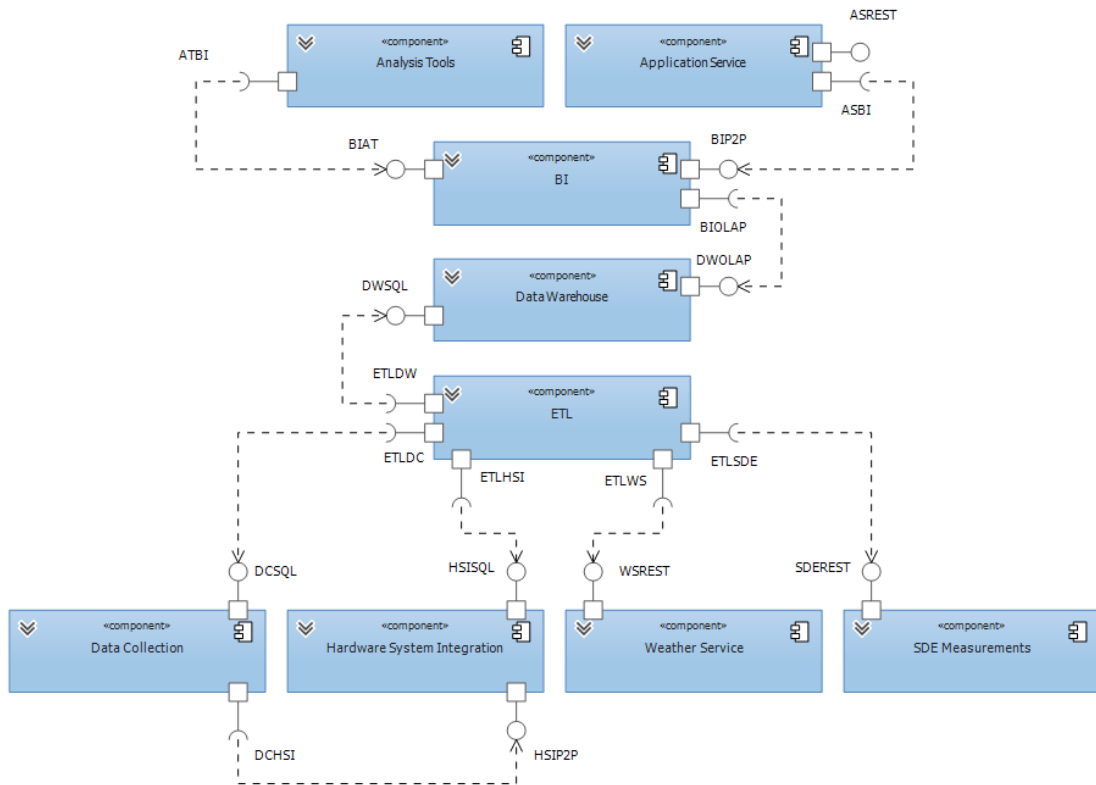


Figure 6.1.1: Architecture of the DW/BI solution

systems: Data Collection, Hardware System Integration, Weather Service and SDE Measurements. It also shows the DW/BI solution which consists of the ETL, Data Warehouse and BI components. And lastly the components which consumes the BI service, i.e. Analysis Tools and the Application Service (App Service). The Data Collection component requires the HSI, which it communicates with through P2P (denoted by DCHSI and HSIP2P in figure). The Data Collection service subscribes on messages sent from the HSI. A helper service in the Data Collection component also sends requests, i.e. publishes messages to the HSI, which is also done through P2P. Note that the port is only one-way, this is because the Data Collection component knows the existence of the HSI component, but the HSI component does not know the existence of the Data Collection component. The operational source systems are split into two types, one which is accessible through SQL (denoted by SQL ending of DCSQL and HSI SQL in figure) and another which is accessible through REST web services (denoted by REST ending of WSREST and SDEREST). The Data Collection and HSI components are accessible through SQL. The Weather Service and SDE Measurements are accessible through REST web services. Both types are consumed by the ETL component (denoted ports ETLDC, ETLHSI, ETLWS and ETL SDE), which makes it possible for the ETL to process data from the four operational source systems. Note that the four operational source systems only exposes interfaces, hence the systems does not know the existence of the ETL component. But the ETL component

knows the existence of these four systems. The ETL processing results in data which needs to be put into a data warehouse. The Data Warehouse component exposes a SQL (denoted by DWSQL) port which is used by the ETL component to load data into the data warehouse. The Data Warehouse component exposes two ports a SQL and an OLAP port. Both ports can be used to query the data warehouse. The SQL port is connected to a RDBMS system, while the OLAP port is connected to a OLAP system in the Data Warehouse component. Note that the DWSQL port exposes SQL from the RDBMS system, which does not know the existence of the ETL component, but the ETL component knows that it should use this port to load data into the Data Warehouse. The same principal is used for the DWOLAP port, which is exposes the OLAP part of the Data Warehouse component and it does not know the existence of its consumers. The BI component handles the communication between the OLAP port of the Data Warehouse and the applications, i.e. Analysis Tools and Application Service. The component acts both as a relay or proxy for the Analysis Tools and as a translate component for the Application service. The proxy port, denoted BIAT, is consumed by the Analysis Tools through port ATBI. The translate port, denoted BIP2P, is consumed by the Application Service through port ASBI. Note that the BIP2P port subscribes to P2P messages, which can be published by any system in the P2P network. Both the BIP2P and BIAT ports are one-way, since the BI component does not know the existence of any of its consumers. The Application Service uses the BI component through the ASBI port which it uses through P2P. The Application Service is a relay or proxy component which makes it possible for systems outside the P2P network to use the systems inside the P2P network. The Application Service component exposes a REST web service through the ASREST port. This component also uses other subsystems e.g. the HSI component, however this is not of concern to the DW/BI solution, hence not shown in the figure. The Analysis Tools component are analysis tools which can consume the data from an OLAP port, e.g. the relayed port BIAT through the ATBI port. Note that these tools can also consume data directly from the OLAP port from the Data Warehouse component, however this usually requires that the Analysis Tools are used on the same network as where the Data Warehouse is deployed. For this DW/BI solution it is most likely not the case that the analysis tools are used on the same network, hence the BI component needs to act as a relay or proxy. It should be noted that the components of the DW/BI solution should work independently of each other. E.g. if the Data Collection component is not running it should still be possible for the rest of the solution to work. The same applies for the ETL component, if this is not running, the Data Warehouse should still be able to work, and the Data Collection component should still collect data. If the BI component is not running it should still be possible for the ETL component to load data into the Data Warehouse component. However if the Data Warehouse component is not running, it is okay that the BI component and ETL component are not able to work properly, this should however be handled gracefully. The largest component of the DW/BI solution is the ETL component, which needs to handle multiple operational source systems, transform the data and load it into the Data Warehouse.

6.2 Data Collection

The data collection service application collects data which is sent in the P2P network, notably the set points and values sent by the HSI service as a response for requests sent by other service applications in the network. This means that the data collection service needs to subscribe to the same response messages as other service applications in the network. However the data collection service should not send requests into the network unless specific values must be saved, and which are not used by any other services in the network. By focusing on collecting data and not sending requests it is possible to make the service application as robust and reliable as possible, and hence (nearly) all messages in the network will be received/intercepted and saved. All messages cannot be received and saved, because of the service architecture used in the network, where it is only guaranteed that messages are received as long as the subscription information is in the network. This information might be lost if nodes/services in the network unexpectedly stops. The messages received should just be saved, without any filtering, if possible. Because of the nature of the HSI system which sends values in different types for the three different discriminators. The data collection service should handle these, by means of type casting and conversion. And should save the values in three different tables for each discriminator. The three discriminators are boolean, numeric and custom. The boolean typed item values are always of type boolean. These boolean values have a certain state included in the HSI system, one for true and one for false. The numeric typed item values can be of any numeric type, e.g. int, long, double. These types should be converted to an appropriate container type, which can hold these values. The last discriminator type is custom, which can use any number of states for its value. The custom type is not used for the Embrace house, and hence this type should not be considered, for now, by the data collection system. Figure 6.2.1 shows the components of the Data Collection component. The Data Collection component consists of three components, two services, and a database. The two services communicate with the HSI component, through the DCHSI port. Both services should use a separate P2P broker, so both services can operate as robust and reliable as possible. The Service component is the primary component of the Data Collection component, this handles all received messages and saves them in the Database component. The Helper Service component is only used to make requests to the HSI system through P2P. The Database component contains all the collected data. It also exposes the collected data through SQL which can be consumed by any database reader.

6.2.1 Data model

The data model for the Data Collection component is kept as simple as possible. Figure 6.2.2 shows the data model for the entities used by the data collection service. The data model consists of an interface (IRecord) which defines the required fields of any type of Record. A Record is an entity for a collected message from the P2P network. The response messages should always contain an item id, a sent field containing a timestamp and a value. For the interface it is defined that a Record should have an Id, an ItemId, a Sent and a Type field. The Type field is of a RecordType enumeration, which indicates the type of response message, i.e. a ReadResponse, UpdateEvent, or UpdateResponse. The UpdateRequest should be used, if these need to be collected. This is however not done for the Embrace house. For the Numeric Records a LongValue

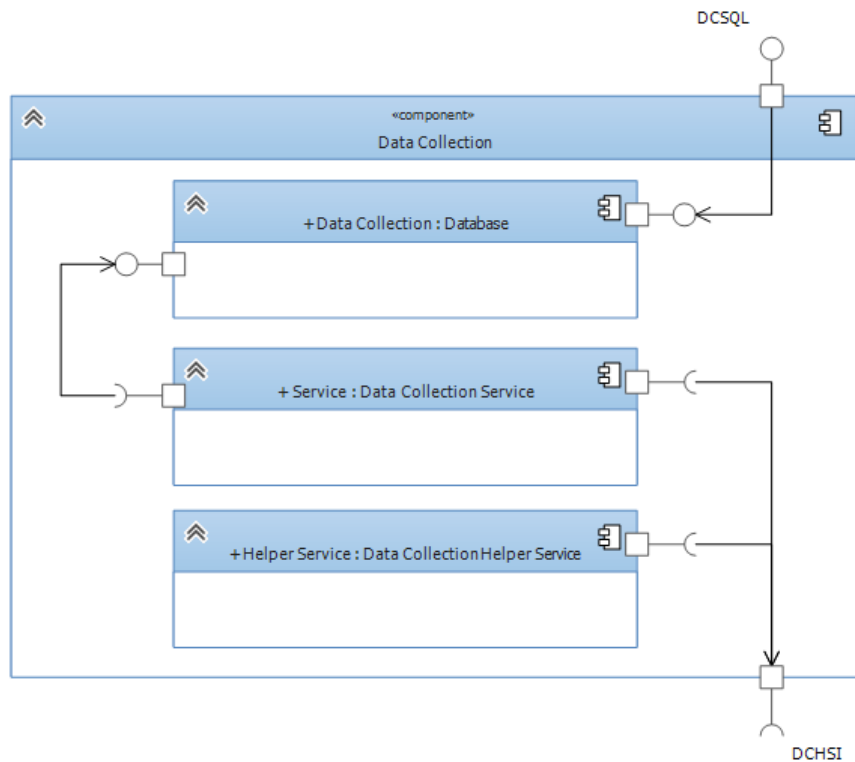


Figure 6.2.1: Subcomponents of the Data Collection component

and DoubleValue field is used to hold the value received in the response message, these are of type nullable long and double, since it is not guaranteed that any numeric data is in the value received. For Bool Records a BoolValue of type bool is recorded, this is not nullable since the type should be inferred by the type, which for a Boolean value will always be either true or false. For received messages which does not fit the criteria of either being a bool or numeric, or if certain fields are missing in the received message, these should be collected as Error Records. This entity has an XmlValue which is a string of a XML serialized value, this makes it possible to identify the correct type, when these records are reviewed. Another important field is the Message field, which is used to indicate what is wrong with the received message. All the other fields are nullable, since these might not have been set in the received message. The difference between the Sent and Received fields is that the Sent is the timestamp set in the received message and the Received is the timestamp for receiving the message by the Data Collection service. The three types of records are numeric and bool which reflect the usage from the HSI system, The last record type is error, which is used to collect and keep record of incorrect received messages and values. The two types of records should only reflect the actual type of the value in the message, and not by the discriminator type of the HSI system. Since the HSI system does not make any distinction on the actual types of the data; inside the HSI system the values are stored and handled as objects, which is problematic when values needs to be recorded e.g. in a database. Hence the type should be reflected on the actual type of the values, so no additional information

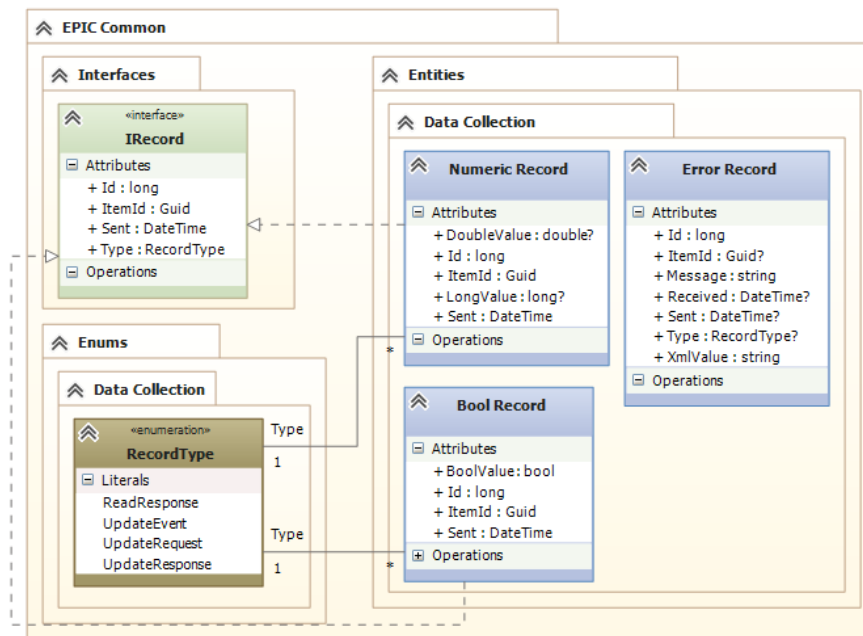


Figure 6.2.2: Data model for Data Collection

is required to be stored, e.g. the full type name. As long as the data is stored in a container type, which can hold the same amount of information as the actual values, these types can be used for the record keeping.

6.2.2 Data Collection application layers

The service application consists of three layers. These layers form a n-tier application structure. The bottom layer handles the data access to the database. The middle layer handles all business logic between the bottom and top layer. The top layer is the "presentation" layer which consists of the service, which communicates with the other services in the network. Since this application is a service it has no actual presentation layer apart from what other services in the network is able to communicate with. The classes of the different layers (packages) are seen in Figure 6.2.3.

6.2.2.1 Service Layer

The service layer communicates with the P2P network through a broker node. Both services use a separate P2P broker. The primary service subscribes to the three message types which includes a reading of the current value of a HSI item (hardware device). The three messages are UpdateResponse, ReadResponse, and UpdateEvent. The UpdateResponse message is a response from the HSI system which is sent to the network, when a request to update the set point for a hardware device is received by the HSI system. The ReadResponse message is a response from the HSI system which is sent to the network, when a request to read the current value of a hardware device is received by the HSI system. The UpdateEvent is a notification

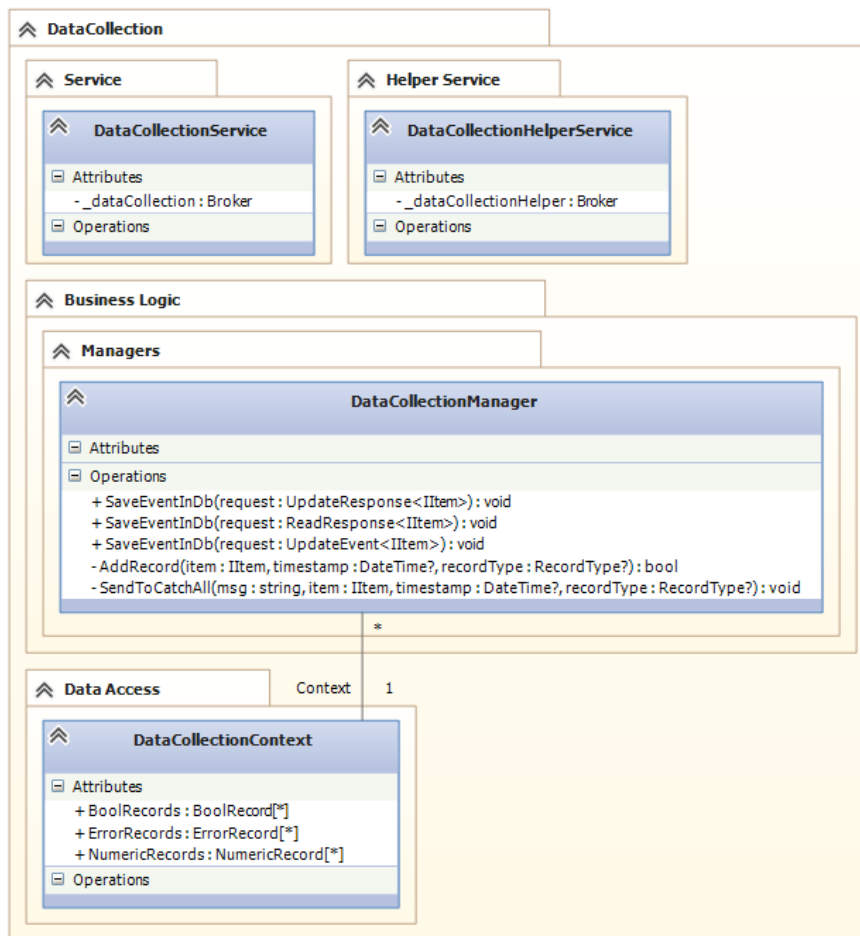


Figure 6.2.3: Data Collection packages

sent to the network because a hardware device updated its current value or set-point. This event notification can be sent multiple times for the same event, so here some filtering must be done. The filtering could e.g. be done by checking whether the same message was received within a given time period. The helper service sends out requests (UpdateRequest and ReadRequest) out in the network for HSI items which are not used/read from other systems in the network. This is e.g. necessary for making readings of the pulse counters mentioned in the Analysis chapter.

6.2.2.2 Business Logic Layer

The business logic layer contains the managers which handle the interaction between the service layer and the data access layer. These managers should handle the extraction of item related information from the messages received by the primary service. They should also handle the conversion and filtering of data. The SaveEventInDb methods handles the callback from the P2P broker. These methods call the AddRecord,

which handles the extraction of item related information, the conversion of the value as well as the filtering of messages. If e.g. an item or the value in the received message cannot be properly inserted into a Record, the `SendToCatchAll` method is used to retrieve as much information from the message as possible and saving it.

6.2.2.3 Data Access Layer

The data access layer and package consists of the database access layer which handles all transactions with the database. The package also includes the three types of entities available for the rest of the application. These entities are called records to reflect the essence of logging and keeping records of the operation of the services for the Embrace house. The DAL contains a Context which handles all the data access to the database.

6.2.3 Helper service

The Helper service is used to retrieve messages which are not send in the P2P network during normal operation, this is e.g. done for the primary service to receive messages from pulse counters. The Helper service is timer, which at periodic intervals publishes request messages in the P2P network. In Figures 6.2.4 and 6.2.5 the sequence of this is illustrated. The lifelines from left to right is: `DataCollectionService`, the primary service, `DataCollectionHelperService`, the helper service, P2P Broker DCS, the P2P broker for the primary service, P2P Broker DCS Helper, the P2P broker for the helper service, P2P Broker HSI, the P2P broker for the HSI system and `HardwareSystemIntegration`, the HSI system. First both the primary service and the HSI system subscribes to their individual messages. When the timer ticks in the helper service a `ReadRequest` with a `IItem` is published to the network, which is received by the HSI. The HSI reads the value from the integration and publishes a `ReadResponse` with the `IItem`. This response is both received by the primary service as well as the helper service, since a synchronous request was sent by this. The primary service saves the event using `SaveEventInDb`. Depending on the value from the response, the helper service will publish a `UpdateRequest` with a `IItem`, which is received by the HSI system, which publishes a `UpdateResponse`. This `UpdateResponse` is also both received by the primary service and the helper service. A second `UpdateResponse` might need to be published by the helper service. Also this issues the HSI to publish an `UpdateResponse` with a `IItem`, which is also received by both the primary service and the helper service. This pattern is used for e.g. the pulse counters, where after reading a value above zero, the pulse counter needs to be reset. The reset is done by setting a true boolean value on the reset item, followed by setting a false boolean value on the reset item. Hence two update requests are sent to the HSI by the helper service. The sequence diagrams also illustrate how the primary service receives response messages when other systems in the network publishes requests to the HSI system, since the helper service could easily be another system in the network. (It actually acts as a separate system for the network, since it has its own P2P broker).

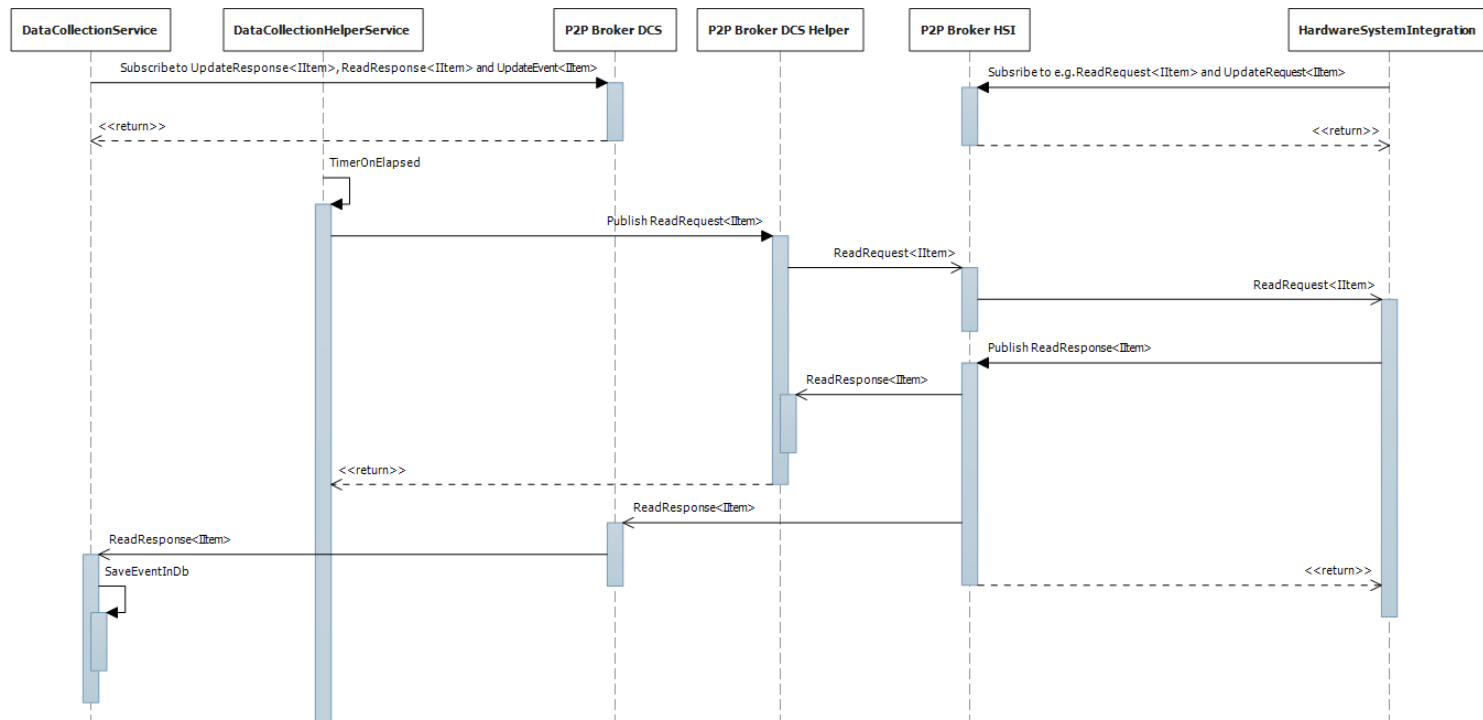


Figure 6.2.4: Sequence for Data Collection Helper service

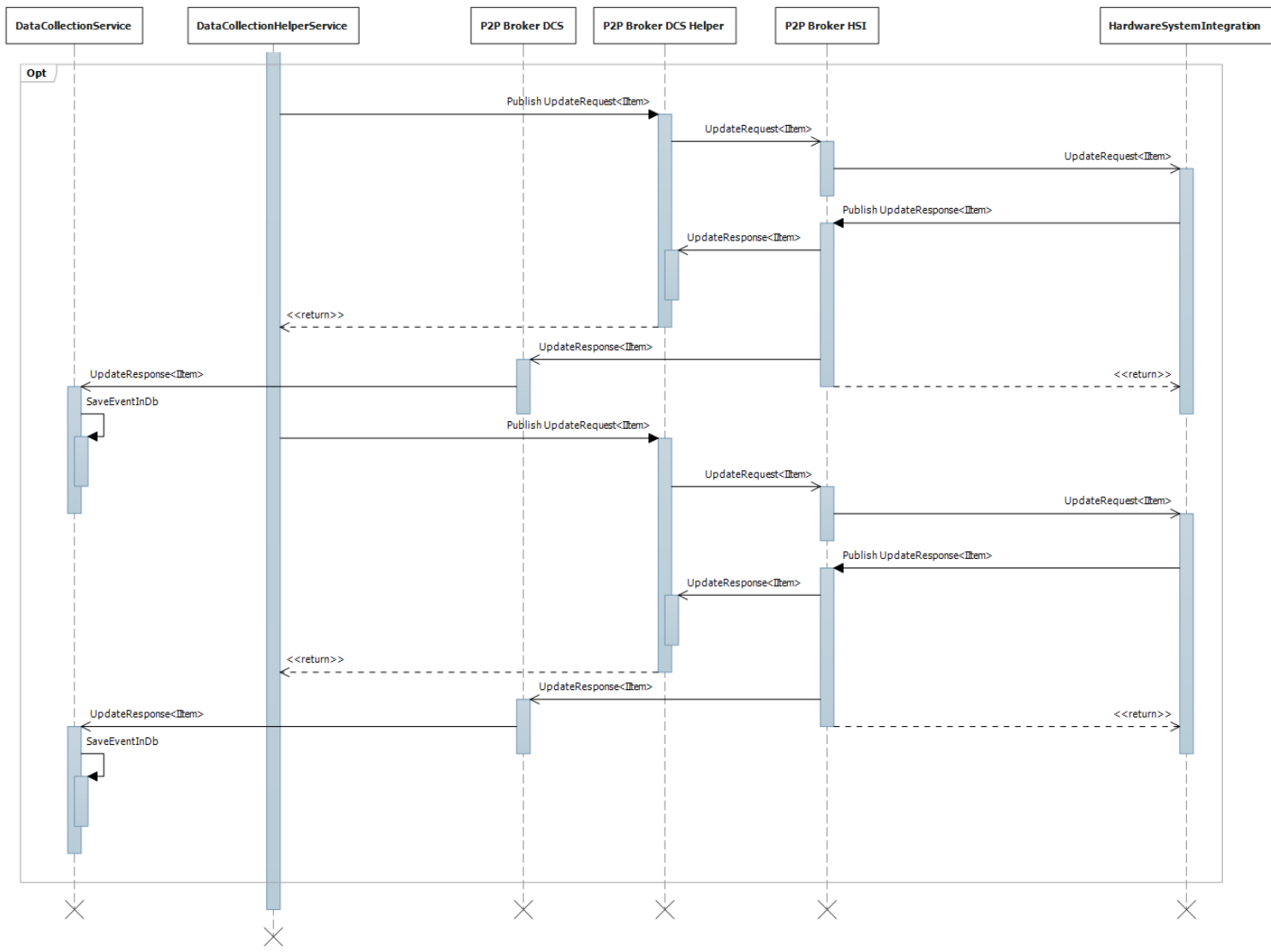


Figure 6.2.5: Optional sequence for Data Collection Helper service

6.3 Data Warehouse

The data warehouse is often referred to as the presentation area of the data, which is generated from the ETL system. This is the data seen/used by the end-consumers, which can either be end-users or end-user analysis tools. Figure 6.3.1 shows the two main components of the Data Warehouse component. The Data

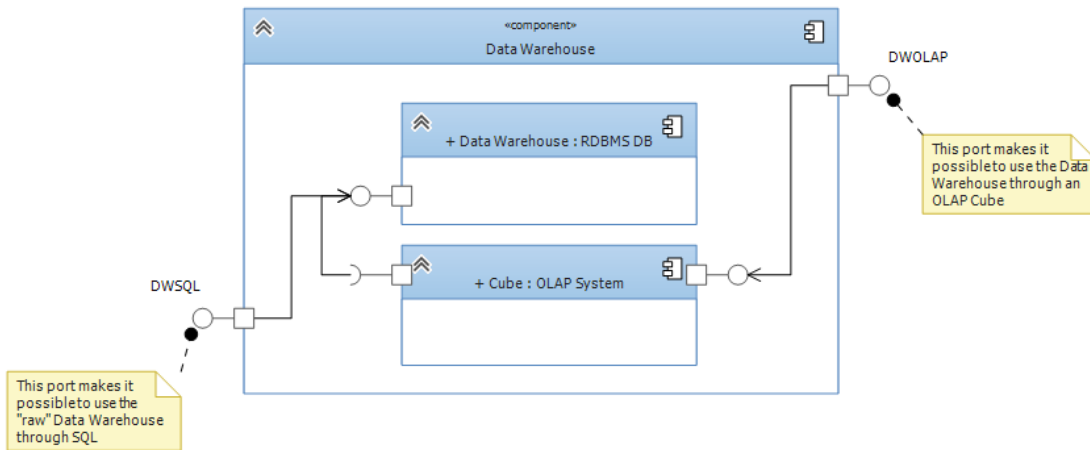


Figure 6.3.1: Subcomponents of the Data Warehouse component

Warehouse consists of a database in a RDBMS system and a cube in an OLAP system. The database is a regular relational database with tables. The OLAP system consumes the relational database and exposes an OLAP cube on this data. This means that end-users can either use OLAP-based or SQL-based analysis tools. The database exposes a SQL port (denoted DWSQL) which is both used by the OLAP system, the ETL system, which populates the data, and by end-users using a SQL-based analysis tool. The OLAP system exposes a proprietary OLAP port (denoted DWOLAP), hence this port can usually only be used by OLAP tools provided from the OLAP system vendor.

6.3.1 Design process

Using the dimensional modelling design process, it is possible to identify the dimensions and facts of the data model, which the data warehouse consists of.

6.3.1.1 Business process

First the business processes from the operational source systems are identified: The four operational source systems handle different operational business processes. The HSI handles integration with the different control systems in the Embrace house. This integration has one major process, which is to facilitate a transparent layer above the different control systems, to be able to control these through a single interface. The Data Collection handles the collection of data which is sent in the P2P network from the HSI system. This is the main process done in this system. The main process for the Weather Service is to report forecasts

for different locations. The main process for the SDE Measurement is to measure different parameters during the competition in Versailles.

6.3.1.2 Grain

With the processes identified it is possible to define the grain for each type of process: The grain for the HSI system is one row per item, per house, per floor, per group. These grains refer to descriptive attributes, since no numeric metric are measurable for the identified process. The grain for the Data Collection is one row per response message. The response messages can be from a read of a value by the HSI, an update of a set point in the HSI, or an update event of a changed value in the HSI. In the data collection each response has one record transaction row. This means that a one to one relationship can be made by using a one row per response message grain, which makes it possible to drill all the way through to the most atomic data. The grain for the Weather service is one row per weather forecast per location. The forecasts are done every hour for every location. The grain for the SDE Measurements is one row per measurement. Most measurements in the SDE Measurements are done for every five minutes. The measurements are not accumulated, which means that the most atomic data from this operational source system is by one row for (nearly) every five minutes.

6.3.1.3 Dimensions

With the grains defined for the different systems and processes, the dimensions, which preferably should be conformed, can be identified: The first two dimensions relate to the fact that all the operational source systems have a date and time defined for the metrics. Hence date and time dimensions should be used. Since these dimensions can be used for all the operational source systems, these are conformed dimensions. Two dimensions are made instead of a single date and time dimension, because of performance. The HSI system can provide a dimension for the Devices it integrates, these are called items. Because of the relation between items and houses, floors, itemtypes and hardware system entities, these can be consolidated into the same Device dimension. For group entities, which have a many-to-many relation with items, a secondary dimension should be made called Grouping, this dimension should be made together with a factless fact table which can be used as a bridge table. Another dimension should be made for the state entities from the HSI system, this dimension should be called SwitchButtonDimension, since all of the states comes from switch states, i.e. boolean item types. These three dimensions can be used for the measurements done by the Data Collection system. Lastly a junk dimension needs to be made for the different attributes from the Weather service. This dimension is a junk dimension since non of the attributes (columns) have a functional meaning on the primary key. The dimension consists of a number of different low cardinality attributes, i.e. enumerations. This dimension can only be used with the measures from the Weather service. The identified dimensions can be seen in Figure 6.3.2. The DateDimension contains a number of attributes which describes the date, e.g. the quarter (CalendarQuarter), the Day, the Month and the Year. Note that all the attributes have a Name variation, this is the label which is shown to the end-users, while the ordinary attribute defines the value key of this attribute. The DateKey is the primary key and the DateAlternateKey is a DateTime object which holds the date, and can be used for

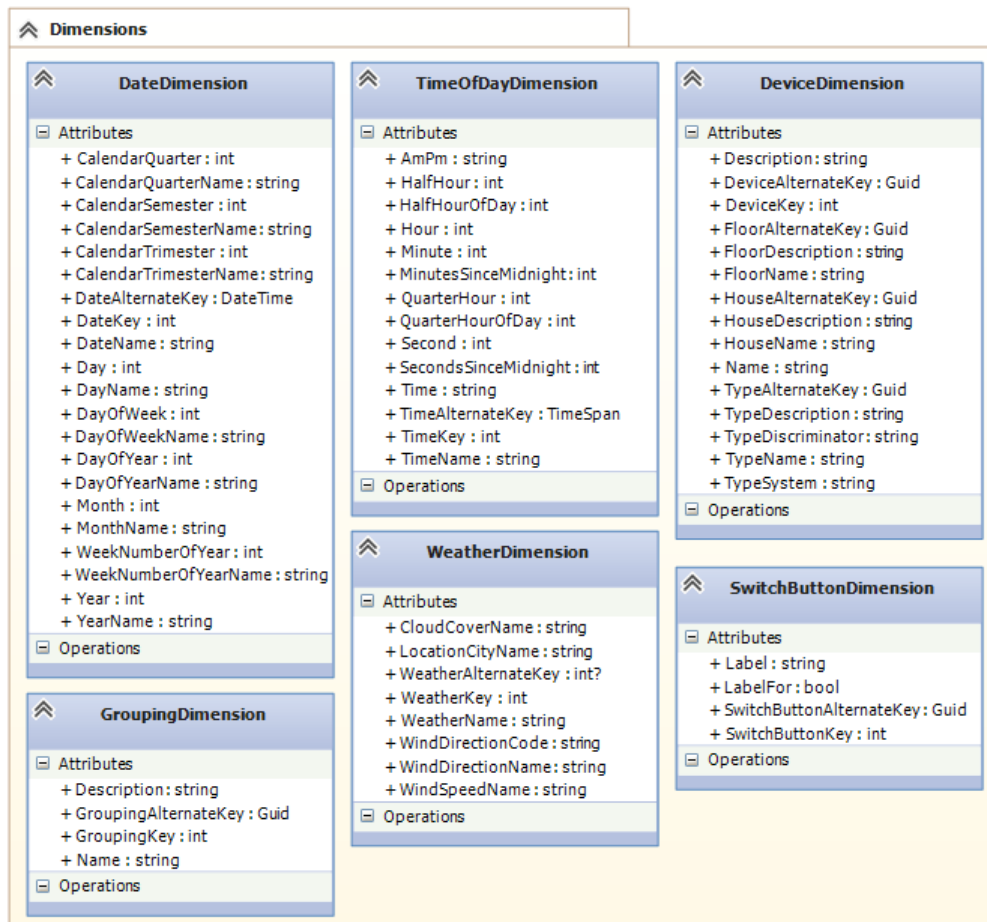


Figure 6.3.2: Class diagrams for dimensions

lookups during ETL processing (this attribute should not be used by end-users). The DateKey has a special format, which is YYYYMMDD, and is an integer, hence this key can also be derived from timestamps. The TimeOfDayDimension contains a number of attributes which describe the time of day. This includes the Hour, Minute and Seconds, as well as more analytical attributes like HalfHour and QuarterHourOfDay. Again these attributes have Name variations which are used by the end-users. The TimeKey is the primary key, and the TimeAlternateKey is a TimeSpan object which contains the time component of a timestamp. The TimeKey has a special format, which is DHHMMSS, where D is a dummy integer, e.g. 9, and the rest is the time, and hence this key can be derived from timestamps. The DeviceDimension contains the Description and Name of the HSI item, house, floor and itemtype. It also contains the TypeDiscriminator and hardware system (System) from the itemtype. The DeviceKey is the primary key, and the DeviceAlternateKey is the key used for the item from the HSI system. The alternate key for house, floor and type are also contained in the DeviceDimension. The GroupingDimension only contains a Name and Description, together with the primary key (GroupingKey) and the alternate key from the HSI system.

The `SwitchButtonDimension` holds the `Label` and the `LabelFor`, which is which boolean value the `Label` is for. It also contains the alternate key together with the `SwitchButtonKey` which is the primary key. The `WeatherDimension` contains a number of different descriptive attributes for a weather forecast. This includes the `CloudCoverName`, for the cloud condition, `LocationCityName`, for the city, `WeatherName`, for the overall description of the weather, `WindDirectionCode` and `WindDirectionName`, which indicates the wind direction and a `WindSpeedName`, which is a description of the wind speed. The `WeatherKey` is the primary key. The `WeatherAlternateKey` is bound to the `WeatherName`, and not for the `WeatherDimension`. The contents of this dimension table is combinations of the different type of enumeration values for the different attributes, and only for the combinations which exists from the weather forecasts. This means that a much smaller dimension table can be created. This also means that when new forecast combinations exists, these should be loaded together with the fact loading. All the primary keys, except for `DateDimension` and `TimeOfDayDimension`, use surrogate keys, so the primary keys have no relation to the keys in the operational source systems. The keys for the operational source systems are included as alternate keys for ETL processing only, and should never be used by the end-users.

6.3.1.4 Facts

With the dimensions identified it is now possible to identify the individual facts.

The factless fact table (`DeviceGroupFact`) for the groups from the HSI system, which is used as a bridge table, is seen in Figure 6.3.3. It is seen that the fact table has no other attributes than an `Id`, and two

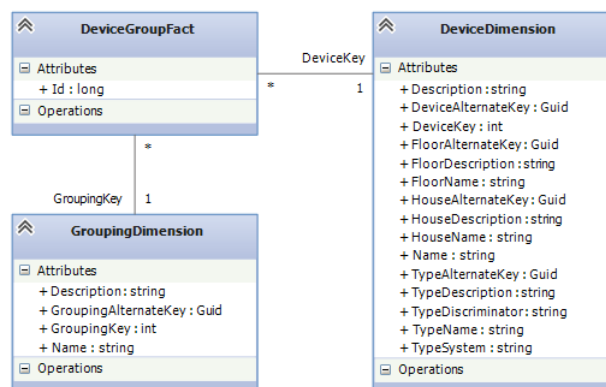


Figure 6.3.3: Class diagram for `DeviceGroupFact`

foreign keys, which bridge a `GroupingDimension` row with a `DeviceDimension` row and vice versa. This is the reason why it is referred to as a fact less fact table, since there are no measures in this fact table. Each row of the fact table is a relation between one row in each dimension. So e.g. if a device dimension row has multiple relations with a grouping dimension, each relation will be a row in the fact table. Using this fact table makes it possible to select a set of devices based on a grouping, and select a set of grouping from based on a device. I.e. it models a many-to-many relation.

For the measurements from the Data Collection service, two major types of facts are identified. These are

the numeric and boolean measurements. For the numeric measurements, the measurement from a NumericRecord row is put into the fact table for numeric facts. These facts are usually not fully additive, so a second measure can be calculated from this measurement, which is the delta, or change of the measurement, since the last measurement. For the bool measurements, the measurement from a BoolRecord row is put into the fact table for this measurement, and a Duration measure can be calculated based on this measure. The duration is the number of seconds since the last measurement occurred with the same value. When the value has changed, i.e. true to false or false to true, a second row is needed in the fact table for a duration calculation of the previous value and a new row for the current measurement, which duration is zero seconds, since the change has just occurred. A third type is also identified for numeric measurements of discrete values. Because of the discrete value it is possible to calculate a duration like for boolean facts. This fact table contains the measurements like the normal numeric fact table, i.e. measurement and measurement delta, and it includes a duration measure, which is the number of seconds the measurement value has been the same. A second row is also needed for this type when a change occurs, one with the previous value and the duration, and another with the changed measurement value and a duration of zero seconds. The three types of fact tables can be seen in Figure 6.3.4. In addition to the measures and an Id, the fact ta-

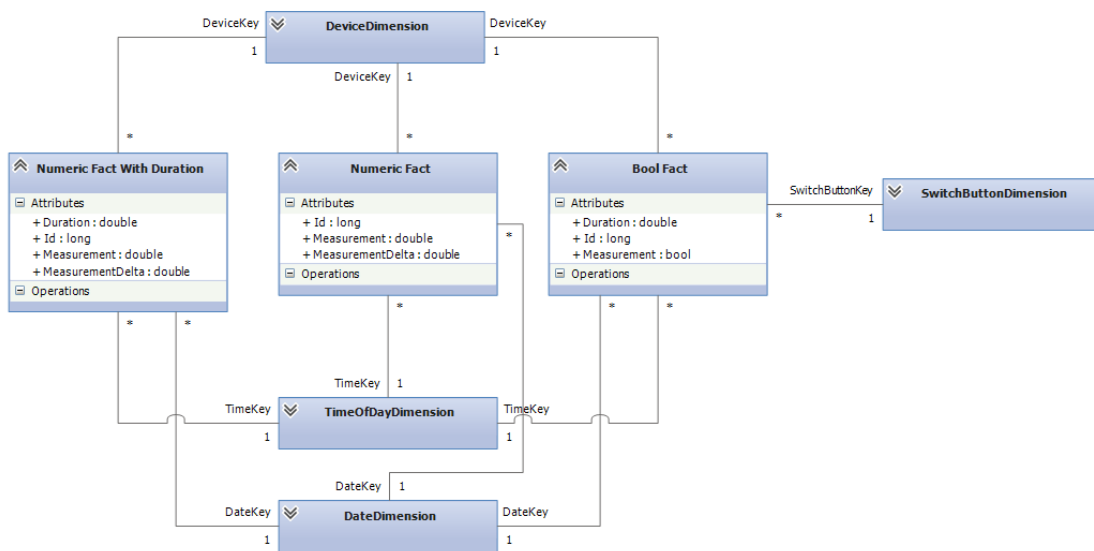


Figure 6.3.4: Class diagram for records

bles have a foreign key for the primary key of the DeviceDimension (DeviceKey), TimeOfDayDimension (TimeKey) and DateDimension (DateKey). For bool fact tables a foreign key for the SwitchButtonDimension (SwitchButtonKey) is also included. It should be noted in the figure that there is a one-to-many relationship between the fact tables and dimension tables. This is the essence of dimensional modelling, where it is possible to define a dimension table row and get a set of fact table rows out of a single query. All the facts from the Data Collection service could be modelled using just these three fact tables, this would however require a new dimension, which can differentiate the different types of measurements made from

the Data Collection service, e.g. temperature measurements and energy measurements. If these cannot be differentiated, e.g. through a new dimension, the summarised data would not give much sense. The reason why it is possible to do this, is because the measurements have the same grain. However since these measurements have significant different meaning, they should be split into several fact tables, one for each type of measurement. This also increase usability, since a "type"-dimension is not required from the beginning of every analysis. Figure 6.3.5 shows the identified bool fact tables. The fact tables corresponds to the different ItemTypeBool entities from the HSI system. Figure 6.3.6 shows the identified numeric

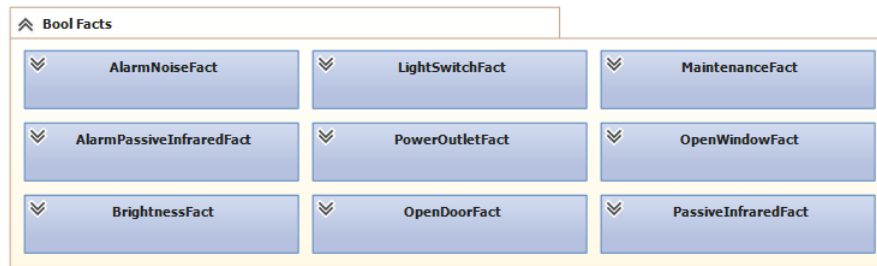


Figure 6.3.5: Bool records

fact tables. The fact tables corresponds to the subset of different ItemTypeNumeric entities from the HSI system, which have numeric and non-discrete values. Figure 6.3.7 shows the identified numeric fact ta-

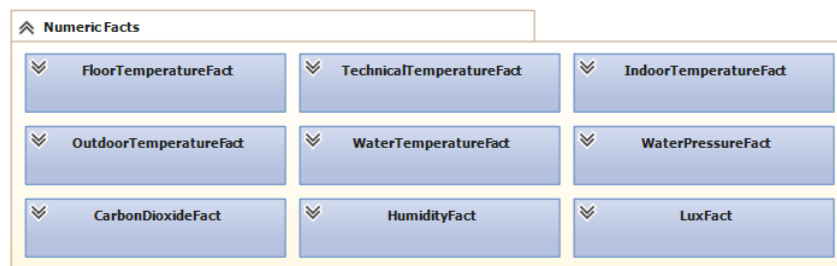


Figure 6.3.6: Numeric records

bles. The fact tables corresponds to the subset of different ItemTypeNumeric entities from the HSI system, which have numeric and discrete values. The attributes of the bool, numeric and numeric with duration facts corresponds to the attributes defined above for the different types of fact tables.

For the measurements from the weather service, the forecasts gives the following measures: A percentage of cloud cover, a relative humidity, a pressure, a rain precipitation, a snow precipitation, a temperature max, min and current value, a wind direction, and a wind speed measure. All of these measure, are non-additive, however a delta, a semi-additive measure can be calculated for these measures, except the wind direction, since this might not give much value to any analysis. The Weather fact table is seen in Figure 6.3.8. The weather fact table has foreign keys to the TimeOfDayDimension (TimeKey), DateDimension (DateKey) and WeatherDimension (WeatherKey).

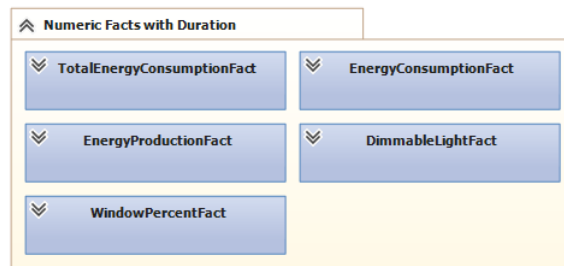


Figure 6.3.7: Numeric records with duration

For the measurements from the SDE Measurements, the following fact tables have been identified: SDE-FridgeFreezerTemperatureFact, SDEPowerFact, SDEDishwasherWashingMachineTemperatureFact, SDEAirQualityFact, SDEOvenTemperatureFact, SDERoomTemperatureFact and SDEHumidityFact. The facts and measures corresponds to the measures found in the analysis. Figure 6.3.9 shows an overview of the fact tables, and also the two foreign keys for the two dimensions for each fact table. One for the DateDimension (DateKey) and another for the TimeOfDayDimension (TimeKey). The full specification of the attributes can be found under the implementation chapter.

6.3.2 Cube

The cube of the Data Warehouse should have a one-to-one relation with the fact tables and dimension tables defined above. Since the cube can provide more calculation and since most of the facts defined above are non-additive, averages should be included for the facts. Another calculation which will help end-users is to add derived duration measures, e.g. minutes, hours and days. And the seconds measure should also be kept. This can be done by using a view. In addition to the above, for the energy, numeric fact tables, i.e. TotalEnergyConsumptionFact, EnergyConsumptionFact and EnergyProductionFact, which only includes the number of pulses and the duration. A view should be used to derive the kW and kWh measures. Note that for the TotalEnergyConsumptionFact a conversion factor of 1/1000 is used instead of 1/100, which is used for the other two facts. Another dimension which could be helpful to add in the cube is a Percent dimension for the discrete numeric values, as well as for the cloud cover.

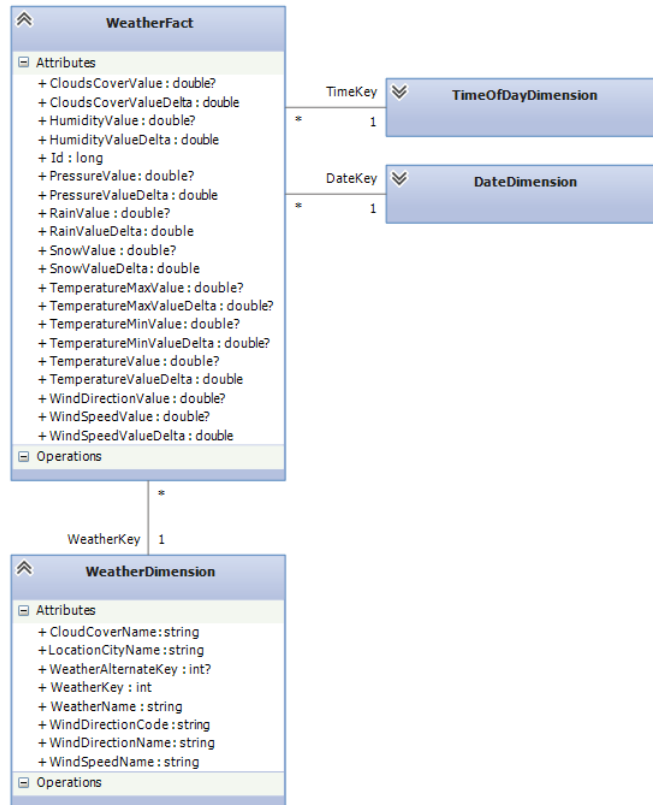


Figure 6.3.8: Class diagram for WeatherFact

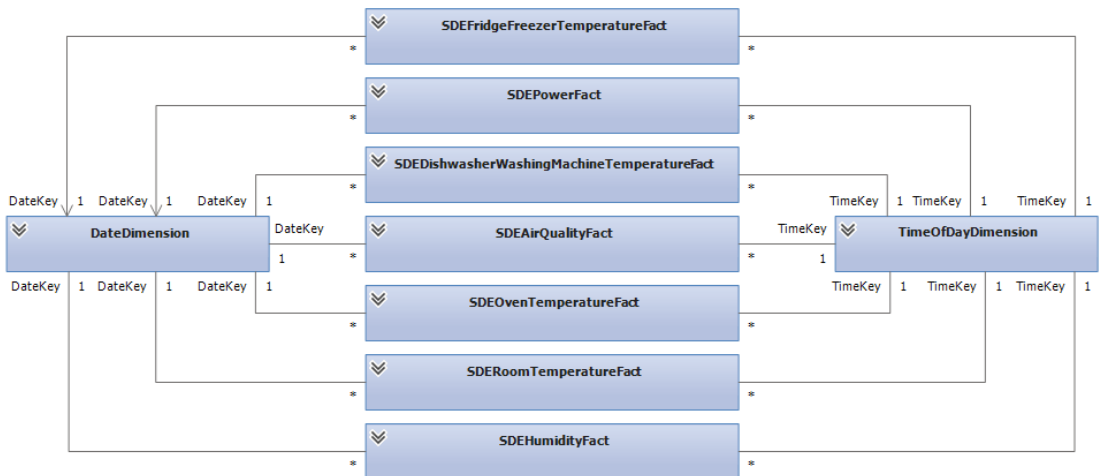


Figure 6.3.9: Classes for SDE measurement facts

6.4 ETL

The ETL component handles the extraction, transformation and load of data. The subcomponents of the ETL component can be seen in Figure 6.4.1. The main component is the Service component, which

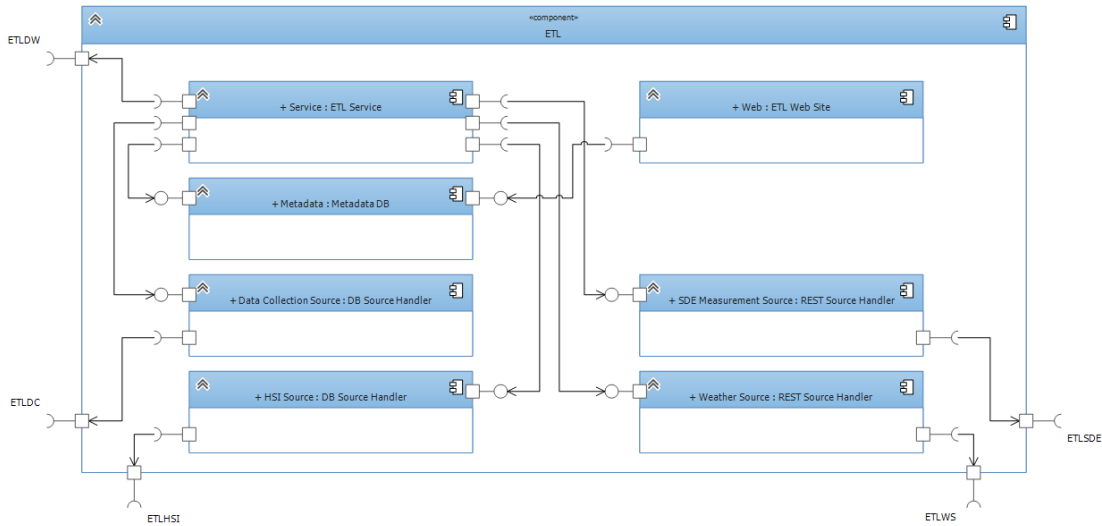


Figure 6.4.1: Subcomponents of the ETL component

handles all the processing of the data. The data comes from the four different source components (Data Collection Source, HSI Source, Weather Source and SDE Measurement Source). These components handles and consumes the sources through the ports (ETLDC and ETLHSI) for the database access through SQL, and the ports (ETLSDE and ETLWS) for the REST web services. To help the continuing processing of the sources, a Metadata component is used, which holds metadata like the latest extraction ids, performance metrics for the processing, and configurations of the processing. To configure and monitor the Service component a Web component is needed, which exposes a web site, where configuration is done using the metadata component, and the monitoring is also done through the metadata. As the data is processed by the Service component, it is loaded into the Data Warehouse using the ETLDW port, which is a SQL port.

6.4.1 Data processing

The flow of the processing of the data from the different operational source systems is illustrated in Figure 6.4.2. First all dimensions needs to be processed, since foreign keys for these are needed by the facts. The dimensions are processed in the following order: DateDimension, TimeOfDayDimension, GroupingDimension, DeviceDimension, SwitchButtonDimension. Note that the WeatherDimension is not mentioned above, this is because of the nature of the dimension, (it is a junk dimension), hence it needs to be processed together with processing of the Weather facts, where combinations of the Weather dimension rows will be generated, and inserted if necessary. Once the TimeOfDayDimension has been processed, this does

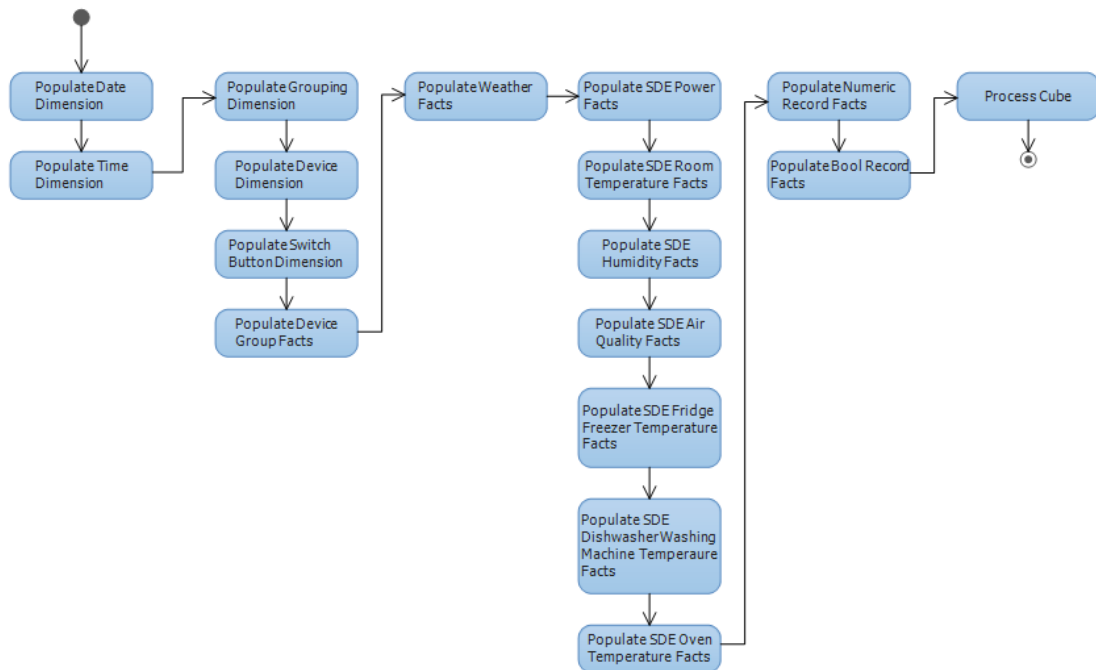


Figure 6.4.2: Processing flow for the ETL

not need any further processing in future iterations of the processing. For the DateDimension a number of year should be loaded once, and when a year has passed, i.e. the first processing at January 1st, a new year should be loaded. The processing of HSI related dimensions uses slowly changing dimension type 1 for changes, since a change of these parameters does not necessarily mean that e.g. an item is now another item, and hence a new entity from this point on. The change is most likely done because the entity was e.g. not named properly from the beginning. Dimension table rows are never deleted, since foreign keys to older rows might exist in the fact tables, changes are hence only done for every changed attributes and new rows. E.g. if a HSI item does no longer exist in the HSI system, the dimension row should not be removed. After the dimensions have been processed, the facts are processed, in the following order: DeviceGroupFact, WeatherFact, SDE facts, NumericRecord facts, Bool Record facts. All the facts except DeviceGroupFact is loaded such that only new facts are inserted and the existing facts in these fact tables are kept in the Data Warehouse, this means more efficient loading of facts, since they are continuously loaded. This is possible since all the fact tables are transaction fact tables. For the DeviceGroupFact an iteration of all the items and states from the HSI system is done, any changes are reflected in the fact table, e.g. if a relation is no longer in the HSI system, the relation is removed from the fact table. All processing of fact tables use lookup attributes (alternate keys) on the dimension tables to get the surrogate keys, which are the primary keys of the dimensions. For date and time, the primary keys are derived from the timestamp of the facts, because these primary keys have a special structure. After all these have been processed a processing of the cube is required, since new dimensions and new summarised data needs to be included

in the cube.

6.4.2 ETL application layers

The ETL application is split into three layers, which form a 3-tier architecture, from a n-tier design pattern. The layers are the presentation layer, the business logic layer, and a data access layer. The ETL application consists of the packages seen in Figure 6.4.3.

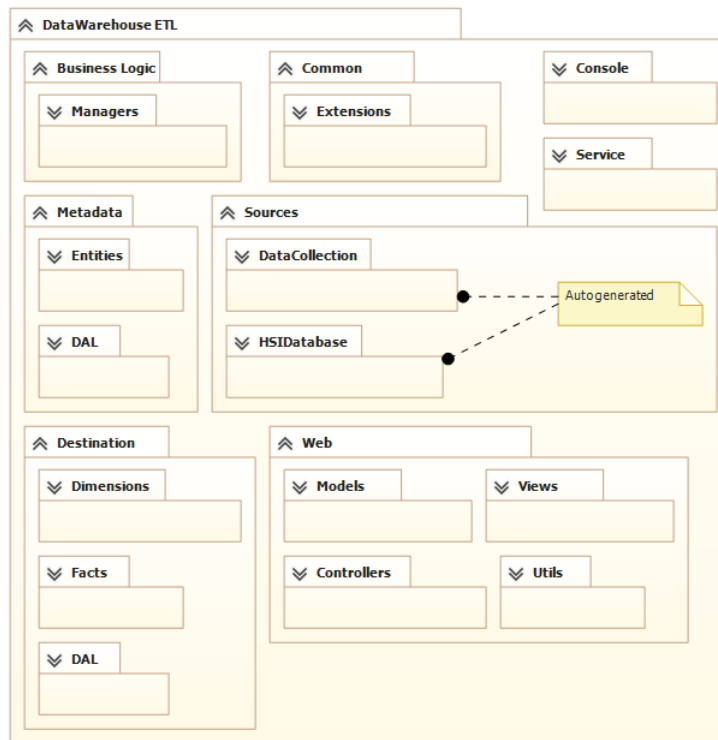


Figure 6.4.3: Package diagram for ETL

6.4.2.1 Presentation layer

The first layer is the presentation layer, which includes a service, a console and a web interface. These are seen in Figure 6.4.3. These interfaces are the end-point for the end-users. The service is special because it has no real interaction with end-users other than a start and stop method. The service has a timer, sends requests to process the data warehouse when an interval has elapsed (a tick). The console interface makes it possible for end-users to process individual steps of the overall processing, as well as a overall processing, if this is required. The individual steps of the processing is seen in Figure 6.4.2, where each column is a processing step. The web interface makes it possible for end-users to configure the ETL processing, i.e. the mapping between NumericRecord, BoolRecord fact tables and HSI items. And the weather locations

which needs to be loaded. The web interface also gives an overview of the processing by the use of metadata from the ETL processing. These metadata include how long different processing steps take, how much is loaded and whether the ETL service is currently running. The web site uses the Model View Controller paradigm, which is also a type of n-tier architecture. The views are the web interface, these are built upon a view model, and the user interaction is done through a controller. In addition to these a number of utilities are needed which can be shared across the controllers. These packages are also seen in Figure 6.4.3.

6.4.2.2 Business logic layer

The second layer is the business logic layer, which contains the manager classes which handles all the processing, these managers should have Populate and/or Process methods which can be called by the presentation layers. Separate manager classes should be used for the different types/steps of processing. To be able to get metadata and access to other data a manager base class should be used, so all managers can get the same access to the data and load the data into the data warehouse. The manager base can be inherited by the other manager classes. Commonly used extensions, which includes deriving DateKey and TimeKey is placed in a Common::Extension package. The two packages is seen in 6.4.3

6.4.2.3 Data access layer

The bottom layer of the ETL application is the data access layer (DAL). For the ETL application multiple DALs are needed, since it has access to multiple data sources. This includes a Destination DAL, which is the data warehouse, Sources DALs which are auto generated DALs of the data sources, in this case to the Data Collection and HSI Database databases. The last DAL is the data access to the Metadata. These packages can be seen in Figure 6.4.3, where they are denoted Destination, Sources and Metadata. The DAL packages includes Context classes which gives generic access to the different types of entities of the databases. For the Metadata the entities which is mapped to the data from the database is located in the Metadata::Entities package. For the Destination the entities which is mapped to the data from the database is located in the Dimensions and Facts packages. For the auto generated sources the entities are included in the DAL. The reason for using auto generated DAL for the sources, is to make the ETL application independent on the operational source system, by making a "copy" of the entities and DALs from the operational source systems based on the databases. This makes it possible to operate the ETL application even if new columns or tables are added to the operational source systems, however if tables or columns are removed or the new columns or tables needs to be added a new auto generation of the entities and DALs is needed, which also makes the overall maintenance of the sources in the ETL application much easier.

6.5 BI Application

The BI component/application consists of a service component and a web site component. The subcomponents of the BI component is seen in Figure 6.5.1. The BI application uses the OLAP port (denoted

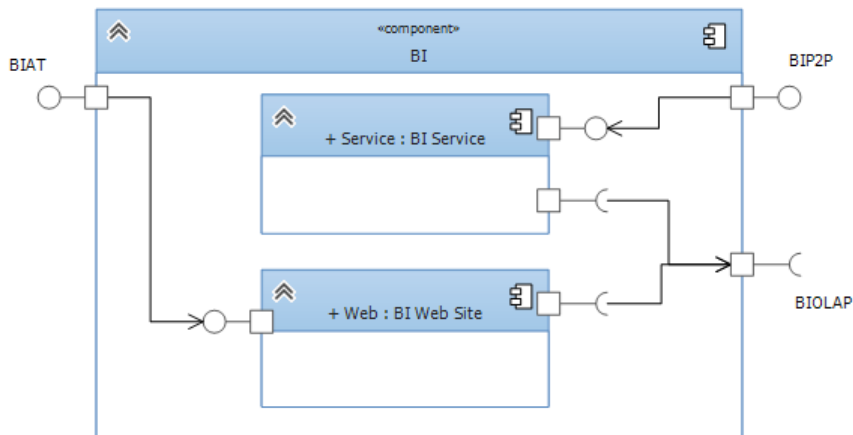


Figure 6.5.1: Subcomponents of the BI component

BIOLAP) from the Data Warehouse, which is a proprietary port, which means that the BI application needs to use OLAP vendor specific libraries to interact with the Data Warehouse. To be able to use the Data Warehouse OLAP port from the internet a relay/proxy needs to be included in the Web component, this relays all communication between the BIAT and BIOLAP ports. The BIAT port is used by Analysis tools. Another client/analysis tool is the service, which also consumes the BIOLAP port. This service component makes it possible to use P2P messages to query and generate reports from the data warehouse, through the BIP2P port. The service is in an basic analysis tool, because it can generate reports based on the queries made to the data warehouse.

6.5.1 Data model

The data model for the BI application is seen in Figure 6.5.2, where the Requests and Responses packages are the P2P messages, which the service subscribes and publishes, respectively. The GetDimensionRequest and GetMeasuresRequest does not have any parameters, and the respective GetDimensionsResponse and GetMeasuresResponse contains all the dimensions and measures, respectively. Included in the Response messages are also an ExceptionMessage field which is populated with an error message, if the request was not handled successfully. A derived boolean field Exception indicates whether a response contains an exception message or a valid response is provided. Included in a GetDimensionResponse is a list of Dimension objects, the Dimension class shows the attributes of a dimension, the UniqueName field is used as a parameter in the GetInsightReportRequest. For the BI service a dimension is flatten and hence hierarchies and level of a hierarchy are set to be a "dimension". To get the hierarchy the ParentUniqueName and ParentDimension refer to the parent "dimension". For the lowest level of a dimension

hierarchy the field `IsLevel` is set to true. The parent for a level is a hierarchy and a parent to a hierarchy is a "real" dimension, i.e. a dimension table. To provide a complete list of dimensions a "ROOT" parent dimension is provided, where the child dimensions have a `ParentUniqueName` which is ROOT, however a ROOT dimension should not be included in the list of dimensions in a `GetDimensionsResponse`, since this is not a valid dimension. The `MeasureGroupNames` is a list of the valid measure groups that the dimension is mapped to. If a measure is chosen, which is part of a measure group, which is not in the `MeasureGroupNames` of a dimension, the resulting report will include the dimension, but the measures (metrics of a fact) will not be split properly between the rows of a dimension. Another interesting field of the `Dimension` class is the `IsNamedSet`. A named set is set of dimension rows, which can e.g. be used to filter queries. The named sets are defined in the OLAP cube. Another option which is defined in the OLAP cube is a display folder, which can be used to split dimensions into different folders, this is usually only used for larger DW/BI solutions. Included in the `GetMeasuresResponse` is a list of `Measure` objects. The `Measure` class includes a `UniqueName` which is used as a parameter in the `GetInsightReportRequest`. A `FormatString` is provided to format the value of a report, this format string is defined in the OLAP cube, a `Units` and `DataType` is also defined in the OLAP cube, which can be used when rendering the values from a report. The `Measure` class also have a `MeasureGroupName`, which is the measure group that the measure belongs to. The two display name fields, i.e. `DisplayName` and `ParentDisplayMeasureGroup` are user-friendly string which can be shown to the end-user. The `GetInsightReportRequest` takes a number of dimension and measure unique names. The `FilterUniqueNames` is a list of dimension unique names, this parameter is convenient when e.g. using named sets. The `MeasureUniqueNames` is a list of measure unique names. When multiple measures are selected, the different measures are split into two axis, one for the measure and one for the different types of measure. The `PrimaryDimensionUniqueNames` is a list of dimension unique names, where the measures should only be split between one axis of dimension rows. The `SecondaryDimensionUniqueNames` is a list of dimension unique names, where the measure should be split into two axes of dimension rows, e.g. one for the primary dimension and one for the secondary. When using the primary dimensions, the axis are flattened by making a cross join. When using multiple measures it is not possible to use the secondary dimension parameter, since this would cause multiple axes to be retrieved when querying. The secondary dimension is also called column dimensions, because using this type, columns of dimension table rows are made. Whereas for the primary dimensions only rows are made. And the measures are on the columns. When using a measure and a column dimension the two are concatenated, i.e. the column label will be "measure name - dimension row name". The `GetInsightReportResponse` includes an `Insight` object, which is the insight report. The report includes the titles of the row and column, i.e. x- and y-axis titles, respectively. Included is also the labels for the the individual rows and columns. For the actual summarised data, a list of `InsightCells` objects are included in the `Insight` object. This class includes a single x-axis and a list of yValues of the x-axis. The x-axis is dimension rows. While the yValues are the measure, multiple yValues occur when multiple measures are selected or secondary dimensions are used. Note that the xValues are strings, this is because the OLAP cube controls the order of the x-axis. To provide an ordering for the x-axis to the analysis tool, the ordinal value (`xOrdinal`) is provided as an integer value. Also note that yValues are nullable double values, this is because an OLAP

cube supports empty values for summarised data, e.g. for when there is no data (measures) available for a dimension row.

6.5.2 BI application layers

The BI application is split into two layers, a presentation layer, which consists of a service and a web site. And a business logic layer, which includes a manager class which handles the querying and generation of reports. The application classes are seen in Figure 6.5.3.

6.5.2.1 Presentation layer

The service has its own P2P broker, which is used to get callbacks to the `SendResponse` methods seen in Figure 6.5.3. The `SendResponse` methods call the respective methods in the manager. A web site is also included in the BI application, this web site is mainly for the relay of the OLAP cube, but it is also used to get access to the OLAP cube, e.g. by providing user names and passwords for the cube, and for quick guides to get end-users started with using the OLAP cube. The web site uses the Model View Controller paradigm, to make the web site more modular and extendible.

6.5.2.2 Business logic layer

The business logic package seen in Figure 6.5.3 shows the BI manager class, which is used to handle the requests from the service. The manager contains a connection to OLAP cube (denote `_conn`) and a field which defines which cube or perspective to use. This field is populated by a configuration file, together with a connection string to get the `_conn` connection. A perspective is a subset of a cube. The cube consists of measures and dimensions. The primary methods of the manager class are `GetDimensions`, `GetMeasures` and `GetInsightReport`. These are the methods called by the presentation layer. They provide the list of dimensions and measures of the cube. The `GetInsightReport` calls the `GenerateQuery` based on the parameters sent, and afterwards it calls the `GetQueryResult`, which is then processed to fit into the `Insight` and `InsightCells` objects, and return back to the presentation layer, which sends these as response messages. Additional methods are provided by the BI manager, including `GetDimensionByUniqueName` and `GetMeasureByUniqueName`, which returns a single dimension or measure, respectively. The `GetMeasureGroup` gets the measure group based on a unique name for a measure. To get all measure groups the `GetMeasureGroups` method can be used. Lastly the `ProcessDatabase` method enables the manager to re-process the OLAP cube, this can be helpful if the newest data is still not available in the cube. However this is also done periodically together with the ETL processing, so the need for this method is only provided because the BI manager is used to query the OLAP cube, and the processing of the cube is also a query. All the methods are public since there might be a need to use these methods outside the scope of the manager class.

6.5.3 App Service

One of the consumers of the BI service is the App Service, which is used as a relay to be able to use services inside the P2P network from the internet. This is done through a REST web service. The App service uses the BI service to relay lists of dimensions and measures and relays report requests to the BI service. The requests and responses from the BI application have one-to-one relation with the requests and responses done through REST web service. The App service is used by a tablet application and can be used by any device which can handle REST web services.

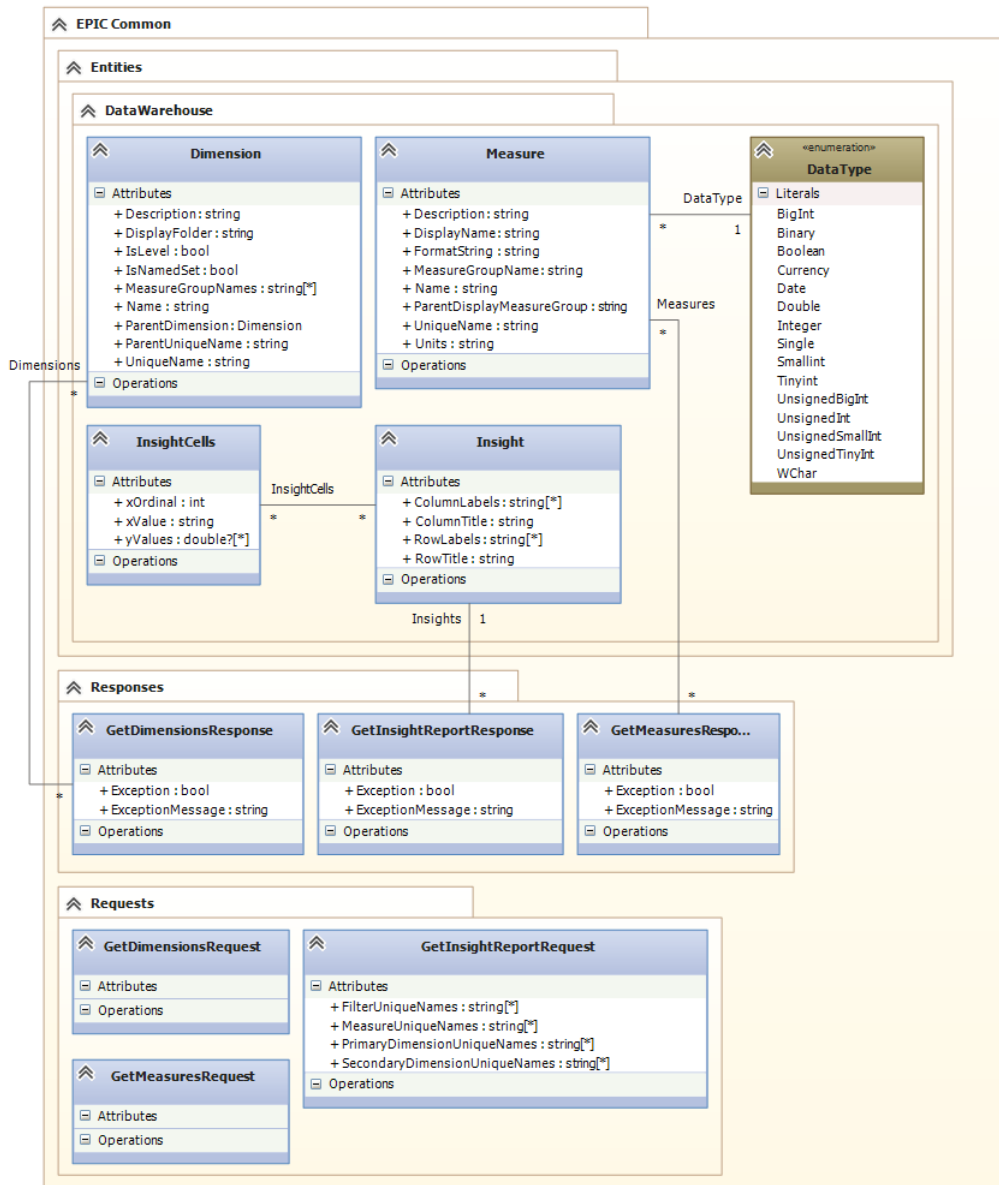


Figure 6.5.2: Data model for BI application

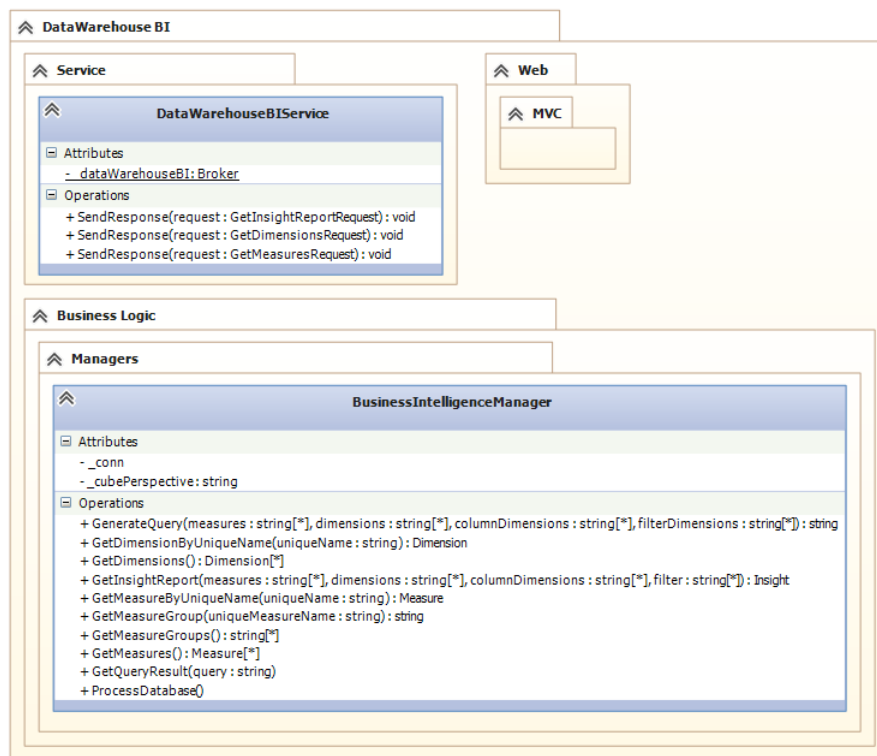


Figure 6.5.3: Class diagram for BI application

Chapter 7

Implementation

”There are only two kinds of languages: the ones people complain about and the ones nobody uses”

—Bjarne Stroustrup, 2007

Contents

7.1	Details on Data Collection	99
7.2	Details on Data Warehouse	103
7.3	Details on ETL	112
7.4	Details on BI application	129

This chapter provides the specific details on the implementation of the DW/BI solution.

7.1 Details on Data Collection

The implementation of the Data Collection service application is done using the same data model defined in the design and most of the methods from the design. The implementation is done using C#.Net and the Entity Framework (EF) is used for Code-First database creation, which makes it possible to make classes and generate a database schema based on the classes, which are called entities. The EF also makes it possible to make a transparent DAL, where the instances of the entities are available through sets, called IDbSet. These sets have a number of basic operations, which makes it possible to do Create, Read, Update and Delete (CRUD) operations on the entities. These methods are called Add, Update and Remove. Reading is done by finding the instance of the entity. Finding instances is possible through e.g. the use of LINQ, which is a programmatic querying language, which resembles SQL a little. The final class diagram of the implementation is seen in Figure 7.1.1. The services are made as Windows Services, these requires an OnStart and OnStop method, which are called, when the service is starting and stopping, respectively. The OnStop method for the two services are not used, since no clean up is necessary for these services. Additionally a service installer is also generated for the Windows Service, which is used to define the service, which is used when the service is installed. This includes start type, which for both services are set to automatic, meaning the services will start with the server start. The Windows Account is set to use

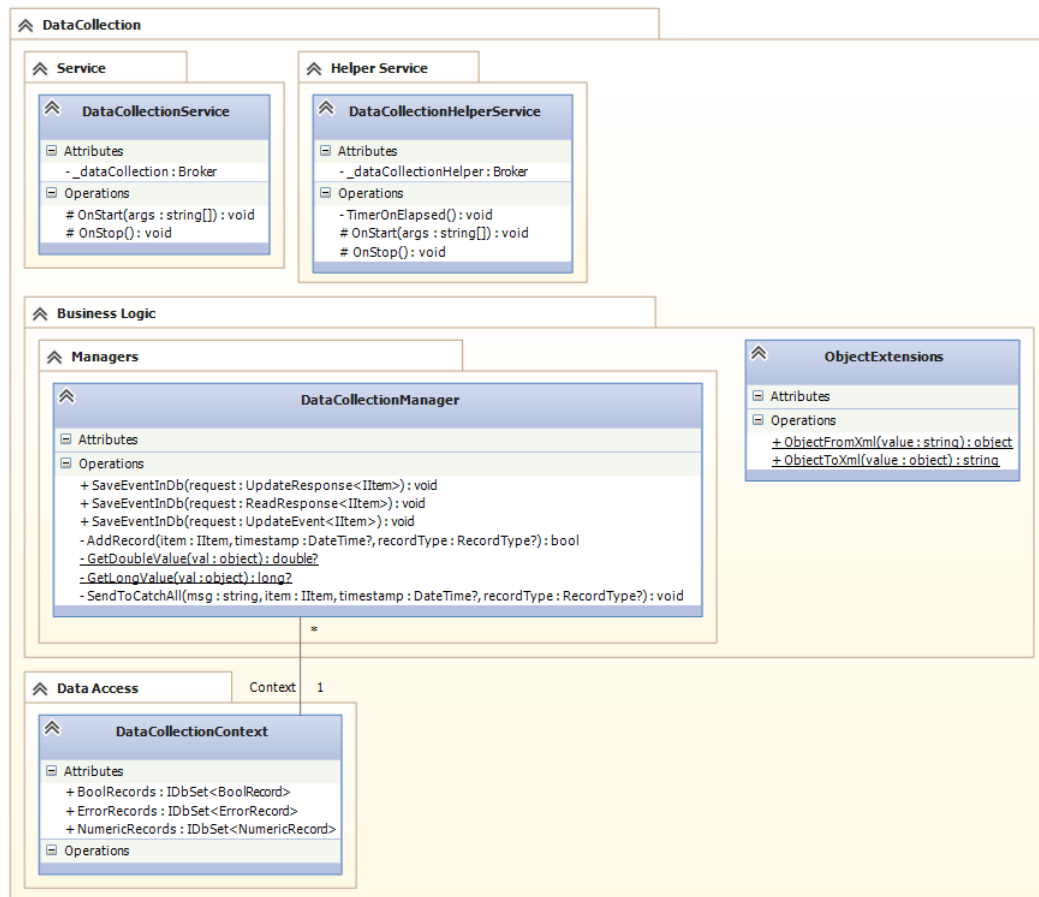


Figure 7.1.1: Class diagram for Data Collection

User, which means a Windows user account is needed when installing the service, this user account will be used when interacting with the database, where access rights should be given to this user account.

7.1.1 DataCollectionService

The DataCollectionService is a separate windows service.

The OnStart method in the DataCollectionService instantiates a P2P broker and subscribes to the response messages provided by the HSI system. All the messages, which are sent in the P2P network is included in a EPIC.Common NuGet package, which makes it possible to have different versions of the common messages. However the latest package should be used for the best compatibility with the other systems in the network.

Anonymous callback methods are used for the subscribed messages, which instantiates a new DataCollectionManager and calls the respective SaveEventInDb method.

For numeric records the following code is used to determine whether a LongValue can be set on a NumericRecord. It checks whether the epsilon (also called error) of the double value is below a threshold, if it is, it is considered a long value. The code for this is seen in Listing 7.1.1.

```

1 var isLong = false;
2 if (doubleVal.HasValue && longVal.HasValue)
3 {
4     var epsi = (Math.Abs(doubleVal.Value - longVal.Value));
5     if (epsi < 0.01)
6     {
7         isLong = true;
8     }
9 }
10
11 if (!isLong)
12     longVal = null;

```

Listing 7.1.1: Determination of long or double

The DoubleValue of a NumericRecord is always set. Since the value in the response message is an object, the doubleVal and longVal are set using a type conversion using the GetDoubleValue and GetLongValue methods, which calls Convert.ToDouble or Convert.ToInt64, respectively, and returns null, if the conversion could not be done. If a response message cannot be saved properly, e.g. if the Value of the HSI Item is another type than a numeric or bool, the SendToCatchAll is called, which saves an ErrorRecord. This method uses the ObjectToXml from the ObjectExtensions class, to serialise the value, to keep the type and value intact in the XmlValue field of the ErrorRecord. Filtering is done for both BoolRecord and NumericRecords, but only for UpdateEvent messages. The code for the filtering for BoolRecords is seen in Listing 7.1.2.

```

1 if (recordType == RecordType.UpdateEvent)
2 {
3     var lastestRecord = Context.BoolRecords.Where(x => x.ItemId ==
4         item.Id).OrderByDescending(y => y.Sent).Take(1).
5         SingleOrDefault();
6     if (lastestRecord != null)
7     {
8         if (lastestRecord.Type == RecordType.UpdateEvent &&
9             lastestRecord.BoolValue == boolVal)
10        {
11            if (timestamp.Value.Subtract(lastestRecord.Sent).
12                TotalSeconds < 1d)
13            {
14                // if bool is the same and the last record was from
15                // less than a second ago, discard the record..
16                return false;
17            }
18        }
19    }
20 }

```

Listing 7.1.2: Bool record filtering

The filtering for the NumericRecords are done slightly different because of the double value a threshold needs to be used, to check whether the value was already recorded. The code for the filtering is seen in Listing 7.1.3.

```

1  if (recordType == RecordType.UpdateEvent)
2  {
3      var lastestRecord = Context.NumericRecords.Where(x => x.ItemId ==
         item.Id).OrderByDescending(y => y.Sent).Take(1).
         SingleOrDefault();
4      if (lastestRecord != null)
5      {
6          if (lastestRecord.Type == RecordType.UpdateEvent &&
             lastestRecord.DoubleValue.HasValue && doubleVal.HasValue
             && Math.Abs(lastestRecord.DoubleValue.Value - doubleVal.
             Value) < 0.001)
7          {
8              if (timestamp.Value.Subtract(lastestRecord.Sent).
                 TotalSeconds < 1d)
9              {
10                 // if double is the same and the last record was from
11                 // less than a second ago, discard the record..
12                 return false;
13             }
14         }
15     }

```

Listing 7.1.3: Numeric record filtering

For numeric records the number of pulses for the pulse counters are also filtered, if the number of pulses is zero, the record is discarded, because these responses are useless, and would require additional filtering in the ETL, if not done in the Data Collection. This is a special opportunity which is only possible, when an operational source system needs to be made together with a DW/BI solution. Usually all the operational source systems are pre-existing systems. But since this operational source system will not be used for any other purposes other than the DW/BI solution, it is possible to make implementation decisions like this.

7.1.2 DataCollectionHelperService

The DataCollectionHelperService is a separate Windows Service. The OnStart method instantiates a P2P broker which is used for the entire operation of the service, and sets up a Timer which is initially set to 30 seconds. On the timer tick the TimerOnElapsed method is called. AutoReset for the timer is set to false, because the timer should not send another tick after the first 30 seconds. The timer should be enabled after a tick has completed. The TimerOnElapsed method runs the sequence seen in Figures 6.2.4 and 6.2.5 in the Design chapter. After a tick is complete the timer is set to a interval of 5 minutes, which means another tick is executed after 5 minutes, again the AutoReset is set to false, so if the TimerOnElapsed takes more than 5 minutes, another tick is not executed.

7.2 Details on Data Warehouse

The data warehouse components are split into two different parts, the database part is moved into the ETL solution, while the OLAP cube part is moved into the BI solution. This also means that the entire DW/BI solution is split into three solutions, a DataCollection solution, an ETL solution and a BI solution.

7.2.1 Data Warehouse database

To minimise redundancies the database using the RDBMS component from the design is moved into the ETL solution, where EF Code-First is used to generate the database schema. The resulting entities from the data model is seen in Appendix D.1. To enforce referential integrity between dimension tables and fact tables, references to the dimension entities are made from the fact entities. Since the foreign key naming is different for the design than what is default for EF Code-First, which is "referred entity name_Id", the ForeignKey attribute is used to decorate the references to the dimension entities and a specified foreign key property is introduced. An example is given in Listing 7.2.1.

```
1 [Table("FactDimmableLight")]
2 public class DimmableLightFact
3 {
4     public long Id { get; set; }
5
6     // dimension keys
7     [ForeignKey("DateKey")]
8     public DateDimension Date { get; set; }
9     public int DateKey { get; set; }
10
11     [ForeignKey("TimeKey")]
12     public TimeOfDayDimension Time { get; set; }
13     public int TimeKey { get; set; }
14
15     [ForeignKey("DeviceKey")]
16     public DeviceDimension Device { get; set; }
17     public int DeviceKey { get; set; }
18
19     // facts
20     public double Measurement { get; set; }
21     public double MeasurementDelta { get; set; }
22
23     public double Duration { get; set; }
24 }
```

Listing 7.2.1: Example of fact entity

EF Code-First requires all entities to have explicit Id-fields, which are auto-incremented. The naming and auto incrementation of the Id, should not be used for e.g. the DateDimension where the primary key has the special format YYYYMMDD. To override this a Key and DatabaseGenerated attributes can be decorated on the primary key field. The names of the entities follow a regular naming convention for entities, however a different naming convention is used for dimension and fact tables in data warehouse databases, these

are usually called DimName and FactName, respectively, where the Name is the name, e.g. Date. Hence the DateDimension entity should be named DimDate in the database and the DimmableLightFact entity should be named FactDimmableLight in the data base. To override this the Table attribute is decorated above the entity class. An example of the DateDimension is given in Listing 7.2.2

```

1 [Table("DimDate")]
2 public class DateDimension
3 {
4     [Key]
5     [DatabaseGenerated(DatabaseGeneratedOption.None)]
6     public int DateKey { get; set; }
7     public DateTime DateAlternateKey { get; set; }
8     public int Year { get; set; }
9     public string YearName { get; set; }
10    public int Month { get; set; }
11    public string MonthName { get; set; }
12    public int Day { get; set; }
13    public string DayName { get; set; }
14    public int DayOfWeek { get; set; }
15    public string DayOfWeekName { get; set; }
16    public int DayOfYear { get; set; }
17    public string DayOfYearName { get; set; }
18    public int WeekNumberOfYear { get; set; }
19    public string WeekNumberOfYearName { get; set; }
20    public int CalendarQuarter { get; set; }
21    public string CalendarQuarterName { get; set; }
22    public int CalendarSemester { get; set; }
23    public string CalendarSemesterName { get; set; }
24    public int CalendarTrimester { get; set; }
25    public string CalendarTrimesterName { get; set; }
26    public string DateName { get; set; }
27 }

```

Listing 7.2.2: Example of dimension entity

The RDBMS system used for this DW/BI solution is Microsoft SQL Server 2014.

7.2.2 SSAS cube

The OLAP component is put into the BI solution, because the OLAP cube contains BI specific calculations. The OLAP system used for this DW/BI solution is Microsoft SQL Server 2014 Analysis Services (SSAS). The SSAS system contains Data Sources, Data Source Views, Cubes, Dimensions, Mining Structures, Roles, Assemblies and Miscellaneous. The Data Sources are the sources of the data, in this solution the only data source is the DataWarehouse database. The Data Source Views are different views for all the tables available from the data sources. For this solution, only a single data source view is used. The Cubes are the different cubes in the OLAP system, the cubes define which dimensions and measures are used, how they relate through measure groups. The cube also contains a set of calculations and named sets. KPIs are also

defined for the cubes. Additionally actions, partitions, aggregations, perspectives, and translations are also defined in the cube. The actions are different actions possible through the analysis tools, this includes drill through actions, where the columns of the drilled through data can be specified. This feature is not used in this solution. The partitions can be used to defined logical parts of the data, this feature is not used in this solution. The aggregations are used to define special types of aggregations, e.g. for optimising precalculated summaries, this feature is not used in this solution. Perspectives makes it possible to make subsets of the cube available for end-users. A perspective is used for the BI application, such that it is possible to control which dimensions and measures area available in the BI application and thereby in the App Service. Translations are used for translating elements of the cube, e.g. the dimensions and measures, this could be used if multilingual support is needed, however this is not needed in this solution. The Dimensions define all the dimensions available for the cubes. A dimension is defined by selecting a number of attributes, the attributes can have NameColumn and KeyColumn defined if these are not the same. If special ordering is needed a OrderBy and OrderByAttribute is available. In the dimension attribute relationships are also defined, i.e. for hierarchies. When the attribute relationships are defined, hierarchies can be defined, which are shown in the analysis tools. The Mining Structures are for data mining, this feature is not used for this solution. The Roles are used to define specific access right, using roles it is possible to filter the cube of data for specific roles, which users can have membership to. An External User role is included in this solution, to be able to define which data is available to external users. The Assemblies are used if custom code is needed in the cube. This is not used for this solution. The Miscellaneous is for miscellaneous files, which is not used for this solution either.

7.2.2.1 Data Source View

Most of the cube is made with a one-to-one relation to the fact and dimension tables which are in the data warehouse database.

A new dimension is however introduced, which is a Percent dimension, called DimPercent, which is based on a view with the SQL seen in Listing 7.2.3. The resulting view gives a Percent column and a PercentName column. The Percent column contains the interger values from zero to 100, and is used as the primary key. The PercentName is the value of the Percent column concatenated with a percent sign.

```
1 SELECT
2 CAST(number AS int) as [Percent], CAST(number AS varchar) + '%' as [
   PercentName]
3 FROM master .. spt_values
4 WHERE type='p' AND number between 0 and 100
```

Listing 7.2.3: Example of basic MDX query

The Percent dimension is used with the FactWeather and FactDimmableLight tables. For FactDimmableLight a foreign key is added to the Percent dimension with the value from the Measurement column of the fact table, the foreign key is called MeasurementPercentFK. For FactWeather a foreign key is added to the Percent dimension with the value from the CloudsCoverValue column of the fact table, the foreign key is called CloudsCoverPercentFK. For both tables the foreign key is derived by casting the value to an integer,

this truncates any decimal points, and corresponds to a primary key in the Percent dimension. The Percent dimension is introduced in the OLAP cube to test whether this dimension can be candidate to a proper dimension, which can be used for more fact tables. This is easily accomplished in a SSAS cube, where views and derived columns (called Named Calculations) can be added to the Data Source View.

The fact tables FactEnergyConsumption, FactTotalEnergyConsumption and FactEnergyProduction are converted to views, because the contents of these tables is a little different than the other fact tables. The Measurement column is a count of the pulses, which will most likely not be used for analysis, instead derived measurements are needed to get kW and kWh measurements out. In addition to this the fact table contains a zero count with a duration of zero, because the normal ETL processing is used where the duration is calculated. These zero counts should not be included in the data which is provided by the cube. Hence the view based on the SQL listed in Listing 7.2.4, 7.2.5 and 7.2.6 are used.

```

1 SELECT      Id , DateKey , TimeKey , DeviceKey , Measurement ,
      MeasurementDelta , Duration , Measurement / 100 AS MeasurementAskWh ,
      ( Measurement / 100 ) / ( Duration / 3600 )
2                                     AS MeasurementAskW
3 FROM        FactEnergyProduction
4 WHERE       ( Duration > 0 )

```

Listing 7.2.4: SQL view for FactEnergyProduction

```

1 SELECT      Id , DateKey , TimeKey , DeviceKey , Measurement ,
      MeasurementDelta , Duration , Measurement / 100 AS MeasurementAskWh ,
      ( Measurement / 100 ) / ( Duration / 3600 )
2                                     AS MeasurementAskW
3 FROM        FactEnergyConsumption
4 WHERE       ( Duration > 0 )

```

Listing 7.2.5: SQL view for FactEnergyConsumption

```

1 SELECT      Id , DateKey , TimeKey , DeviceKey , Measurement ,
      MeasurementDelta , Duration , Measurement / 1000 AS MeasurementAskWh
      , ( Measurement / 1000 )
2                                     / ( Duration / 3600 ) AS MeasurementAskW
3 FROM        FactTotalEnergyConsumption
4 WHERE       ( Duration > 0 )

```

Listing 7.2.6: SQL view for FactTotalEnergyConsumption

Derived columns/Named Calculations are also added for all the fact tables which have a Duration column. Since this column contains the duration in seconds, a named calculation is done to get the duration in minutes, hours and days. The named calculations are called DurationInMinutes, DurationInHours and DurationInDays. The DurationInMinutes is derived by: $[Duration]/60$. The DurationInHours is derived by: $([Duration]/60)/60$. And the DurationInDays is derived by: $(([Duration]/60)/60)/24$.

Because of the foreign keys constrain already in the database, all relationships are automatically generated in the data source view. This is one of the big benefits of having a proper ETL layer, instead of having the ETL done inside the data source view.

7.2.2.2 Cube

A single cube is defined in the SSAS system.

All the fact tables are made into a measure group, the measures defined for the measure groups if the sum of the metric columns and a count of all the rows of the fact table. For the SDE measurements the count is replaced by a count of the non-empty rows for each of the metric columns, except the Delta columns. This is done because the SDE Measurements can have null values, and to calculate proper summarised data, the count should be of the non-empty rows.

The counts and sums are used to make average calculations. The calculations are defined in a calculation script, which is a MDX script, that generates MDX members. The averages are made for the durations and measurement from the Numeric and Bool Record facts. For the energy consumption, production and total consumption, the average is done for the kW and kWh measures. For the Weather measures averages are calculated for all. The same is done for the SDE Measurements. In addition to the average calculations, named sets are defined for the Date and Time dimensions, e.g. for Today, Last month, etc. And for time e.g. Last hour, Evening, etc.. An excerpt of the calculation script can be seen in Appendix C.1.

An example of a calculated average using the calculations in the cube is seen in Listing 7.2.7. It is seen that a member is created under the measures member of the current cube. The measure is called Dimmable Light Average, and has the unique name: [Measures].[Dimmable Light Average]. It is calculated using [Measures].[Dimmable Light Measurement] / [Measures].[Dimmable Light Count], which is the sum divided by the count.

```

1 CREATE MEMBER CURRENTCUBE.[Measures].[Dimmable Light Average]
2 AS [Measures].[Dimmable Light Measurement] / [Measures].[Dimmable
   Light Count],
3 VISIBLE = 1 , ASSOCIATED_MEASURE_GROUP = 'Dimmable Light';

```

Listing 7.2.7: Calculating average using MDX

Two examples of named sets using the calculations in the cube is seen in Listing 7.2.8. It is seen that named set called "Date - Last 6 Months" is created in the current cube member. The set is defined by a string, because it uses the current time (NOW()) to get the year and month, hence the STRTOMEMBER method needs to be used. The STRTOMEMBER gives the current month member, to get the last 6 months, the LAG method is used, this gives the month member which is 6 members behind the current member. The last six months does not include this month, hence the set should end at the previous month member, which is why the LAG(1) is used at the end. The '+FORMAT(Now(), 'yyyy')+ ' gives the month key, which is used to get the current month member. A simpler set is defined for the Time - Evening, where the hour key 18 to 21 is used to define the named set.

```

1 CREATE DYNAMIC SET CURRENTCUBE.[Date - Last 6 Months]
2 AS STRTOMEMBER(' [Date].[Month Calendar].[Month].&['+FORMAT(Now(), '
   yyyy')+']&['+FORMAT(Now(), '%M')+']') .LAG(6) :STRTOMEMBER(' [Date

```

```

    ].[ Month Calendar ].[ Month ].&[ '+FORMAT(Now() , 'yyyy ') + ' ]&[ '+FORMAT
    (Now() , '%M' ) + ' ]' ) .LAG(1) ;
3 CREATE DYNAMIC SET CURRENTCUBE.[ Time - Evening ]
4 AS [ Time ].[ Full Day ].[ Hour ].&[ 18 ]:[ Time ].[ Full Day ].[ Hour ].&[ 21 ] ;

```

Listing 7.2.8: MDX for sets

The relation between the measure groups and dimensions are defined in the Dimension Usage. For the Numeric and Bool Record facts the dimensions Date, Device, Device Grouping and Time is defined, for the Bool Record facts the State dimension is also used. The State dimension is the SwitchButtonDimension, it is called State to give a more clear meaning of the dimension. For the Weather facts the Date, Percent, Time and Weather dimension is defined. For the SDE Measurements only the Date and Time dimensions are defined. For the Date dimension the primary key DateKey is mapped to the fact tables foreign key named DateKey, this type of relation is called a Regular type in SSAS. A regular type is also used for the Device, where the DeviceKey (primary and foreign key) is used. The same for Time where the TimeKey (primary and foreign key) is used. As well as the State for the Bool Record facts, where the key SwitchButtonKey (primary and foreign key) is used. A Many-to-Many relationship type is defined for the Device Grouping dimension. This type of relation uses an intermediate measure group, which in this case it the DeviceGroupFact. For the Device Group measure group uses the Device and Device Grouping dimensions, which is defined using a regular type, with the DeviceKey (primary and foreign key) and GroupingKey (primary and foreign key) is used. By using this intermediate measure group it is possible for SSAS to find groups based on the device, and devices based on the group. The Dimension Usage is seen in Figure C.2.1, C.2.2, C.2.3 and C.2.4 in the Appendix C.2.

A single KPI is defined in the SSAS cube, which is the "Energy Production vs Energy Consumption KPI". This KPI is defined by the value $[Measures].[Energy\ Production\ Measurement\ As\ kWh] / [Measures].[Total\ Energy\ Consumption\ Measurement\ As\ kWh]$, the goal which is 1, and a status which is given in Listing 7.2.9. Which basically means that if the fraction $value/goal$ is above or equal to 1, the status is green, if the fraction is between 1 and 0.85 the status is yellow, and otherwise the status is red.

```

1 Case
2   When
3     KpiValue("Energy Production vs Energy Consumption KPI") / KpiGoal("
      Energy Production vs Energy Consumption KPI") >= 1
4     Then 1
5   When
6     KpiValue("Energy Production vs Energy Consumption KPI") / KpiGoal("
      Energy Production vs Energy Consumption KPI") < 1
7     And
8     KpiValue("Energy Production vs Energy Consumption KPI") / KpiGoal("
      Energy Production vs Energy Consumption KPI") >= .85
9     Then 0
10    Else -1
11 End

```

Listing 7.2.9: KPI value MDX

A KPI makes it possible to monitor different measures (values), and set a goal, and a status can also be made. The KPI is available in the analysis tools. A "BI Service" perspective is defined in the cube, which defines e.g. which dimensions and measures are available in the BI service application. For now, all dimensions and measures are available.

7.2.2.3 Dimensions

The Date Dimension is defined with the attributes and hierarchy seen in Figure 7.2.1. The attribute relationship is seen in Figure 7.2.2. Because of the hierarchies SSAS needs to know how to process the levels, this is defined by defining key columns, which are composite keys, since otherwise the key used, would occur multiple times (for each parent level). Hence the Month attribute has a composite key of Year, Month, in this order. The Quarter attribute has a composite key of Year, Calendar Quarter, in this order. The Semester attribute has a composite key of Year, Calendar Semester, in this order. The Trimester attribute has a composite key of Year, Calendar Trimester, in this order. All other attributes use their own key. The Device Dimension is defined with the attributes and hierarchy seen in Figure 7.2.3. The attribute relationship is seen in Figure 7.2.4. The hierarchy in this dimension does not require a composite key, since the devices will always only be under one Floor level, and a floor will also be under one House level. The Time Dimension is defined with the attributes and hierarchy seen in Figure 7.2.5. The attribute relationship is seen in Figure 7.2.6. The hierarchies in this dimension requires composite keys for the Minute attribute, where the composite key is Hour, Minute, in this order. For the Quarter Hour attribute, where the composite key is HalfHour, QuarterHour, in this order. For the Quarter Hour Of Day attribute, where the composite key is HalfHourOfDay, QuarterHourOfDay, in this order. For the Second attribute, where the composite key is Hour, Minute, Second, in this order. All other attributes use their own key. The Weather Dimension is defined with the attributes seen in Figure 7.2.7. This dimension has no attribute relationships, because no hierarchies are flattened in this dimension. This also means no composite keys are needed to be defined. Note that this is a junk dimension. The State, Percent and Device Grouping dimensions only have a single attribute, which is the key attribute. For all the key attributes for all the dimensions a Key column is defined, which is the primary key of the dimension table, as well as a Name column, which can be found in the design chapter. All the dimensions are ordered by the key column.

7.2.2.4 Roles

An External Users role is defined for the cube, this role has only been given read access, hence the users with this role are not allowed to administer or process the cube, only read from it. An ExternalUser group is defined in the active directory (AD). This AD user group is the only member of the the External Users role. The role gives access to read and drill trough in the cube, hence it is possible for these users to get access to the most granular level of data in the cube. For now the role has access to all dimensions and measures, however this can be configured later, e.g. to only give access to data for a specific house, or only access to summarised data.

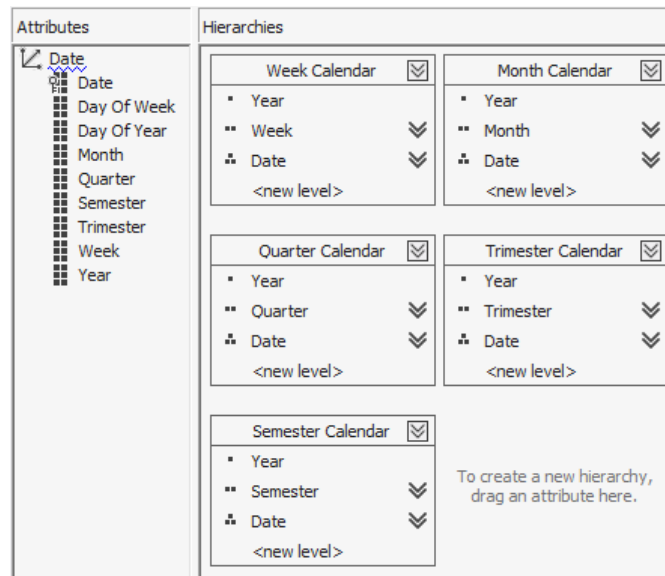


Figure 7.2.1: Date dimension and hierarchies

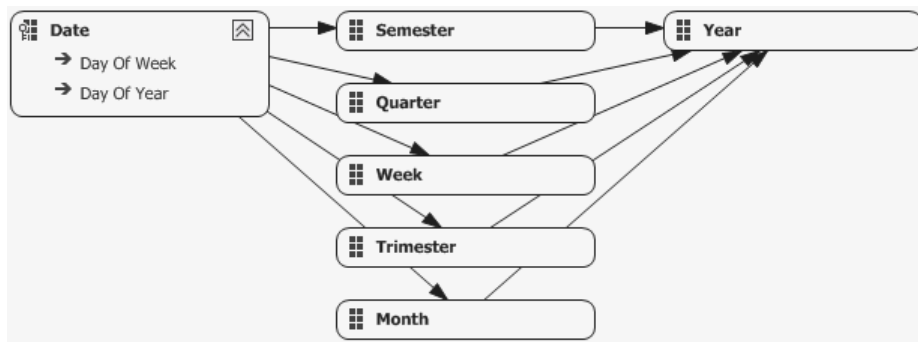


Figure 7.2.2: Date dimension attribute relations

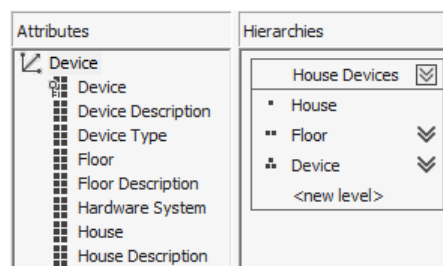


Figure 7.2.3: Device dimension and hierarchies

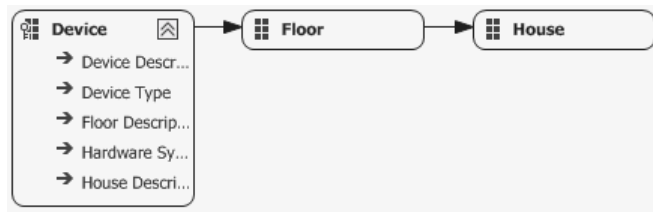


Figure 7.2.4: Device dimension attribute relations

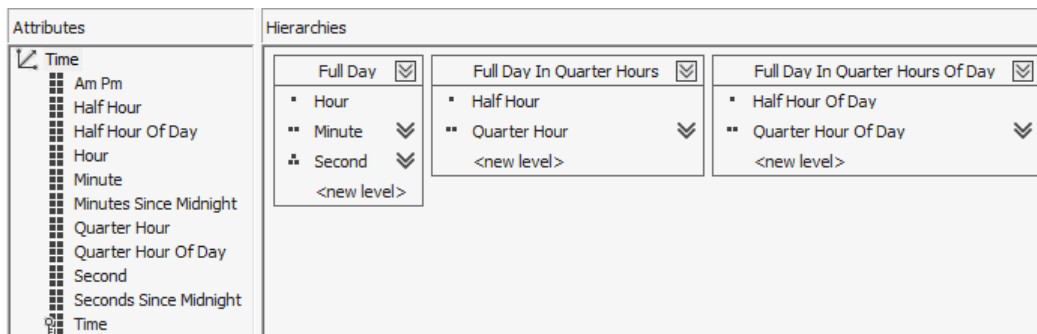


Figure 7.2.5: Time dimension and hierarchies

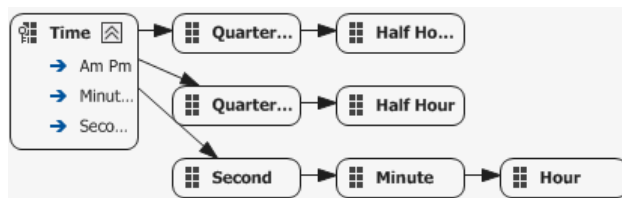


Figure 7.2.6: Time dimension attribute relations

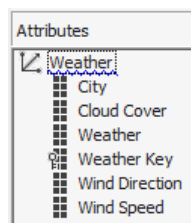


Figure 7.2.7: Weather dimension

7.3 Details on ETL

The implementation of the ETL service application is done using the packages defined in the design chapter. The implementation is done using C#.Net. The Entity Framework is used for all data access. The data warehouse destination is done through EF Code-First, which makes it possible to quickly edit the data model and generate a database schema based on the entities. EF Database-First is used to auto-generate entity classes from an existing database schema, this is done for the two database source, Data Collection and HSI, if these models are updated, the EF can auto-generate new entity classes which match the database schema. When using Database-First, the DAL is made, such that an exception is thrown if the model is changed using the auto-generated classes and a code-first operation is done to generate a database schema. This ensures that the source databases are never accidentally changed though the ETL solution. Code-First is also used for the Metadata database. For the Numeric and Bool Records from the Data Collection operational source system, Microsoft SQL Server Integration Services (SSIS) is used. SSIS can process large amounts of data much more efficiently than EF. Hence for the smaller data source, EF is used, while for the larger, i.e. the data from the Data Collection service is loaded through SSIS.

7.3.1 ETL solution

The ETL solution is formed from three layers, the service layer, business logic layer and data access layer.

7.3.1.1 Service

The service layer consists of three presentation applications. A Windows Service, a console application and a web site. The Windows Service is seen in Figure 7.3.1. The service consists of a Timer, which is instantiated and started in the OnStart method of the service, which is called when the service is starting. The Timer is initially set to an interval of 30 seconds, and the AutoReset is set to false, because the timer should not execute a tick after the first tick has been executed. The tick executes the method Processing-TimerOnElapsed method, which calls the Process method of the ProcessingHelper class, which processes all the processing steps defined in the design chapter. After the processing is done, the timer interval is set to a processing interval which is defined in a configuration file. The default value is set to 15 minutes. Which means that the ETL will process data every 15 minutes after it has processed the previous time. This is important, since a tick is run on the thread pool, race conditions could happen if the AutoReset is not set to false. Race conditions can even happen if the interval is set to a very low value, because the stopping of the timer is done in another thread than the elapsed tick method. The OnStop method makes sure to stop the timer, to avoid the processing is executed during the stop procedure of the Windows Service. The ProcessingHelper class contains methods for the different processing steps, i.e. ProcessDateTime, ProcessHSI, ProcessWeather, ProcessSDE, ProcessSSIS and ProcessSSAS. This is also the order in which the Process method calls these methods. The ProcessDateTime instantiates a DateTimeManager and executes the PopulateDateDimension followed by executing the PopulateTimeDimension of the manager. The ProcessHSI instantiates a HSIManager and executes the PopulateGroupingDimension, PopulateDeviceDimension, PopulateSwitchButtonDimension and PopulateDeviceGroupFacts, methods of the manager, in

this order. The ProcessWeather instantiates a WeatherManager and executes the PopulateWeatherFacts method of the manager. The ProcessSSIS instantiates a SSISManager and executes the Execute method of the manager. Lastly the ProcessSSAS instantiates a SSASManager, which executes the Connect method followed by executing the ProcessDatabase method of the manager. The Console application is seen in Figure 7.3.2, this console application takes one parameter, which can be "all", "datetime", "hsi", "sde", "ssas", "ssis", or "weather". Based on the option the respective method is called in the ProcessingHelper class. For all the Process method is called. The console application hence enables separate processing of the different steps, which are all processed by the Windows Service. The Web site is seen in Figure 7.3.3. The web site is based on the MVC paradigm. There are three different pages, and hence three controllers. The first page is a Home page, which is controlled by the HomeController, which controls the HomeIndex view, which is populated with information from the HomeIndexViewModel. The second page is a Relation management page, which is controlled by the RelationController, which controls the RelationIndex, RelationCreate, RelationEdit and RelationDelete views, which are populated with information from the RelationViewModel, for the RelationIndex view the RelationIndexViewModel is used and for the RelationCreate and RelationEdit the RelationCreateViewModel is used. The last page is a Weather location management page, which is controlled by the WeatherController, which controls the WeatherIndex, WeatherDetails, WeatherCreate, WeatherEdit and WeatherDelete views. The different methods in the controllers populates the view models. The two management pages are used for CRUD operations on SourceDestinationRelation and WeatherLocation entities. Because of the simplicity of the WeatherLocation entity, the default boilerplate which is auto-generated is used. For the SourceDestinationRelation the field SourceRowIdentifier, means that the controller, view models and views is made manually. The SourceRowIdentifier is a dictionary which is not easily mapped to web site controls, the solution for this is done by using SelectList of the different types of identifiers for the relations, i.e. a SelectList for HSI Groups, Items and ItemTypes. SelectList are also used for end-user convenience for the Source and Destination tables. Since the identifiers are used for all the Data Collection sources this field is pre-populated with the value GenericRecords. And since the only source is the Data Collection, this field is also pre-populated. The destination table list is a list of all the Numeric Record and Bool Record fact tables. The HomeIndex view is used to show all the metadata from the Metadata entities. As well as give a summary of the latest load of the Numeric and Bool records, where unmapped records are put into UnmappedGenericFact entities. All the views use the SharedLayout to give the same look and feel for the pages. The default look and feel is used, which is based on Bootstrap. To secure the web site from unauthorised users the AdGroupAuthorize class is used, which is decorated as an attribute on the methods returning an ActionResult from the controllers. The AdGroupAuthorize checks the LogonUserIdentity which is sent in the request. If it is not sent, the web site will prompt the user for a valid Windows user name and password, which is on the AD or the local server. Since external users are given access through an external user account which is also a valid Windows user, the AdGroupAuthorize, checks the associated user groups for the user. An "InternalUsers" user group is made in the AD or the local server, which includes the authorised users of the ETL configuration.

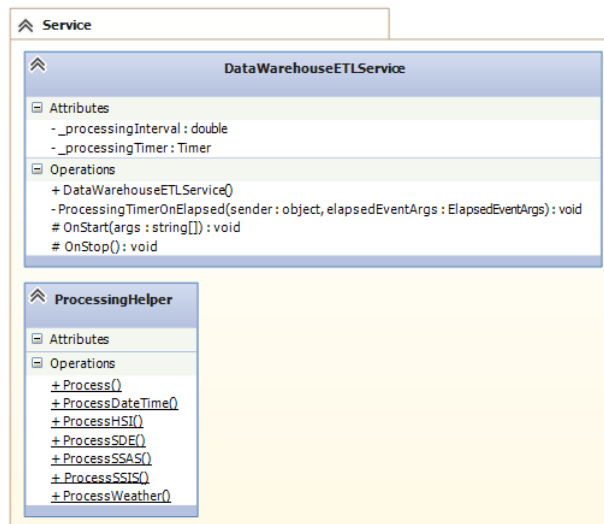


Figure 7.3.1: Service related classes

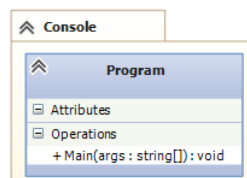


Figure 7.3.2: Console class

7.3.1.2 Business logic

The business logic is seen in Figure 7.3.4. The different managers for the different processing steps is seen. The ManagerBase is the base manager, which includes access to all the data access, i.e. the two data sources, the data warehouse destination and the metadata. Because the Delta measure is calculated so much, a static GetDelta method is included in the ManagerBase. The DateTimeManager has two methods one for populating the TimeOfDayDimension and one for populating the DateDimension. Because of the amount of data in the TimeOfDayDimension, the parameter AutoDetectChangesEnabled for the Configuration of the data warehouse context, is set to false, otherwise the initial load of especially the TimeOfDayDimension will take several minutes. Before populating any of the dimensions the metadata is checked to see whether the dimensions are already loaded, if so, the populating is skipped. The HSIManager has four methods, one for populating the GroupingDimension, one for populating the DeviceDimension, one for populating SwitchButtonDimension and one for populating the DeviceGroupFact. The GroupingDimension is populated with Group entities from the HSI. The DeviceDimension is populated with the Item entities from the HSI, the HSI Floor, House and ItemType are flattened/denormalised into the device dimension, e.g. the Embrace House entity is populated for all the items which has a reference to this house entity. The SwitchButtonDimension is populated by iterating all the HSI ItemTypes which have the Discriminator ItemTypeBool. For each of these entities the True and False State entities are loaded into the SwitchButtonDimension. For all the HSI dimensions the AlternateKey is used to either add or update the dimension entities, this is done using the AddOrUpdate method of the DAL context. Lastly the Device-

GroupFact is populated by iterating all the devices in the DeviceDimension and finding the HSI Item. For all the HSI Groups associated with the item, the Grouping is found in the GroupingDimension, when the DeviceDimension entity and the GroupingDimension entity is found, a new DeviceGroupFact entity is made, and added or updated using the AddOrUpdate method. This operation only adds and keeps existing many-to-many relations. To remove the groups that are no longer valid for the HSI Item, the list of valid groups (GroupingKeys) is kept for each item and the set of DeviceGroupFact entities where the DeviceKey is the current device in the iteration and where the GroupingKey is not contained in the valid group list, is removed from the DeviceGroupFact context. This results in an up to date list of many-to-many relations between groups and items. The WeatherManager has a single method which is used to populate the weather facts and the junk weather dimension. The manager uses XDocument to load the REST XML which is provided by OpenWeatherMap.org, and Linq to XML is used to get the individual fields of the XML. Each weather location from the metadata is iterated. When the XML is loaded the "lastupdate" field is checked with the latest loaded weather forecast for the location in the WeatherFact context. If the lastupdate value and the DateKey and TimeKey corresponds to this, the iteration is skipped. Otherwise the fields of the WeatherFact is loaded into a new WeatherFact entity and added to the DAL context. To calculate the delta measure for the different fields, the same object which was used to check the lastupdate is used. Before adding the new WeatherFact entity to the DAL context, the dimension combination is checked, if the combination exists, the key is set in WeatherKey foreign key in the WeatherFact. Otherwise the combination is added as a new dimension entity in the WeatherDimension, and the added key is used for the foreign key in the WeatherFact entity. The SDEManager has a method for populating the different measurements. The measurements from the SDE are in REST JSON format, so the same loading as for the Weather cannot be used, instead the Parse method from JObject, which comes from the Newtonsoft.Json library is used. This can parse a JSON object like the JSON arrays which are provided from the SDE as dynamic objects in C#. Dynamic objects are loaded at runtime, and makes it possible to manipulate with the object, as with normal properties and fields of objects. To calculate the delta measures the latest fact row is retrieved from the DAL context. Before loading the different measures, the timestamp, is checked with the metadata, to ensure only the newest facts are loaded. Once the facts are loaded the timestamp is saved in the metadata. The SSISManager has an Execute method, because the loading of data is done through a SSIS package, which is a definition package, which defines the data sources, the destinations and a number of transformations. Before executing the package, metadata is loaded which includes the latest loaded id for both Bool and Numeric Records. The SourceRowIdentifier relations are also loaded. These parameters are loaded into the package, before executing it. After the execution of the package the errors and warnings are collected to see whether the execution was successful. These metrics together with the duration for the processing time is saved in the metadata, where the package has also saved an updated value for the latest loaded id for both types of Records. Lastly the SSASManager, has two methods, one for connecting to the OLAP cube and one for querying a processing request. The duration of the processing time is saved in the metadata for monitoring. For all the populating methods of the different managers, the duration of the processing time is saved in the metadata. The common extensions seen in Figure 7.3.5, are used between the managers to e.g. derive the DateKey and TimeKey. The DictionaryToXml and the reverse DictionaryFromXml are also used for most of the managers, however this is transparently done through the entities. Since it is not possible to save a Dictionary object natively in a database using EF. A workaround is used, where the Dictionary is serialised to an XML string, which is then saved in the database. To make the use of this as transparent as possible, the Dictionary is available through the entity. Listing 7.3.1 shows an example of the implementation of this. The NotMapped attribute which decorates the Dictionary object, makes sure that EF knows to not handle this property. However it handles the AdditionalMetadataXml property, where it calls the get method, when saving the value of the property to the database, and when loading the entity it calls the set method, which sets the Dictionary property, which can hence be used transparently.

```

1 [Table("Metadata")]
2 public class Metadata
3 {
4     public int Id { get; set; }
5     public string System { get; set; }
6     public string TableName { get; set; }
7     public string Description { get; set; }
8     public long? LatestExtractionId { get; set; }
9     public DateTime? LatestExtractionDateTime { get; set; }
10    // serialize dictionary
11    public string AdditionalMetadataXml
12    {
13        get { return AdditionalMetadata.DictionaryToXml(); }
14        set { AdditionalMetadata = value.DictionaryFromXml(); }
15    }
16
17    [NotMapped]
18    public Dictionary<string, object> AdditionalMetadata { get;
19        private set; }
20
21    public Metadata()
22    {
23        AdditionalMetadata = new Dictionary<string, object>();
24    }

```

Listing 7.3.1: Metadata class with XML serialisation

7.3.1.3 Data access

The data access layer contains the contexts for the different database sources, the Sources are auto-generated using EF Database-First. The Destination and Metadata uses EF Code-First to generate the database schema. The Metadata package includes four different entities, one for Metadata, one for SourceDestinationRelations for the Data Collection, one for UnmappedGenericRecords, which is used for auditing of the data from the Data Collection and one for WeatherLocations. The Metadata package is seen in Figure 7.3.6. The Destination package is seen in Figure 7.3.7 and the Sources package is seen in Figure 7.3.8. The WeatherLocation only has a City field, which hold which city forecasts should be loaded into the Data Warehouse. The Metadata holds fields for the System, and TableName as well as a Description for the Metadata. The LatestExtractionId and LatestExtractionDateTime are used to keep the latest extraction information of the loading, this makes it possible to only load the newest data, and ignore the already loaded data. The AdditionalMetadata is used for additional metadata such as performance metrics like the duration of the latest processing. And for values which are not fields, e.g. for the TimeOfDayDimension a flag for whether the dimension is loaded is set in the additional metadata. The SourceDestinationRelation entity defines a mapping between the records in the Data Collection and the Numeric and Bool Record

fact table. The mapping is used in the SSIS Package. The Order field determines which matching relation should be checked first, this makes it possible to make generic "catch-all" relations, and specific relations for more specific items. The matching is done using the SourceRowIdentifier, which is a dictionary, which can e.g. contain a HSI Item id, if this matches any of the processed records in the SSIS Package, the row is sent to the DestinationTable. An IncludeDuration is also made, this is used to minimise unnecessary processing, if the value of this field is false, a duration is not calculated during processing of numeric records. The duration is always calculated for Bool records. When a processed row in the SSIS package cannot e.g. be mapped to a destination, the row should not just be discarded, instead the UnmappedGenericRecords is used. Hence this table contains auditing information of the unmapped processed records. This can help to find more SourceRowIdentifiers which are currently not set, or figure out why a record was not put into a destination table, e.g. if the value was above the set maximum. This makes it possible to optimise the cleaning of the processing of the data. If the unmapped records needs to be reloaded, the data warehouse needs to be reloaded, i.e. the data warehouse database should be deleted and the database schema should be regenerated.

7.3.2 SSIS ETL Package

The SSIS package control flow is seen in Figure 7.3.9. It is seen that first the Numeric Data Flow is executed, followed by the execution of the Boolean Data Flow. The Numeric Data Flow Task is seen in Figure 7.3.10. The Boolean Data Flow Task is seen in Figure 7.3.11. The only transformations used in the two data flow tasks are non-blocking transformations. There exists three types of transformations, blocking, non-blocking and partially-blocking transformations. To minimise the processing time only non-blocking transformations are used. The blocking transformations requires all the data to be loaded into the SSIS package before the data can "flow" to the next transformation, typical blocking transformations include sorting. Since sorting is necessary to be able to optimise the calculation of the delta measures, so previous records are always loaded right before the next, in a sequence sorted by the date and time and HSI items. By doing this it is not necessary to make as many SQL queries to the source to get the latest value, instead the latest value is loaded in the previous iteration of the transformation, and only the first record of the item needs to make an SQL query to the source. Because of this sorting is required and the only non-blocking transformations are wanted in the SSIS package, the ordering is done in the source, i.e. by the database, which is much quicker than first loading all data into SSIS package. Partially-blocking needs to load a certain amount of data before it can continue to the next transformation, and hence these should also be avoided in the SSIS package. Transformations edit the content of columns or adds additional columns.

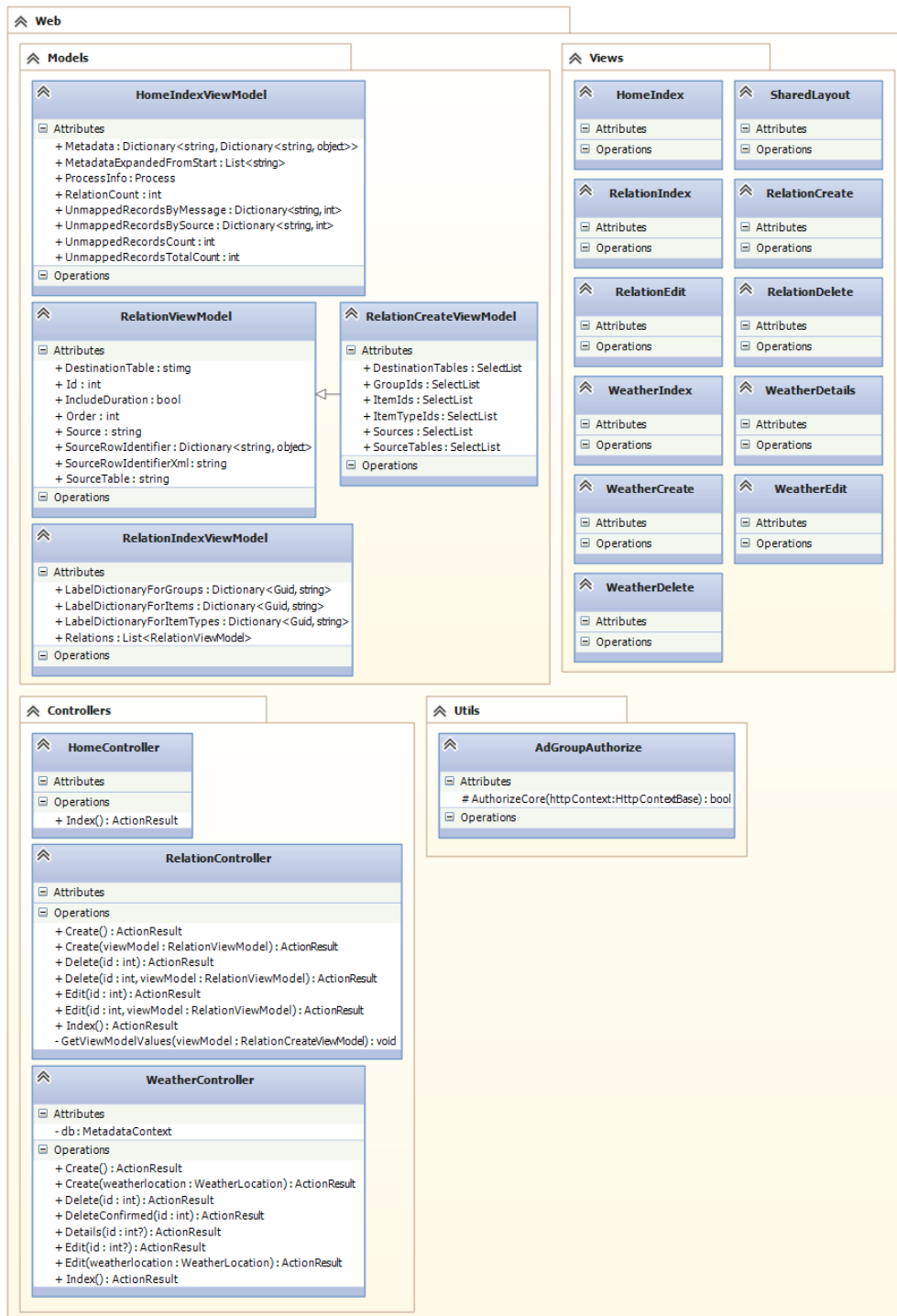


Figure 7.3.3: Web related classes

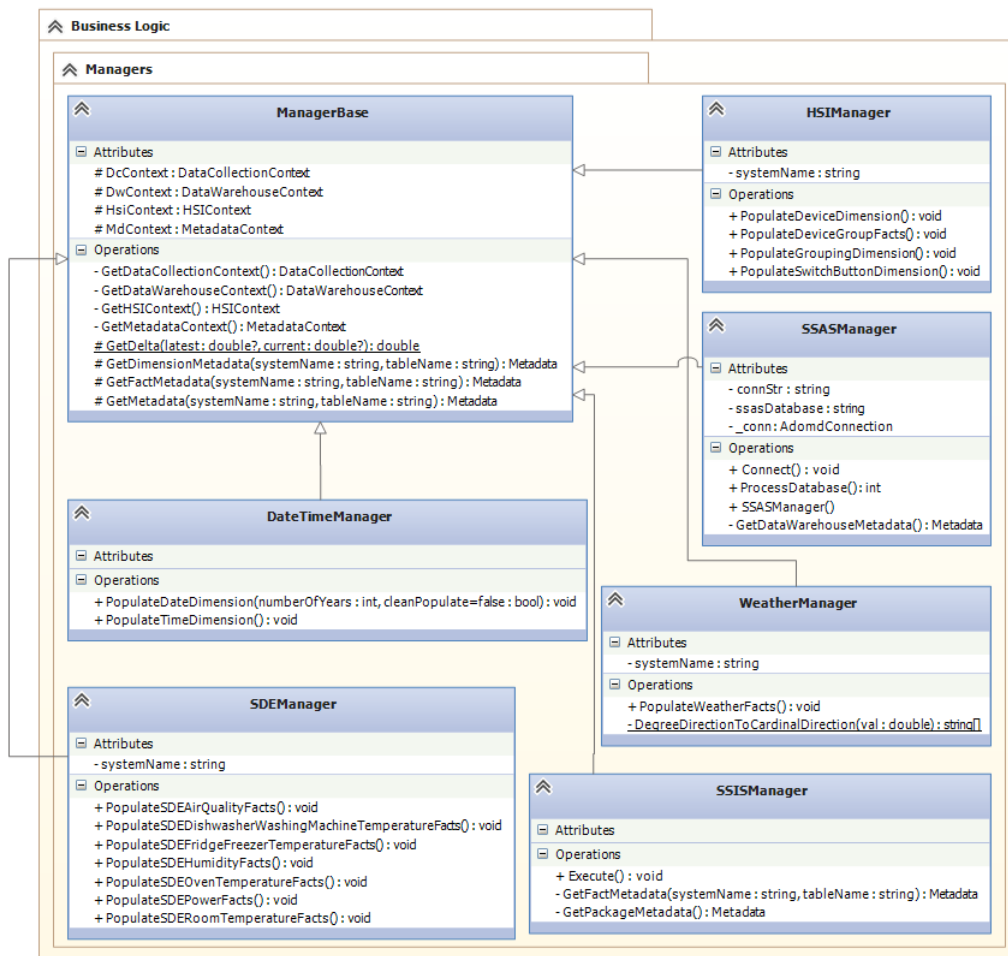


Figure 7.3.4: ETL business logic classes

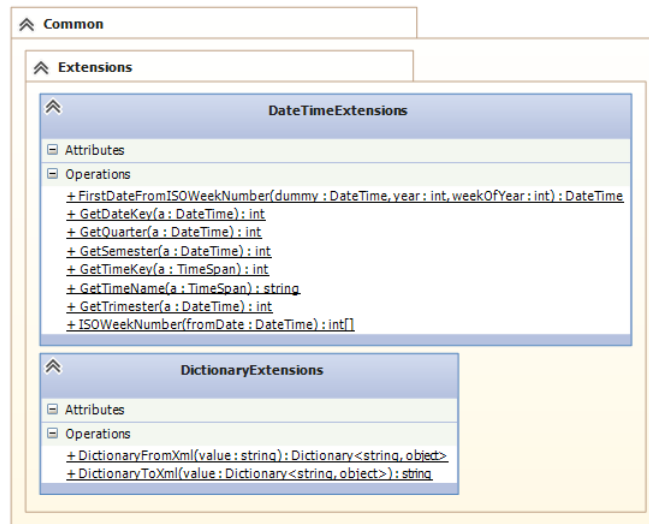


Figure 7.3.5: ETL common classes

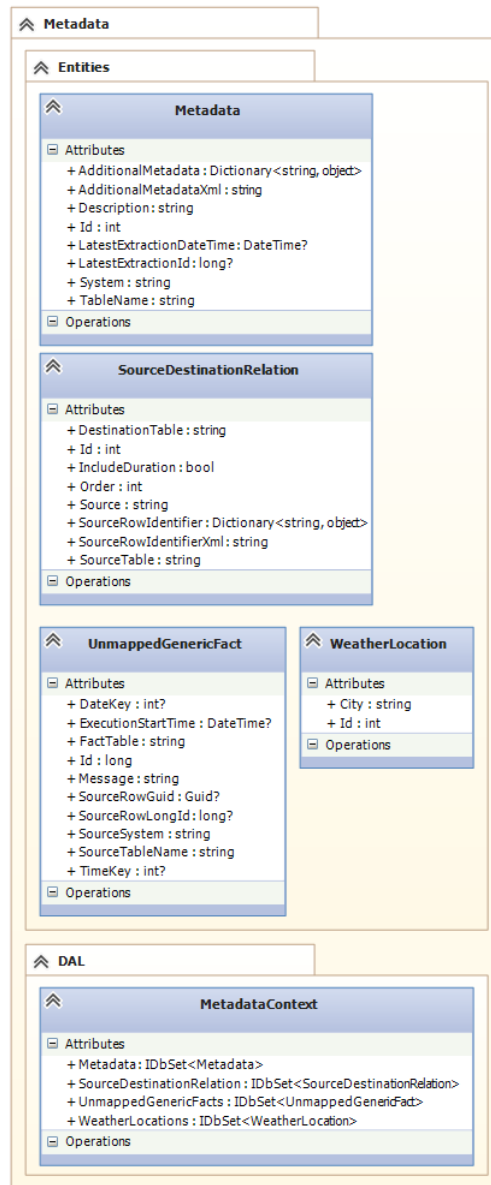


Figure 7.3.6: Metadata classes

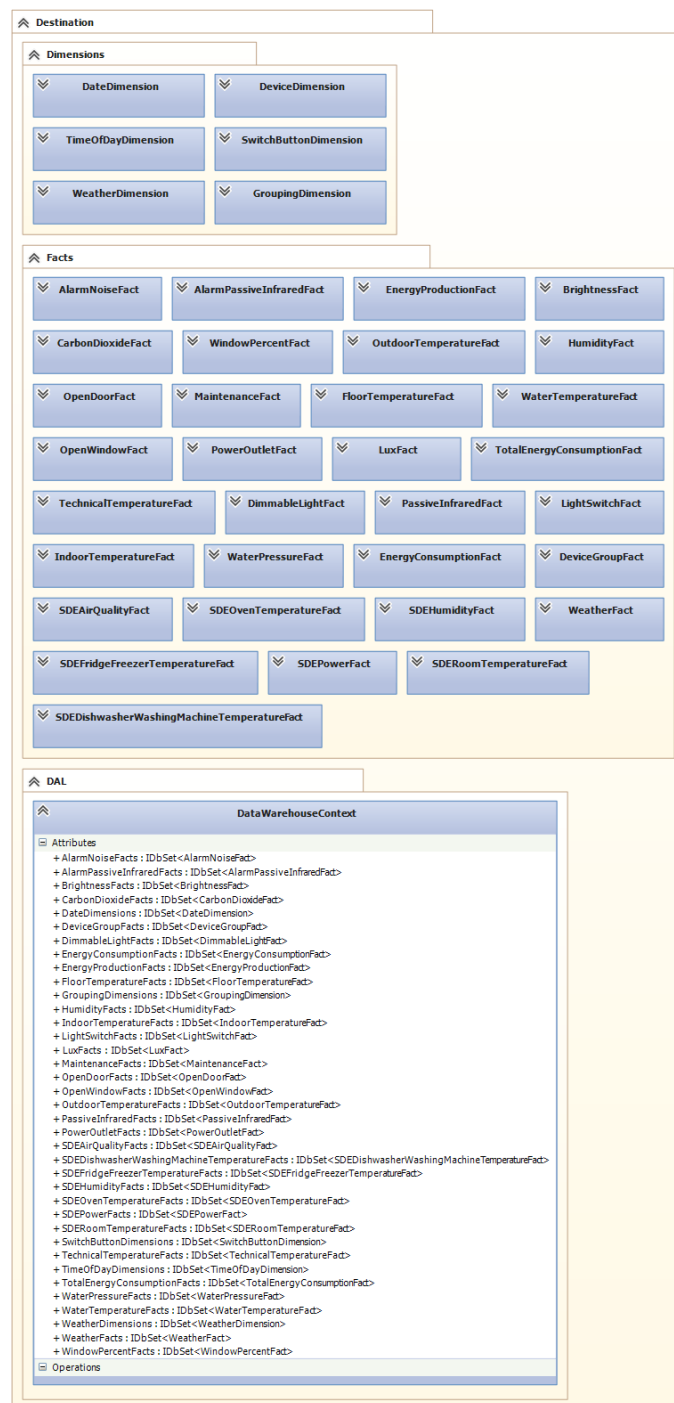


Figure 7.3.7: Destination classes

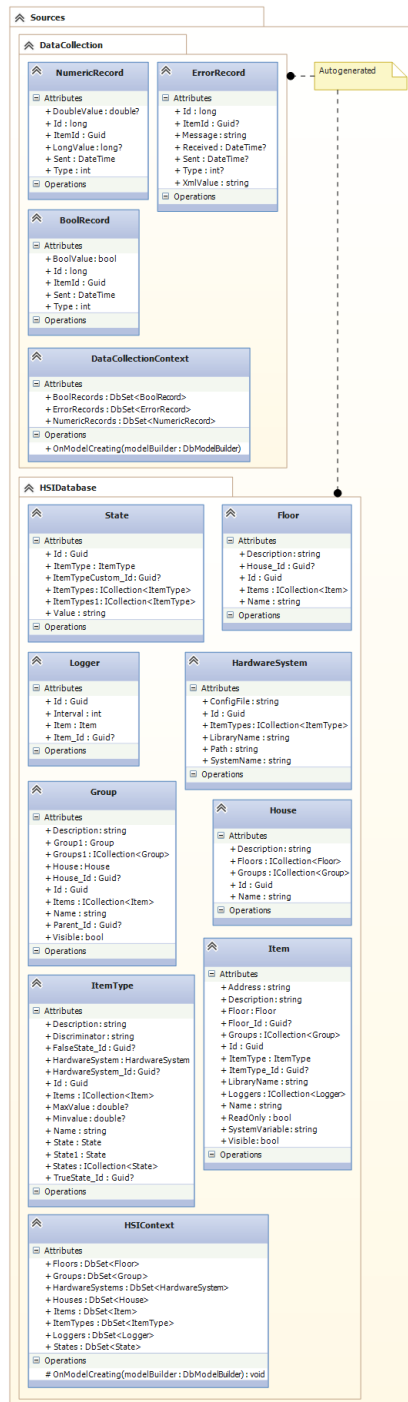


Figure 7.3.8: Sources classes

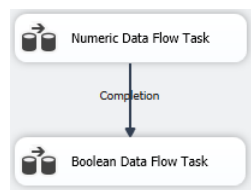


Figure 7.3.9: SSIS Package control flow

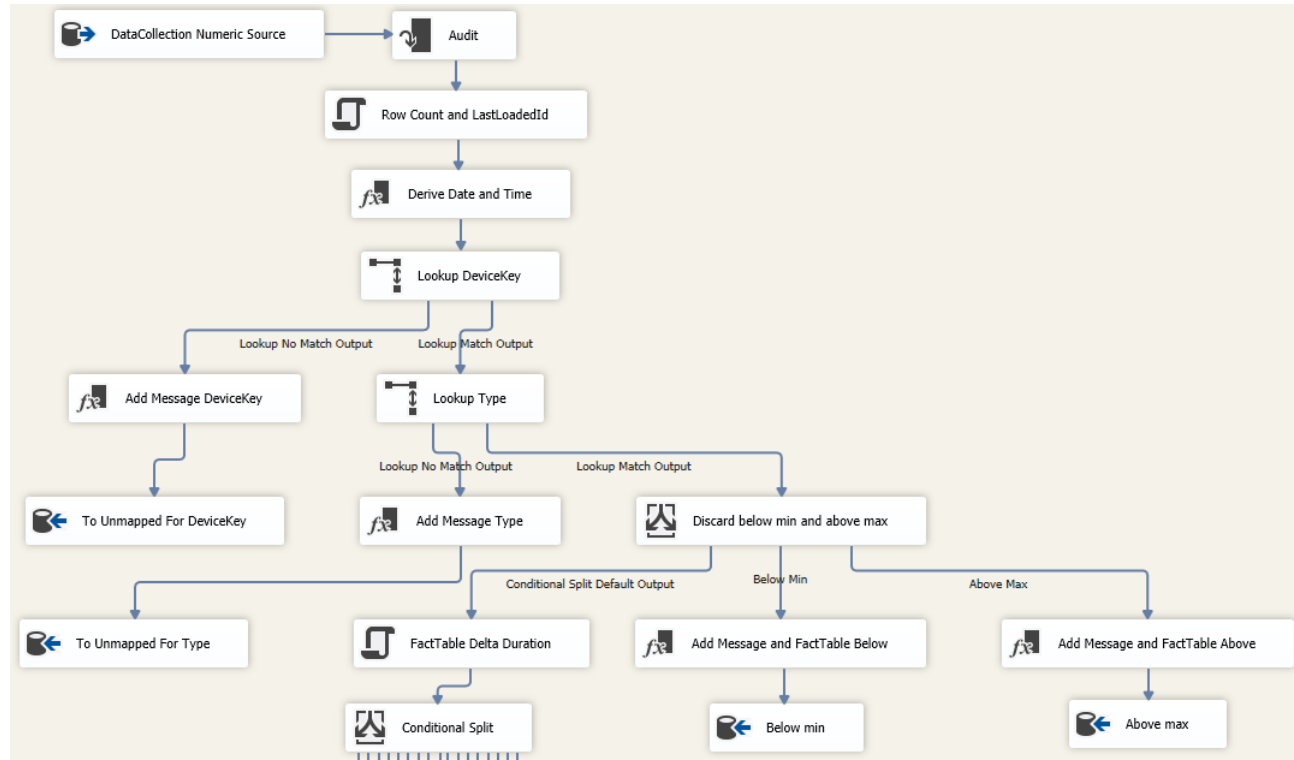


Figure 7.3.10: SSIS Package numeric data flow

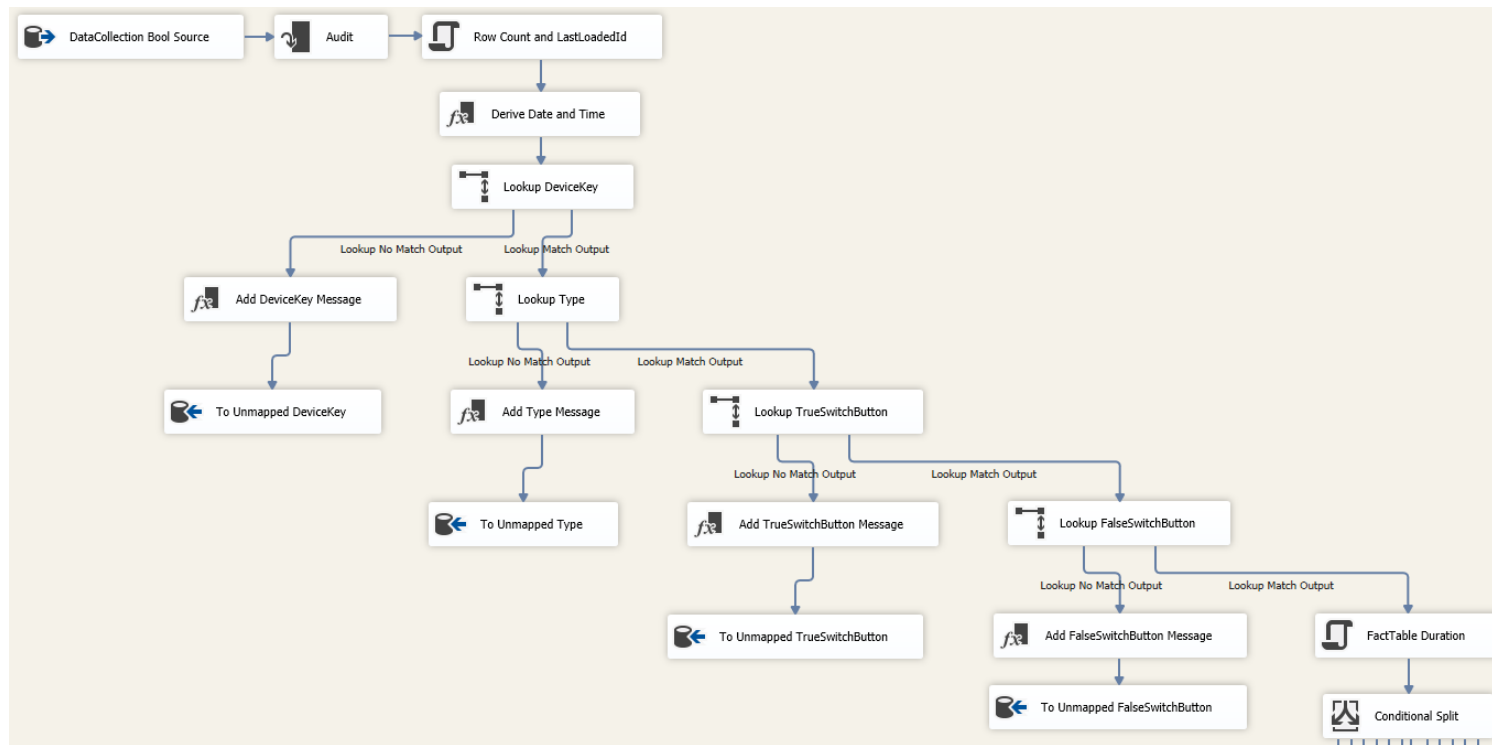


Figure 7.3.11: SSIS Package bool data flow

The Numeric Data flow seen in Figure 7.3.10, starts with a OLE DB source with an SQL command from a variable. The variable contains:

```
1 SELECT * FROM [dbo].[NumericRecords] WHERE [Id] > " + (DT_WSTR, 500)
   @[User::LastLoadedId] + " ORDER BY [ItemId], [Sent]
```

Listing 7.3.2: SQL variable for numeric records

The [User::LastLoadedId] is loaded from the SSIS manager, since it is a long integer the variable is cast to a string. Next the Audit transform adds the execution start time as a column, which is used for the UnmappedGenericRecords. Next a script component is used to count the rows and get the execution start time, and updating the metadata for [System] = 'DataCollection' and [TableName] = 'NumericRecords', with the LastLoadRowsLoaded with the count of rows and ExecutionStartTime with the execution start time, together with the LatestExtractionDateTime which is set to the current timestamp. The LastLoadRowsLoaded and ExecutionStartTime are in the AdditionalMetadata. Next the DateKey and TimeKey is derived from the Sent column, and added as DateKey and TimeKey columns. Additionally a SourceSystem = "DataCollection" and a SourceTableName = "NumericRecords" column is inserted which is used for the UnmappedGenericRecords. Next a lookup is done to get the DeviceKey and TypeAlternateKey from the DimDevice. These are added as new columns. The lookup is done by using the DeviceAlternateKey from DimDevice and ItemId from NumericRecords. If the lookup has no match, the row is sent to the UnmappedGenericRecords. Next a lookup is done to get the MinValue and MaxValue from the ItemTypes from the HSIDatabase. The MinValue and MaxValue are added as columns. The lookup is done by using the TypeAlternateKey from the column and the Id from the ItemTypes table in the HSIDatabase. If the lookup has no match, the row is sent to the UnmappedGenericRecords. Next a conditional split is done to filter rows where the DoubleValue is below the MinValue or the DoubleValue is above the MaxValue, all other rows are sent to the next transform. All the filtered are sent to the UnmappedGenericRecords. Next a script component is used to find the fact table destination based on the SourceRowIdentifiers. If a destination cannot be found or it is set to be ignored, the fact table is set to Error or Ignore and the transformation returns, so no further processing is done for these rows. Next if the include duration option is set, the duration is calculated based on the LongValue. If the previous LongValue is not the same as the LongValue of the current row, an additional row is inserted in the flow. If the LongValue is the same, a single row is continued in the flow. Next a delta measure is calculated from the DoubleValue. The number of rows which is processed by the script component is counted and inserted in the AdditionalMetadata with the key LastLoadRowsLoadedByCalc. Next a conditional split is used to route the row to the found destination fact table. If the fact table is set to Ignore or Error the rows are sent to UnmappedGenericRecords. Otherwise it will be sent to the respective destination. All the destinations use Bulk Insert, which increases the insert performance.

The Numeric Data flow seen in Figure 7.3.10, starts with a OLE DB source with an SQL command from a variable. The variable contains:

```
1 SELECT * FROM [dbo].[BoolRecords] WHERE [Id] > " + (DT_WSTR, 500) @[
   User::BoolLastLoadedId] + " ORDER BY [ItemId], [Sent]
```

Listing 7.3.3: SQL variable for bool records

The [User::BoolLastLoadedId] is loaded from the SSIS manager, since it is a long integer the variable is cast to a string. The flow is basically the same as for the Numeric Data Flow until the Lookup Type. The type lookup is used to get the TrueState_Id and FalseState_Id from the ItemTypes table in the HSIDatabase. If the lookup has no match the row is sent to UnmappedGenericRecords. Next the SwitchButtonKey for the true value is found, based on the SwitchButtonAlternateKey from the DimSwitchButton and the TrueState_Id found previously. If no match is found the row is sent to the UnmappedGenericRecords. Next the SwitchButtonKey for the false value is found, based on the SwitchButtonAlternateKey from the DimSwitchButton and the FalseState_Id found previously. If no match is found the row is sent to the UnmappedGenericRecords. Next the fact table and duration is found, just like for the Numeric Data Flow, except the duration is calculated for all rows. If the fact table was not found or it is set to Ignore, the processing of the row is returned, so the duration is not calculated for these. Next a conditional split is done to send the rows to the found fact table destinations. If the fact table is set to Ignore or Error the rows are sent to the UnmappedGenericRecords. Otherwise the rows are sent to the respective destinations. Bulk Insert is also used for these destinations to increase performance.

7.4 Details on BI application

The implementation of the BI application follows the design. Additional methods are however introduced in the `BusinessIntelligenceManager`, to be able to generate the reports from the cube query output. The implementation uses C#.NET. The `ADOMD.NET` library is used to communicate with the SSAS cube. For the relaying of the OLAP cube through the internet, the ISAPI extension `MSMDPUMP` is used and set up on the Microsoft Internet Information Services (IIS) web server. The `MSMDPUMP` is put into a separate folder in the web site for the BI solution.

7.4.1 BI application

The BI application is made as a Windows Service, which has a `OnStart` and `OnStop` method. The `OnStop` method is not used in this application. The `OnStart` method instantiates a new P2P broker and subscribes to the messages `GetMeasuresRequest`, `GetDimensionsRequest` and `GetInsightReportRequest`. The respective callback methods are the `SendResponse(GetMeasuresRequest)`, `SendResponse(GetDimensionsRequest)` and `SendResponse(GetInsightReportRequest)`. How these interact with the business logic and SSAS cube is illustrated in Figures 7.4.1, 7.4.2 and 7.4.3.

7.4.1.1 Get measures

In Figure 7.4.1 the sequence of getting measures is shown, it is seen that first connection is made to the SSAS cube, next get method `GetAllMeasures` is called, which sends an SQL schema command to the SSAS cube to get the measure group names. Next the list of measures are retrieved from the cube specified in the configuration file, for this the cubes on the server are retrieved and the measures list read. The measure group and measures are put into `Measure` objects and a list of these are published using the `GetMeasures-Response`. When the manager is disposed the connection to the SSAS cube is closed. The reason why an SQL schema command is sent to get the measure groups is because these are not available through the `ADOMD.NET` library.

7.4.1.2 Get dimensions

In Figure 7.4.2 the sequence of getting dimensions is shown, it is seen that first connection is made to the SSAS cube, next get method `GetDimensions` is called, which gets the list of dimensions from the cube specified in the configuration file, for this the cubes on the server are retrieved and the dimensions list read. Next an SQL schema command is sent to the SSAS cube to get the measure group names and associated dimension unique names. Next the hierarchies and levels are flattened and made as `Dimension` objects. Next the list of named sets is retrieved from the cube already read from SSAS, these named sets are also made as `Dimension` objects. The measure group and dimensions are also put into `Dimension` objects and a list of all the dimensions, hierarchies, levels and named sets are published using the `GetDimensionsResponse`. When the manager is disposed the connection to the SSAS cube is closed. The reason why an

SQL schema command is sent to get the measure groups is because these are not available through the ADOMD.NET library.

7.4.1.3 Generate insight report

In Figure 7.4.3 the sequence of generating reports is shown, it is seen that first a connection is made to the SSAS cube. Next the `GetInsightReport` method is called, which generates the query using the `GenerateQuery` method, which is seen in Listing 7.4.1. Next the query is used to get a `CellSet` from the SSAS cube. The `CellSet` is a set of cells and axis. The `CellSet` axis is iterated to get the row and column labels, i.e. axis labels, which are strings and specify the dimensions and measures used in the result. These are multidimensional arrays of strings. Next the `GetCells` is called on the `CellSet` to get all the cells in a multidimensional array of nullable doubles. To generate the `Insight` object, i.e. the report, the column and row labels are flattened by concatenating the labels to a single list of column labels and row labels. Next the nullable doubles for the cells are inserted into a list of `InsightCells` objects. The column and row labels, together with the `InsightCells` and titles for the column and row, i.e. axis titles, are put into an `Insight` object and published with a `GetInsightReportResponse`.

```

1 public string GenerateQuery(List<string> measures, List<string>
  dimensions, List<string> columnDimensions, List<string>
  filterDimensions)
2 {
3     var measureList = measures.Aggregate("", (current, measure) =>
      current + (", " + measure)).Remove(0, 2); // include ", " in
      front
4     var columnDimensionsList = columnDimensions.Aggregate("", (
      current, dimension) => current + (", " + dimension)); //
      include ", " in front
5     var dimensionList = dimensions.Aggregate("", (current, dimension)
      => current + (" * " + dimension)).Remove(0, 3); // include "
      * " in front
6     var filterList = filterDimensions.Aggregate("", (current,
      filterDimension) => current + " FROM ( SELECT ( { " +
      filterDimension + " } ) ON COLUMNS "); // include a FROM
      SELECT for the filter
7
8     // add missing end-parentheses
9     var afterFilterList = "";
10    var noOfFilters = filterDimensions.Count;
11    for (var j = 0; j < noOfFilters; j++)
12    {
13        afterFilterList += ")";
14    }
15
16    // handle multiple measures xor multiple dimensions
17    var columns = (!string.IsNullOrEmpty(columnDimensionsList) &&
      measures.Count == 1) ? "(" + measureList +

```



```

    columnDimensionsList + " )" : measureList;
18
19 // resulting MDX query
20 return "SELECT NON EMPTY {" + columns + "} ON COLUMNS, NON EMPTY
    { ( " + dimensionList + " ) } ON ROWS " + filterList + " FROM
    [" + _cubePerspective + "]" + afterFilterList;
21 }

```

Listing 7.4.1: GenerateQuery method from BI application

The GenerateQuery method is seen in Listing 7.4.1. The functionality of this method is to make a generic MDX query based on the four unique name parameters, to get an insight report. The "base" query is seen in the resulting query, the variables are inserted, when they are needed. The handling of multiple measures or multiple column dimensions makes it possible to use the query result without needing to flatten an extra axes into the insight report.

7.4.1.4 Business logic

The additional introduced methods for the BusinessIntelligenceManager is seen in Figure 7.4.4. Most of the additional introduced methods are used to get the measures, dimensions and generate the insight report, which are specific for the SSAS implementation. The *Count and GetLast* methods are used by the BI web site for a summary of the cube. In addition to the BusinessIntelligenceManager, two new manager classes are also introduced, these are also used by the BI web site. The ActiveDirectoryManager handles resetting the password for the Windows user accounts, which are used externally. The DataWarehouseManager is used to get a summary of the data warehouse, the only method for now is used to get the size of the data warehouse database.

7.4.1.5 Data model

The data model is also slightly changed to add the possibility to get Dimensions and Measures as a Dictionary, with the ParentUniqueNames as keys. The final data model can be seen in Figure 7.4.5.

7.4.2 BI website

The implementation of the BI website is seen in Figure 7.4.6. The BI website is implemented using a MVC paradigm. The website consists of three pages, a home page, which is controlled by the HomeController, a get access page, which is controlled by the AccessController and a quick guide page, which is controlled by the GuideController. All the pages have one view, which is the index view, i.e. HomeIndex, AccessIndex and GuideIndex. The SharedLayout view contains the shared look and feel of the web site, the default layout for MVC applications is used, which is based on Bootstrap. The quick guide page does not have a model because it is just a text page. The get access page has a confirm checkbox and uses a "Completely Automated Public Turing test to tell Computers and Humans Apart" (CAPTCHA) control. The confirm checkbox is in the AccessModel. The CAPTCHA control used is based on Google's reCAPTCHA plugin, and the

HTML for this is generated using the RecaptchaExtension. This is used to protect the site from web crawlers/robots, so the access file, which is a Microsoft Office Data Connection file is only downloaded by actual users. The home page contains stats about the data warehouse, which are saved in the HomeModel. The stats include the database size of the data warehouse, the number of dimensions, KPIs, measures and named sets in the cube and when the cube was last updated and processed. The AdGroupAuthorize from the ETL website is also included in the BI website, however this is not currently used, since the BI website is a public website. The BI website also includes the MSMDPUMP ISAPI extension. This dll file is in a separate folder, called OLAP, in the website and the ISAPI extension is set up on the IIS web server. The MSMDPUMP is a relay component provided by Microsoft to relay the SSAS cube over an IIS web server. The configuration of the MSMDPUMP is set to use the localhost SSAS cube. This can be configured in the configuration file in the OLAP folder.

7.4.3 App service

The BI service application is used by the App service component of the EPIC cloud. The format of the exposed BI requests and responses are in JSON. The app service is made using ASP.NET Web API, and exposes the objects available from the EPIC Common solution/package. An example of an insight report request is seen in Listing 7.4.2. An example of the corresponding insight report response is seen in Listing 7.4.3. The insight report gives the Energy Consumption Measurement As kWh measure and Energy Production Measurement As kWh measure, which are defined in the cube over the three weeks, that the house was online, the first two during the competition and the last during the disassembly. This report can then be used e.g. to make graph in analysis tools.

```

1 PUT http://appservice.epiccloud.solardecathlon.dk/api/Report HTTP/1.1
2 User-Agent: Fiddler
3 Content-Type: application/json
4 Host: appservice.epiccloud.solardecathlon.dk
5 Content-Length: 209
6
7 {
8   "measures":[
9     "[Measures].[Energy Consumption Measurement As kWh]",
10    "[Measures].[Energy Production Measurement As kWh]"
11  ],
12  "primarydimensions":[
13    "[Date].[Week Calendar].[Week]"
14  ]
15 }

```

Listing 7.4.2: HTTP PUT request

```

1 HTTP/1.1 200 OK
2 Cache-Control: no-cache
3 Pragma: no-cache
4 Content-Type: application/json; charset=utf-8

```

```
5 Expires: -1
6 Server: Microsoft-IIS/8.5
7 X-AspNet-Version: 4.0.30319
8 X-Powered-By: ASP.NET
9 Date: Tue, 29 Jul 2014 16:17:30 GMT
10 Content-Length: 556
11
12 {
13   "InsightCells":[
14     {
15       "xOrdinal":0,
16       "xValue":"Week 27, 2014",
17       "yValues":[
18         24.450000000000589,
19         66.630000000000081
20       ]
21     },
22     {
23       "xOrdinal":1,
24       "xValue":"Week 28, 2014",
25       "yValues":[
26         37.530000000000868,
27         105.53000000000001
28       ]
29     },
30     {
31       "xOrdinal":2,
32       "xValue":"Week 29, 2014",
33       "yValues":[
34         4.9699999999999891,
35         23.279999999999998
36       ]
37     }
38   ],
39   "ColumnLabels":[
40     "Energy Consumption Measurement As kWh",
41     "Energy Production Measurement As kWh"
42   ],
43   "RowLabels":[
44     "Week 27, 2014",
45     "Week 28, 2014",
46     "Week 29, 2014"
47   ],
48   "RowTitle":"Week",
49   "ColumnTitle":"Energy Consumption Measurement As kWh – Energy
50     Production Measurement As kWh"
51 }
```

Listing 7.4.3: HTTP response

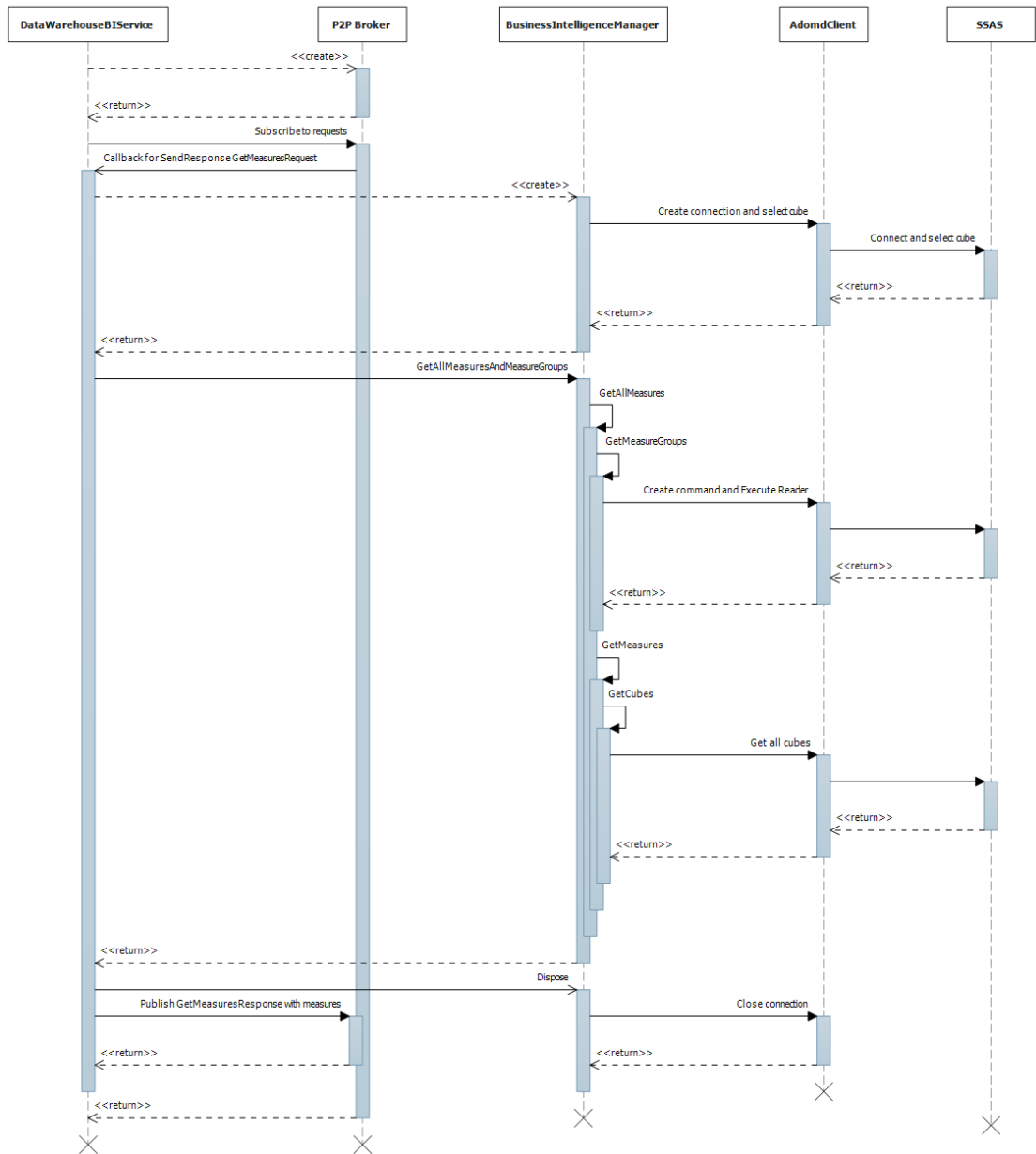


Figure 7.4.1: Sequence of getting measures

7.4. Details on BI application

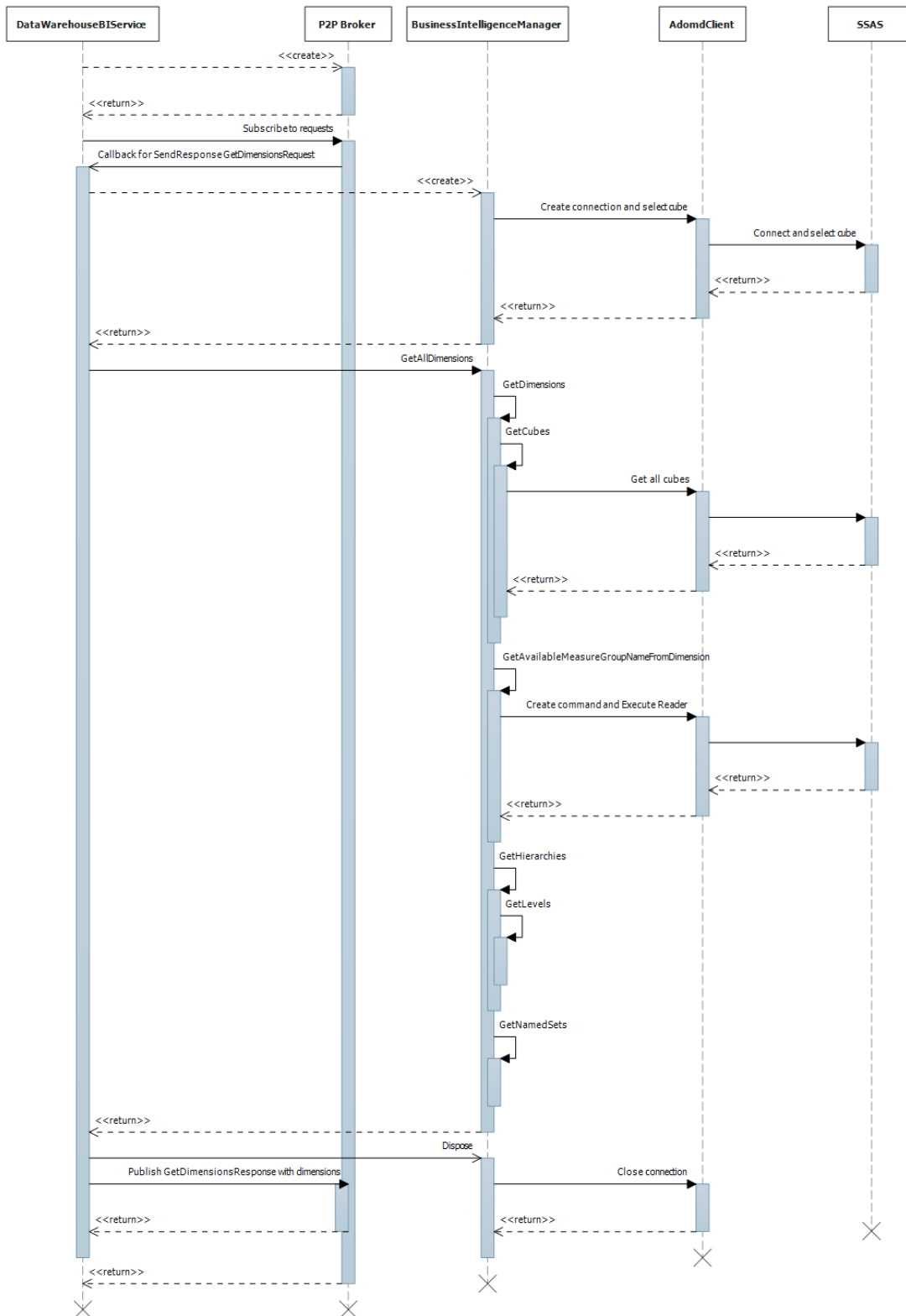


Figure 7.4.2: Sequence of getting dimensions

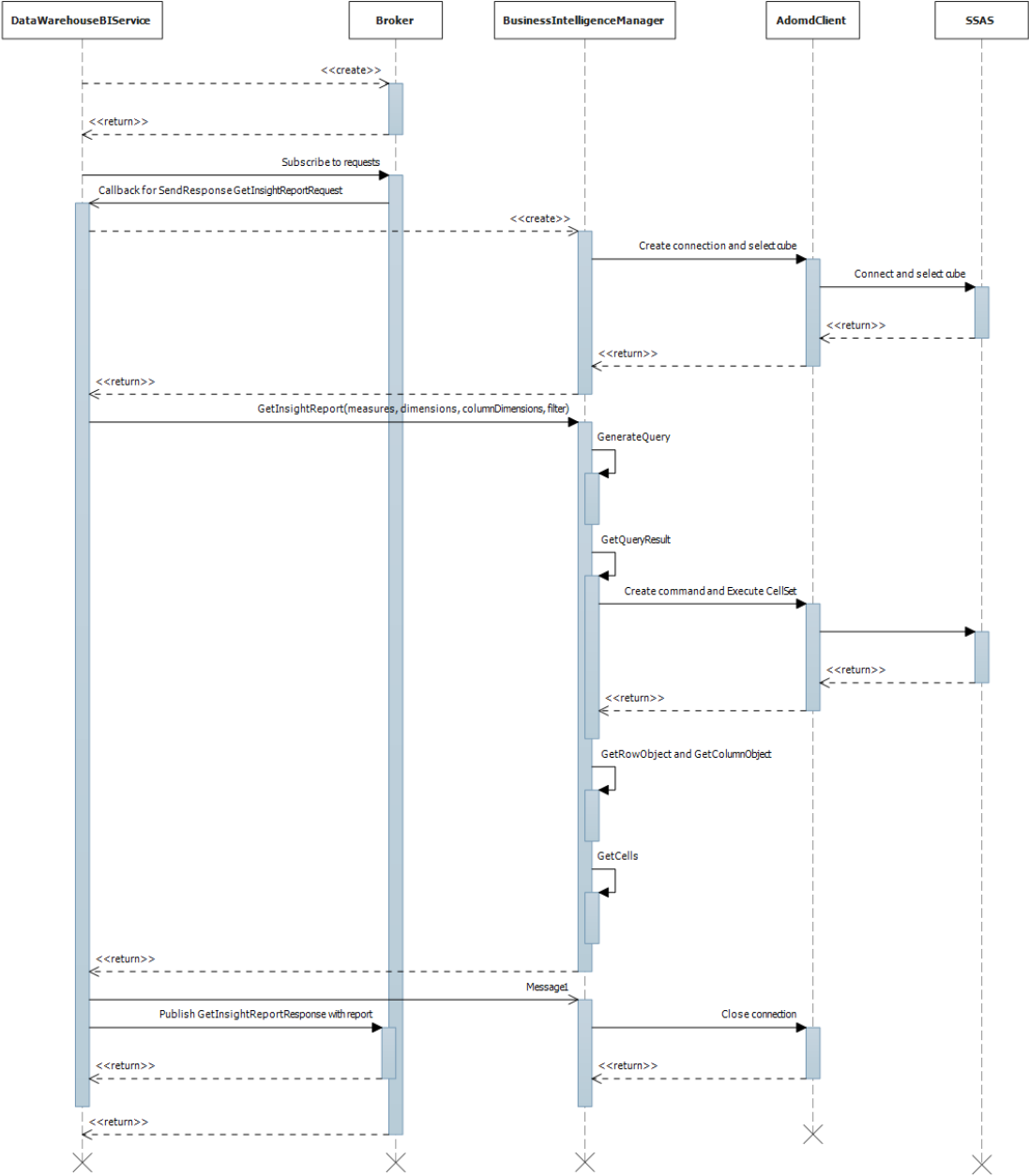


Figure 7.4.3: Sequence of getting report

7.4. Details on BI application

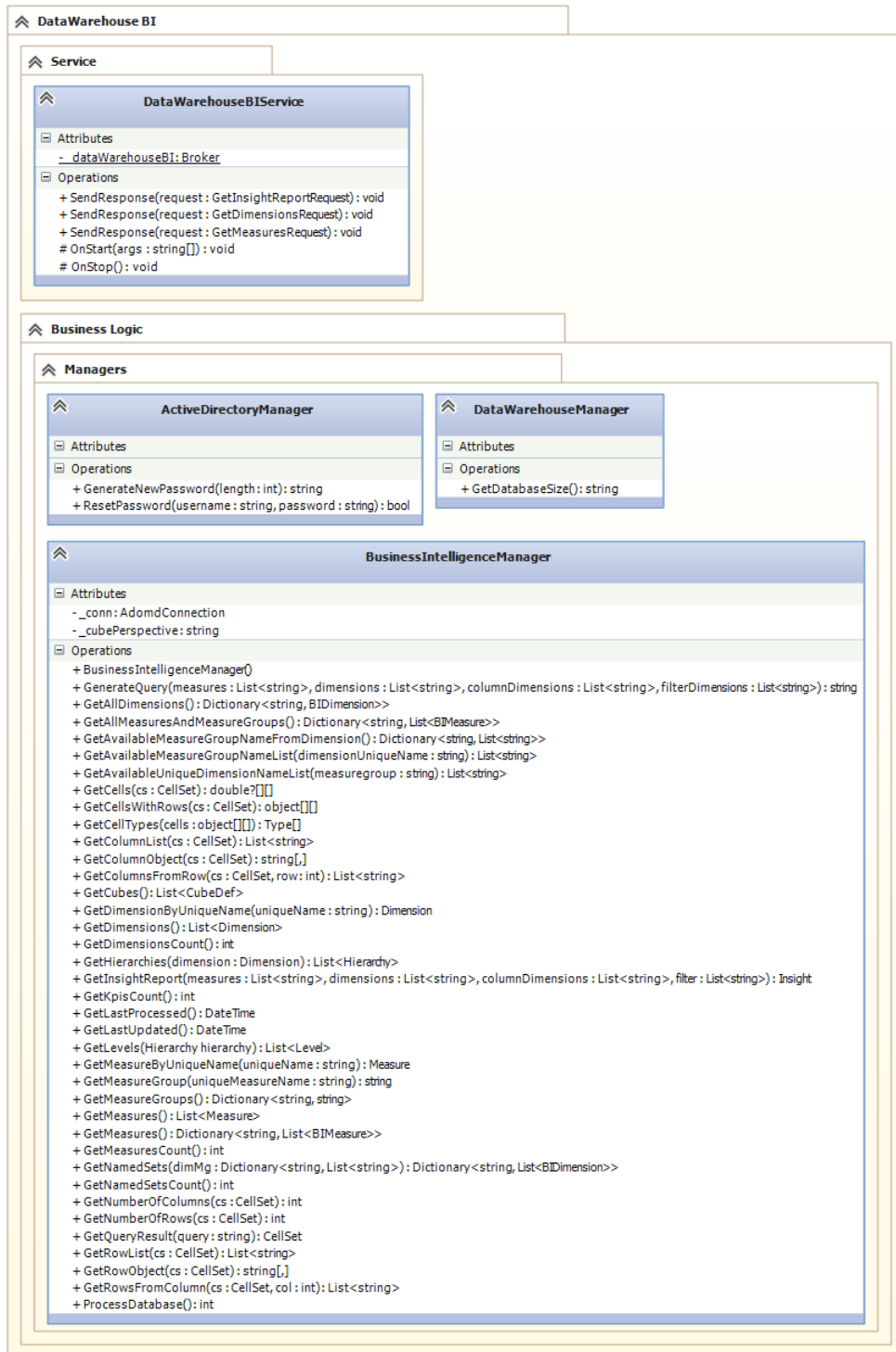


Figure 7.4.4: BI application classes

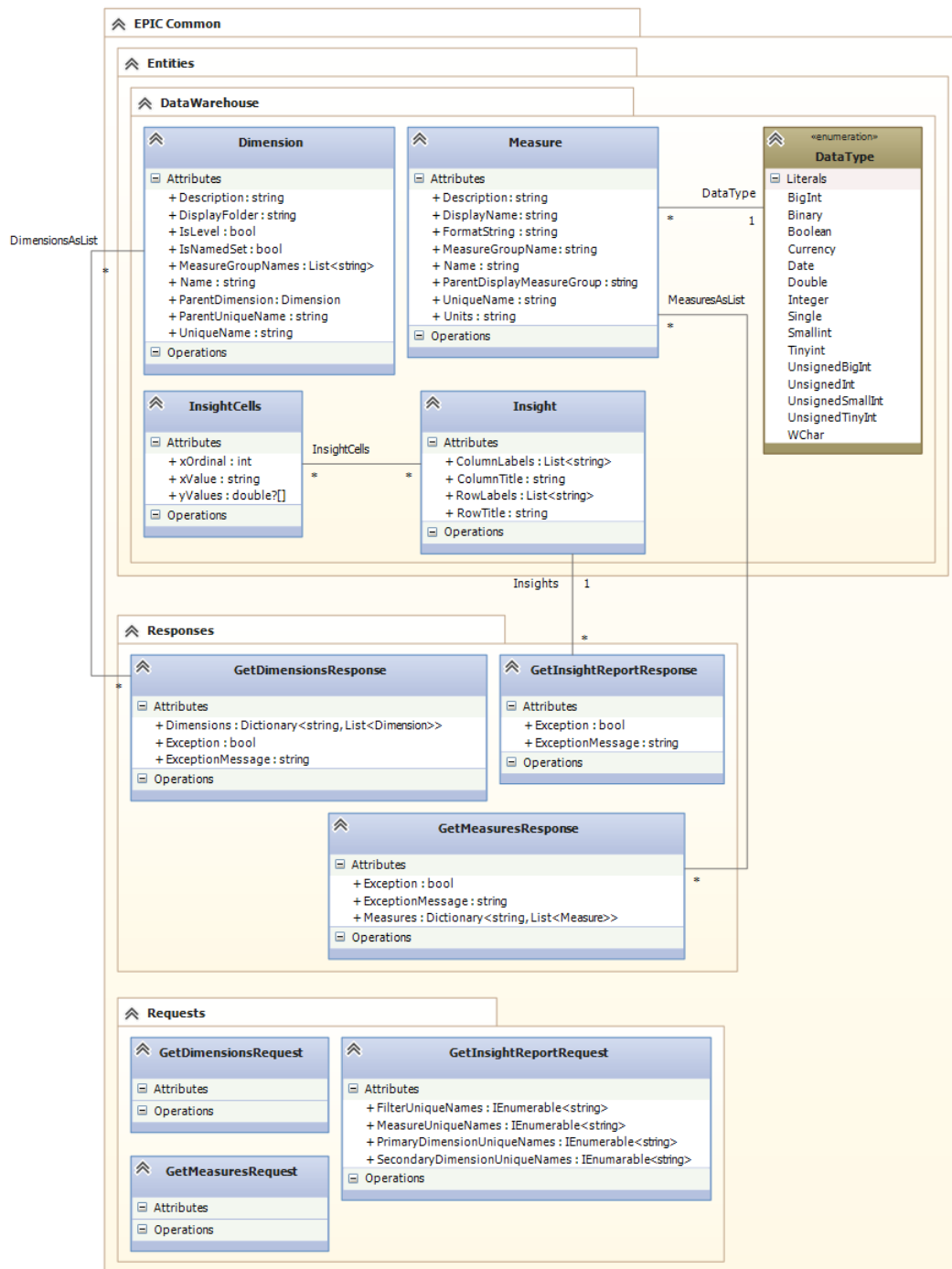


Figure 7.4.5: BI application data model

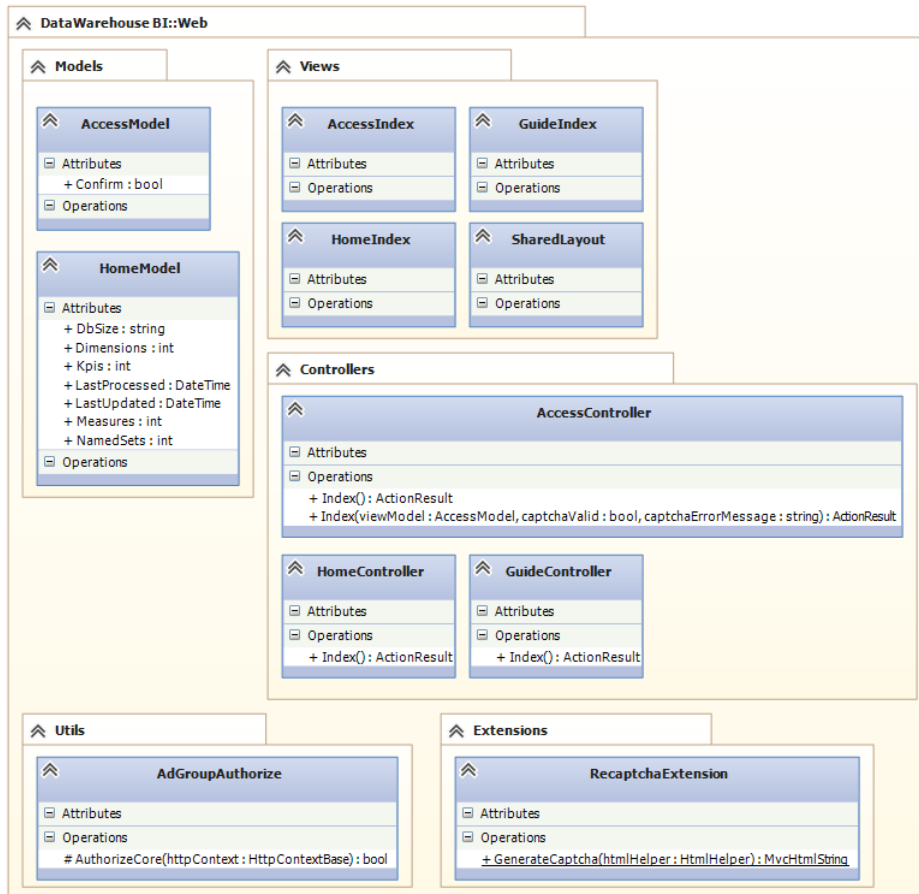
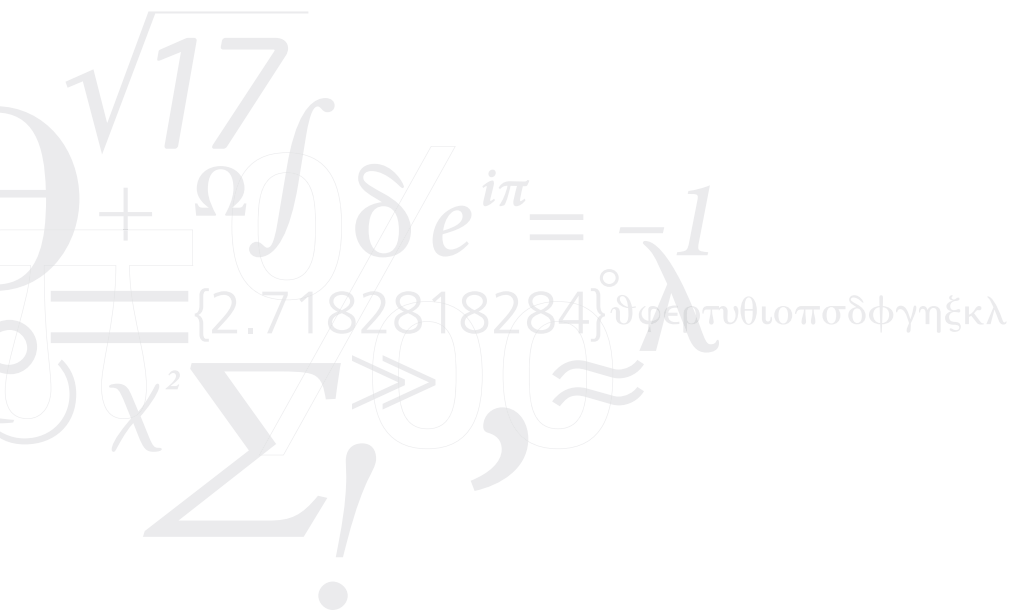


Figure 7.4.6: BI website classes

This page intentionally left blank



Chapter 8

Evaluation

”Testing shows the presence, not the absence of bugs”
—Edsger W. Dijkstra, 1969

Contents

8.1	Integration tests	141
8.2	System tests	148
8.3	Acceptance tests	151
8.4	BI reports	156

This chapter provides the evaluation and test of the system. The tests range from integration testing to acceptance testing.

8.1 Integration tests

The integration tests considers the integration of the individual components of the DW/BI solution. The integration tests below are done through black-box testing.

8.1.1 Data Collection Service

The Data Collection Service is tested by sending response messages in the P2P network, followed by checking the database tables whether the corresponding new rows are created.

Test	Description	Passed
Response to NumericRecords (int), ReadResponse	When a ReadResponse is received a new NumericRecord entity, with RecordType set to RecordType.ReadResponse should be created in the database.	✓
Response to NumericRecords (int), UpdateResponse	When an UpdateResponse is received a new NumericRecord entity, with RecordType set to RecordType.UpdateResponse should be created in the database.	✓

Response to NumericRecords (int), UpdateEvent	When an UpdateEvent is received a new NumericRecord entity, with RecordType set to RecordType.UpdateEvent should be created in the database.	✓
Response to NumericRecords (short)	When a response contains a short integer, both the DoubleValue and LongValue should be set in the row.	✓
Response to NumericRecords (int)	When a response contains an integer, both the DoubleValue and LongValue should be set in the row.	✓
Response to NumericRecords (long)	When a response contains a long integer, both the DoubleValue and LongValue should be set in the row.	✓
Response to NumericRecords (float)	When a response contains a float, the DoubleValue should be set in the row.	✓
Response to NumericRecords (double)	When a response contains a double, the DoubleValue should be set in the row.	✓
Response to NumericRecords (decimal)	When a response contains a decimal, the DoubleValue should be set in the row.	✓
Response to NumericRecords (double) epsilon larger than 0.01	When a response contains a double, with an epsilon larger than 0.01, only the DoubleValue should be set in the row. The LongValue should be null.	✓
Response to NumericRecords (double) epsilon smaller than 0.01	When a response contains a double, with an epsilon smaller than 0.01, both the DoubleValue and LongValue should be set in the row. The DoubleValue should contain all the decimals, while the LongValue contains a truncated DoubleValue.	✓
Response to BoolRecords (true)	When a response contains a boolean, which is true, a new BoolRecord entity should be created in BoolRecords with a true BoolValue.	✓
Response to BoolRecords (false)	When a response contains a boolean, which is false, a new BoolRecord entity should be created in BoolRecords with a false BoolValue.	✓

8.1. Integration tests

Response to ErrorRecords (Item = null)	When a response contains a null Item, a new ErrorRecord entity should be created in ErrorRecords with the Message "item is null"	✓
Response to ErrorRecords (Timestamp not set)	When a response does not contain a Sent timestamp, a new ErrorRecord entity should be created in ErrorRecords with the Message "timestamp not set" and with the XmlValue set to the value inside the Item.	✓
Response to ErrorRecords (Value = null)	When a response does not contains a null Value inside the Item, a new ErrorRecord entity should be created in ErrorRecords with the Message "value is null".	✓
UpdateEvent filtering	When multiple UpdateEvents are received within the same second with the same ItemId and Value, only one record should be created.	✓
Power meter pulse counter filtering	When power meter pulse counter items are received, only the ones where the value is above zero, should be created as a row.	✓

Table 8.1.1: Integration tests for Data Collection Service

8.1.2 Data Collection Helper Service

The Data Collection Helper Service is tested by letting the helper service send request messages in the P2P network to the HSI system, followed by checking the database tables whether the corresponding new rows are created by the Data Collection Service. And checking whether the counter was reset.

Test	Description	Passed
Request reading of power meter counter item, count is above zero	When the Helper service requests a reading of a power meter counter item which has a count above zero, the reading should be saved in the database as a new entity in the NumericRecords, and the counter should be reset.	✓

Request reading of power meter counter item, count is zero	When the Helper service requests a reading of a power meter counter item which has a zero count, the reading should not be saved in the database, and the counter should not be reset.	✓
--	--	---

Table 8.1.2: Integration tests for Data Collection Helper Service

8.1.3 ETL

The ETL system is tested by letting the ETL process each of the processing steps individually, the result in the destination database (the data warehouse) is then checked to see whether the corresponding rows are created, edited or deleted.

8.1.3.1 Date and Time

Test	Description	Passed
Date dimension populated with rows for the next 4 years	The date dimension should be populated with a row for each day of the next 4 years from January 1st, 2014 to December 31st, 2017, which is a total of 1461 rows.	✓
Subsequent processing of date	The date dimension should not be populated with any more rows for subsequent processing of the date dimension, as long as it is within the same year. The next year should first be loaded after December 31st.	✓
Time of day dimension populated with rows for one day	The time of day dimension should be populated with a row for each second of one day, which is a total of 86400 rows.	✓
Subsequent processing of time	The time of day dimension should not be populated with any more rows after the first processing.	✓

Table 8.1.3: Integration tests for ETL Data and Time

8.1.3.2 HSI

Test	Description	Passed
Items in Device dimension	All HSI Items should be in the Device dimension	✓

8.1. Integration tests

Updated HSI Items in Device dimension	When a HSI Item is updated, e.g. the Name, the Device entity is updated in the Device dimension.	✓
Deleted HSI Items in Device dimension	When a HSI Item is deleted, the corresponding Device entity should not be deleted in the Device dimension.	✓
Houses in Device dimension	All Device dimensions should have the HSI House related to the item.	✓
Updated HSI Houses in Device dimension	When a HSI House is updated, e.g. the Name, the HouseName is updated in the Device dimension.	✓
Floors in Device dimension	All Device dimensions should have the HSI Floor related to the item.	✓
Updated HSI Floors in Device dimension	When a HSI Floor is updated, e.g. the Name, the FloorName is updated in the Device dimension.	✓
Groups in Grouping dimension	All HSI Groups should be in the Grouping dimension	✓
Updated HSI Groups in Grouping dimension	When a HSI Group is updated, e.g. the Name, the Grouping entity is updated in the Grouping dimension.	✓
Deleted HSI Groups in Grouping dimension	When a HSI Group is deleted, the corresponding Grouping entity should not be deleted in the Grouping dimension.	✓
States in Switch Button dimension	All HSI States used by bool items loaded in the Device should be in the Switch Button dimension	✓
Updated HSI States in Switch Button dimension	When a HSI State is updated, e.g. the Name, the Switch Button entity is updated in the Switch Button dimension.	✓
Deleted HSI States in Switch Button dimension	When a HSI State is deleted, the corresponding Switch Button entity should not be deleted in the Switch Button dimension.	✓
Group relation in FactDeviceGroup	All HSI Group relations should be loaded in to the FactDeviceGroup.	✓
Deleted group relation in FactDeviceGroup	All deleted HSI Group relations should be deleted in to the FactDeviceGroup.	✓

Table 8.1.4: Integration tests for ETL HSI

Note: If a HSI House or Floor is deleted the related items will no longer have a house or floor, hence this is handled the same way as with deleted items, i.e. that the Device entity is kept.

8.1.3.3 Weather

Test	Description	Passed
Weather forecast for locations	A weather forecast should be loaded into the <code>FactWeather</code> table for each location defined in the metadata.	✓
Weather forecast dimension	A combination of the descriptive context should be loaded into the <code>DimWeather</code> table for the weather forecast.	✓
Subsequent weather forecast processing	Only a single row should be loaded for each forecast, if a subsequent processing is done when there is no new forecast available, there should not be loaded any more rows.	✓

Table 8.1.5: Integration tests for ETL Weather

8.1.3.4 SDE measurements

Test	Description	Passed
All measurements	All the measurements should be loaded into the respective fact tables.	✓

Table 8.1.6: Integration tests for ETL SDE

Note: Sample data points (measurements) have been cross-checked with the corresponding JSON file.

8.1.3.5 Data Collection

Test	Description	Passed
Numeric records	A single <code>NumericRecord</code> row should be loaded into the determined fact table, based on the <code>SourceRowIdentifier</code> , as a single fact row.	✓
Numeric records, duration	A single <code>NumericRecord</code> row, which have a different value than the preceding loaded row for the same <code>Item</code> , should generate two fact rows, one with the duration and one and the value of the preceding row, and one fact row with the new value and a zero duration set.	✓
Bool records	A single <code>BoolRecord</code> row should be loaded into the determined fact table, based on the <code>SourceRowIdentifier</code> , as a single fact row.	✓

8.1. Integration tests

Bool records, duration		A single BoolRecord row, which have the opposite value of the preceding loaded row for the same Item, should generate two fact rows, one with the duration and one and the value of the preceding row, and one fact row with the new value and a zero duration set.	✓
Bool records, no SourceRowIdentifier		If a BoolRecord entity has not been matched with a fact table, the row should be sent to the UnmappedGenericFacts table.	✓
Numeric records, SourceRowIdentifier	no	If a NumericRecord entity has not been matched with a fact table, the row should be sent to the UnmappedGenericFacts table, with a "Unable to determine relation to fact table" message .	✓
Numeric records above max value		If a NumericRecord entity has a value above the defined MaxValue in the HSI system, the row should be sent to the UnmappedGenericFacts table, with a "The value is above max value" message.	✓
Numeric records below min value		If a NumericRecord entity has a value below the defined MinValue in the HSI system, the row should be sent to the UnmappedGenericFacts table, with a "The value is below min value" message.	✓
Records ignored		If a NumericRecord or BoolRecord is matched to an Ignore using the SourceRowIdentifier, the row should be sent to the UnmappedGenericFacts table, with a "Measurement ignored by relation filter." message.	✓

Table 8.1.7: Integration tests for ETL Data Collection

8.1.3.6 Cube

Test	Description	Passed
Process cube	When a process cube query is sent, the cube should be processed.	✓

Table 8.1.8: Integration tests for SSAS cube

Note: The cube processing is tested by editing e.g. the name of a device, when the cube has been processed, the new name should appear.

8.1.4 BI application

The BI application is tested by using the App service.

Test	Description	Passed
Get dimensions	All dimensions defined in the SSAS cube should be returned as a list.	✓
Get measures	All measures defined in the SSAS cube should be returned as a list.	✓
Get report (one primary dimension, no secondary dimensions, one measure, no filters)	A generated report should be returned with a number of x-axis (primary dimension rows) and one y-value for each (measure).	✓
Get report (one primary dimension, no secondary dimensions, one measure, one filter)	A generated report should be returned with a number of x-axis (primary dimension rows) and one y-value for each (measure), the x-axis is only a subset.	✓
Get report (one primary dimension, one secondary dimension, one measure, no filters)	A generated report should be returned with a number of x-axis (primary dimension rows) and multiple y-value for each (one measure for each secondary dimension row).	✓
Get report (one primary dimension, no secondary dimensions, two measures, no filters)	A generated report should be returned with a number of x-axis (primary dimension rows) and two y-value for each (for each measure).	✓
Get report (one primary dimension, one secondary dimension, two measures, no filters)	A generated report should be returned with a number of x-axis (primary dimension rows) and two y-value for each (for each measure). This should return the same as the above test, because a secondary dimension cannot be defined if multiple measures are defined.	✓

Table 8.1.9: Integration tests for BI application

8.2 System tests

The system tests validates the entire system as a whole. It mostly validates whether the functional requirements are met. The system tests are done through black-box testing.

The functional requirements listed in the Analysis chapter is tested using the solution made. The following has been tested using the OLAP cube with the analysis tool Excel. Note that these BI reports can just as

8.2. System tests

will be made using the BI application e.g. through the App service.

8.2.1 Gain knowledge about the functioning of the house

Test	Description	Passed
Gain knowledge about energy consumption and the context in which it is consumed	Make a BI report with the energy consumption and use e.g. the dimensions Device and Time	✓
Gain knowledge about energy production and the context in which it is produced	Make a BI report with the energy production with the dimensions Date and Time together with the weather facts	✓
Gain knowledge about how the devices in the house are used and how long they are in different states	Make a BI report with e.g. the light switch duration measure and use the dimensions State and Time	✓
Gain knowledge about the temperature, humidity and other indoor comfort conditions which might affect the usage of the house and the context for these	Make a BI report with e.g. the Indoor temperature average measure and use the dimensions Date and Time	✓

Table 8.2.1: System tests for gain knowledge about the functioning of the house

8.2.2 See how elements relate

Test	Description	Passed
Seeing differences in curves in a graph for elements in a house	Add a PivotChart to one of the above made BI reports	✓
Seeing similarities in curves in a graph for elements in a house	Add a PivotChart to one of the above made BI reports	✓
Ability to compare graphs to further analyse relationships	Add two PivotCharts for two of the above made BI reports	✓

Table 8.2.2: System tests for see how elements relate

8.2.3 See how elements of the house and outside conditions relate

Test	Description	Passed
Ability to see outside conditions like temperature and cloud cover	Make a BI report with the measures from the Weather fact table	✓
Ability to compare graphs from outside conditions with indoor comfort conditions	Make a BI report with the measures from the Weather fact table and add e.g. the indoor temperature average measure.	✓

Table 8.2.3: System tests for see how elements of the house and outside conditions relate

8.2.4 Perceive the numbers as tangible elements

Test	Description	Passed
Ability to easily add graphs to be generated in reports, instead of showing raw numbers	Add a PivotChart	✓
Ability to summarise as many numbers as possible, so no additional calculations are needed to get the full picture	Dragging measures down to the Values area of a PivotTable, automatically gives summarised data	✓
Ability to change formatting of numbers so they seem familiar to the occupants	This either be done through the cube set up or by using formatting in Excel.	✓
Ensure understandability of the data model, e.g. by using a tangible object like a cube.	Since the data is in a dimensional model, the data model can be taught of as a cube.	✓

Table 8.2.4: System tests for perceive the numbers as tangible elements

8.2.5 Drilling through data to analyse it

Test	Description	Passed
Ability to summarise data from a hierarchy level above the current summarised view (Roll up)	In a BI report, remove one of the dimensions or roll up a level for the current dimension to roll up the summarised data.	✓
Ability to summarise data from a hierarchy level below the current summarised view (Drill down)	In a BI report, add another dimensions or drill down a level for the current dimension to drill down the summarised data.	✓
Ability to summarise data from different measurements (Drill across)	Add several measures to a PivotTable in Excel, this uses drill across.	✓
Ability to drill all the way down to the raw data (Drill through)	In a PivotTable in Excel, double click on a measure value, this will drill through the data, and give the most granular data available for the summarised data.	✓

Table 8.2.5: System tests for drilling through data to analyse it

8.2.6 Slice and dice data cubes

Test	Description	Passed
Ability to specify the view for specific facts	Add a measure to a PivotTable in Excel.	✓

8.3. Acceptance tests

Ability to specify the view for specific dimension to see the facts	Add a dimension to a PivotTable in Excel.	✓
Ability to give numbers context by means of dimensions	By adding a dimension e.g. to the rows or filter in a PivotTable in Excel, makes it possible to find/filter on the context for the numbers.	✓

Table 8.2.6: System tests for slice and dice data cubes

8.2.7 Get relevant and valid data

Test	Description	Passed
Ability to calculate with known metrics	Use e.g. kWh and kW for energy consumption instead of the count of a power meter pulse counter.	✓
Having valid data	By drilling through the various numeric measures, it should be clear that the data is properly cleaned, with no values above the defined maximum value and no values below the defined minimum value.	✓

Table 8.2.7: System tests for get relevant and valid data

Note to the last test: the auditing of the cleaned data during the competition showed that the humidity sensors was not properly set up, since values of above 100% was received. This data does not make any sense because the humidity sensors give values of relative humidity, so humidity levels of above 100% is discarded properly and is never shown to the end-users.

8.3 Acceptance tests

The acceptance tests validates whether the user can use the system and thereby whether the non-functional requirements are met, which also includes performance testing. The acceptance tests are done through black-box testing.

8.3.1 Performance tests

Performance tests is conducted to see how well the subsystems perform and see how well the overall system performs. The performance tests are conducted on a laptop with an Intel Core i7-2670QM 2.20GHz processor and 16GB of ram. This laptop is significantly faster than the server used for the cloud, so the processing times below are significantly lower than what is achieved by the server.

8.3.1.1 Processing steps

The processing times for the individual processing steps in the ETL system is given in Table 8.3.1. The average is based on processing 5 times, each time the database is cleared. This means that the processing times are for the first processing. Subsequent processing is not measured, since this will vary depending on how data is added between the subsequent processing. Most of the processing time for the subsequent processing will be done in the SSIS package.

Test	Average	Standard deviation	Average per row	Standard deviation per row
Populating Dim-Date	2397.16 ms	91.48	1.64 ms	0.06
Populating Dim-TimeOfDay	93356.61 ms	2237.67	1.08 ms	0.03
Populating Dim-Grouping	1144.4 ms	61.97	95.37 ms	5.16
Populating DimDevice	1212 ms	81.67	9.05 ms	0.61
Populating Dim-SwitchButton	109.6 ms	4.16	4.22 ms	0.16
Populating Fact-DeviceGroup	1193.6 ms	33.15	8.91 ms	0.25
Populating FactWeather and DimWeather	1361.99 ms	225.52	454 ms	75.17
Populating FactSDEPower	20208.22 ms	673.8	5.85 ms	0.2
Populating FactSDE-RoomTemperature	23050.30 ms	884.89	6.7 ms	0.26
Populating FactSDEHumidity	28715.87 ms	1000.91	8.31 ms	0.29
Populating FactS-DEAirQuality	34767.68 ms	1499.45	10.06 ms	0.43
Populating FactSDE-FridgeFreezerTemperature	41398.94 ms	2121.82	11.98 ms	0.61

8.3. Acceptance tests

Populating FactSDE Dish-washer-Washing Machine Temperature	47342.73 ms	2436.13	13.7 ms	0.70
Populating FactSDE Oven Temperature	53283.73 ms	3536.32	15.42 ms	1.02
Populating from SSIS Package (10000 rows (25% bool, 75% numeric))	3152 ms	65.62	0.32 ms	0.01
Processing SSAS cube	14710.01 ms	3858.47	N/A	N/A

Table 8.3.1: Performance for ETL processing

Note that for the SSIS package 25% of the rows were bool while 75% was numeric, this was approximately the distribution of the data after 2 weeks of the competition (281253 numeric rows ~77%, 83869 bool rows ~23%).

Note that for the SSAS processing the number of rows is not determined, so the average per row is not calculated.

8.3.1.2 Overall processing

The overall processing takes on average: 367404 ms ~6 min for a first processing with the SSIS package processing 10000 rows.

8.3.1.3 SSIS package

To see how the SSIS package processing scales, different number of rows have been tested. Still the distribution used is 25% bool records and 75% numeric records.

In Figure 8.3.1 the processing time vs. number of rows is seen, including a linear trendline. In Figure 8.3.2 the processing time vs. number of rows is seen, including an exponential trendline. The R-squared value reveals that the linear trendline is more appropriate to model the processing times. Hence the processing time vs. number of rows has linear growth. To give an estimate on how long a subsequent processing takes, the above figures can be used.

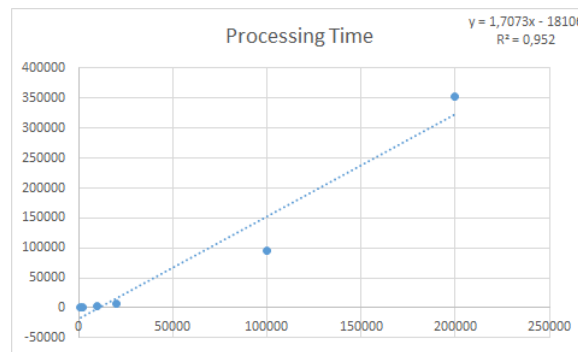


Figure 8.3.1: Processing time vs. number of rows - linear trendline

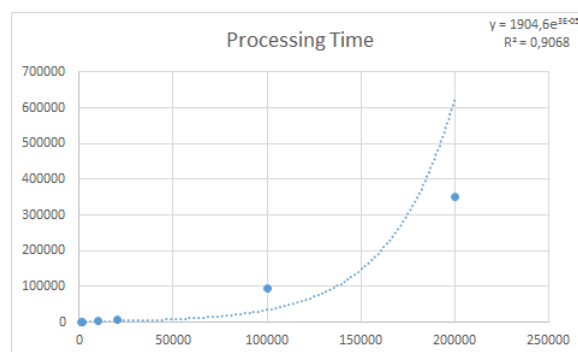


Figure 8.3.2: Processing time vs. number of rows - exponential trendline

8.3.1.4 CPU profiling

To give a better overview of the CPU usage during ETL processing, CPU profiling has been done. The analysis shows that most of the CPU is, during the first processing, used on the following methods: `ProcessSDE` (69.89%) and `ProcessDateTime` (27.49%). When looking at the individual methods it is found that the `SaveChanges` method from the `DbContext` uses the most CPU (86.81%).

Note that since the SSIS package processing is done externally from the ETL process, the processing of this is not included in the above analysis. (Only the loading of the package).

The analysis shows that most of the CPU is, during subsequent processing, used on the following methods: `ProcessSSIS` (33.64%), `ProcessDateTime` (23.34%), `ProcessSDE` (20.98%) and `ProcessHSI` (19.91%). When looking at the individual methods it is found that the `LoadPackage` used in the `SSISManager` uses the most CPU (32.66%). 20.66% is used on executing LINQ `Where` statements, and 8.91% is used on the `AddOrUpdate` method.

Note that since the SSIS package processing is done externally from the ETL process, the processing of this is not included in the above analysis. (Only the loading of the package).

8.3. Acceptance tests

8.3.1.5 BI application

The performance for the BI application is tested using the App service. The tests were done 5 times and an average was calculated. The results are given in Table 8.3.2.

Test	Average	Standard deviation
Getting dimensions	189.62 ms	11.28
Getting measures	180.42 ms	9.84
Getting measures	180.42 ms	9.84
Getting generated report with one measure and one primary dimension	142.81 ms	8.53
Getting generated report with two measures and one primary dimension	139.01 ms	6.29
Getting generated report with one measure, one primary dimension and one secondary dimension	138.61 ms	4.39

Table 8.3.2: Performance for BI application

8.3.2 User acceptance tests

Below is an example of end-user usage of the data warehouse.

The data warehouse was used during the competition to monitor the energy consumption from different power meters, because the house was not performing as well as other houses. Through a quick analysis done with the data warehouse it was apparent that most of the energy consumed at night was consumed by the pumps and the control systems. This made it possible to act quickly to reduce the amount of energy consumed by the pumps by adjusting the pumps at night and thereby get more points in the overall competition.

8.4 BI reports

The following is a number of example BI reports, most of which can be made through the insight reports from the BI application.

If an occupant wants to find out whether there is a relation between the floor, the indoor and the weather temperatures outside during a normal day, the report chart seen in Figure 8.4.1 can be used. This shows the average temperature for the floor, the indoor and outside temperatures for Versailles during a full day. The dimensions used is the time of day and to filter the Versailles weather the weather dimension is used. The measures used are the average temperatures for the floor temperature, the indoor temperature and the weather temperature. Note that this is an example of a drill across operation between two operational source systems: the Data Collection and the Weather service. In fact the data from the Data Collection comes from two different hardware systems: the Uponor system (floor temperature) and the IHC system (indoor temperature). It is seen in the figure that the indoor and floor temperatures are very slightly dependent on the outside temperature, a small drop is seen during the morning and a small rise is seen during the evening. However considering the fluctuations of the weather temperature, the indoor and floor temperatures are very stable.

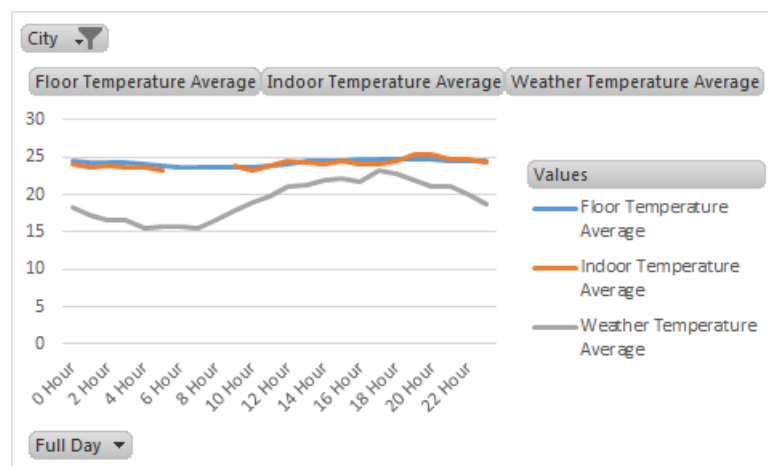


Figure 8.4.1: Average temperature for the floor, the indoor and outside temperatures for Versailles during a full day

If an occupant wants to find out whether there is a relation between the indoor humidity and the weather humidity outside during a normal day in week 27 in Versailles and check it against the humidity measurement from the SDE organisation, the report chart seen in Figure 8.4.2 can be used. This shows the average indoor, SDE and weather humidity in Versailles during week 27 for a full day. The dimensions used is the time of day, to filter the Versailles weather the weather dimension is used and to filter the week, the date dimension is also used. The measures used are the average humidity for the indoor humidity, the SDE humidity measurements and the weather humidity. Note that this is also an example of a drill across operation between three operational source systems: the Data Collection, the SDE measurements and the

Weather service. It is seen in the figure that the indoor humidity is somewhat dependent on the humidity outside.

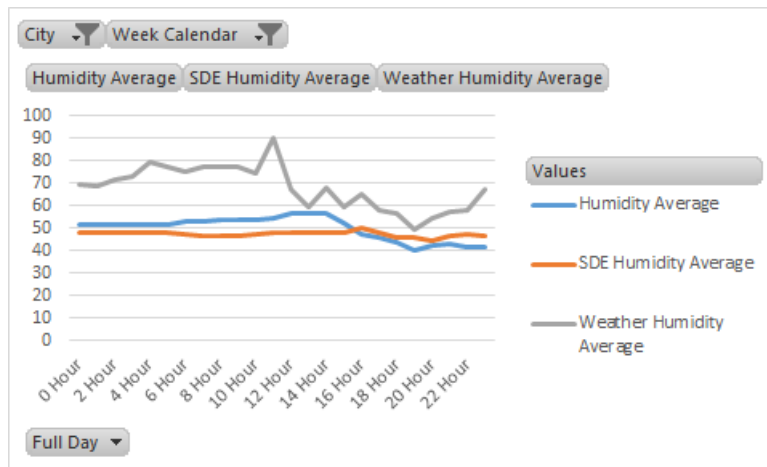


Figure 8.4.2: Average indoor, SDE and weather humidity in Versailles during week 27 for a full day

If an occupant wants to find out how many hours on average the dimmable lights are at the different percentage levels, the report chart seen in Figure 8.4.3 can be used. This shows the average dimmable light duration in hours by the percent dimension. Note that this is for all the dimmable lights in the house. It is clearly seen that the lights are used most at 62%.

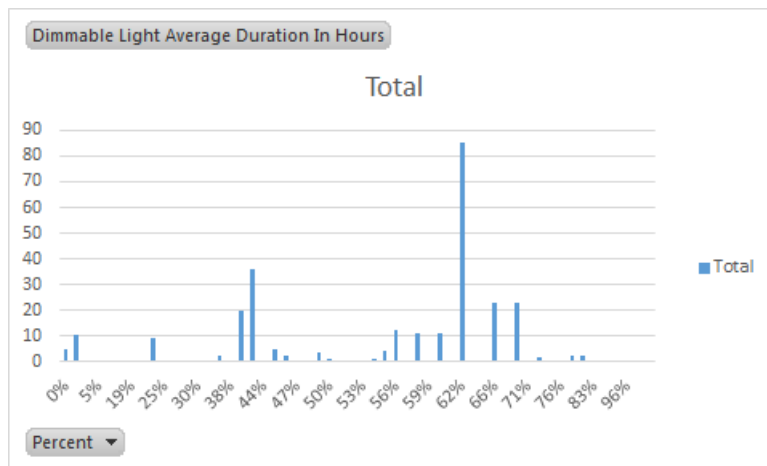


Figure 8.4.3: Average dimmable light duration in hours by percent level

If an occupant wants to find out what dimmable light level is used during a full day on average by floor, the report chart seen in Figure 8.4.4 can be used. This shows average dimmable light level measure by a full day on the time dimension and by floors on the device dimension. Note that this is for all the dimmable lights in the house. It is seen in the figure that the light level is constantly around 50% on the ground floor,

and almost always turned off on the first floor. At night however the light level on both floors are set on a high level. This is because during the competition, the all lights were required to be on during the night.

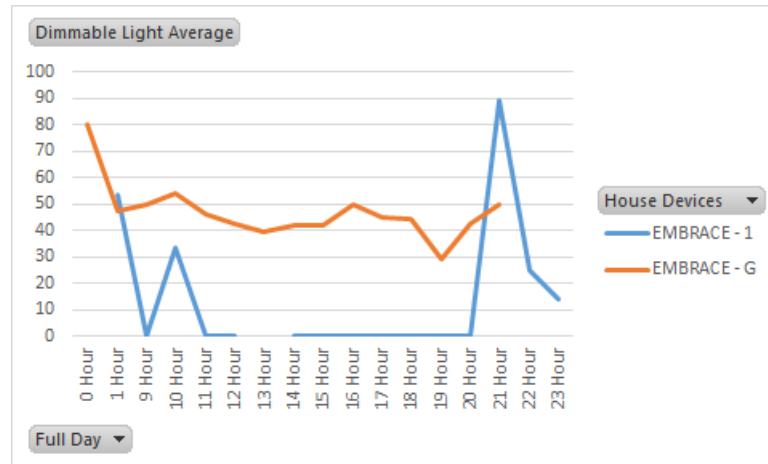


Figure 8.4.4: Average dimmable light level by a full day and by floors

If an occupant want to find out which elements of the house is consuming the most energy, the report chart seen in Figure 8.4.5 can be used. This shows the energy consumption divided by the different groups of power meters. The measure used is the energy consumption in kWh and the dimension used is the device grouping. It is clearly seen that the appliances and the climate systems consume the most energy.

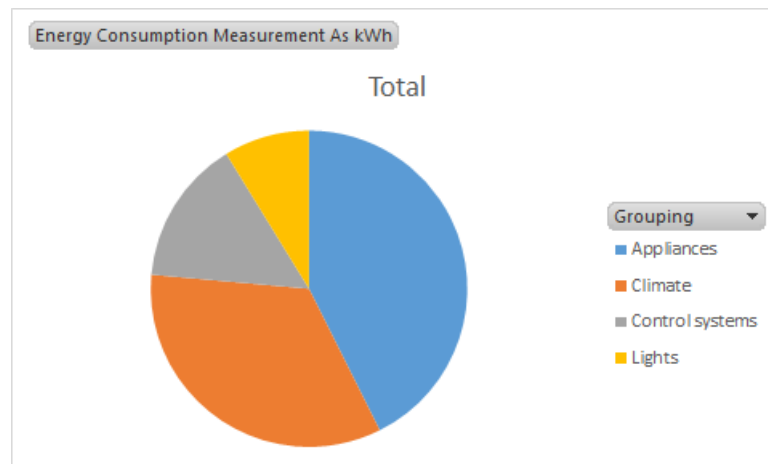


Figure 8.4.5: Energy consumption divided by the different groups of power meters

If an occupant wants to find out how much energy consumption was consumed and energy production was produced by weeks, the report chart seen in Figure 8.4.6 can be used. This shows the total amount of kWh consumed and produced by weeks. The measures used are the total energy consumption in kWh and energy production in kWh by the date dimension. It is clearly seen that the energy produced is much

higher than the energy consumed. And during week 28 the most energy was produced. This is most likely do to the fact that only week 28 contains full data. For week 27 the first days of the data was not collected and for week 29 the house was disassembled.

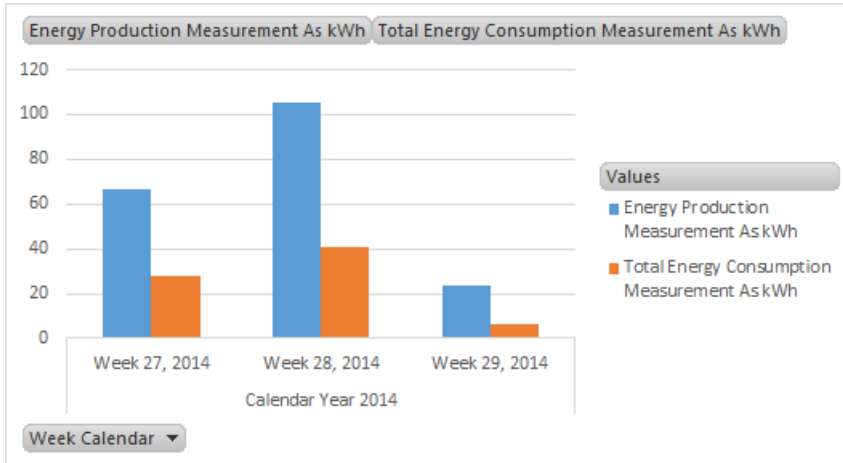


Figure 8.4.6: Total amount of kWh consumed and produced by weeks

If an occupant wants to drill a little further to find out when on a full day the energy consumption and energy production is highest, the report chart seen in Figure 8.4.7 can be used. This shows the total amount of kWh consumed and produced by the hours of a full day. The measures used are the total energy consumption in kWh and energy production in kWh by the time dimension. It is clearly seen that during the night the no energy is produced and very little energy is consumed compared to the energy produced. A spike in energy consumption and production is seen at 16 o'clock.

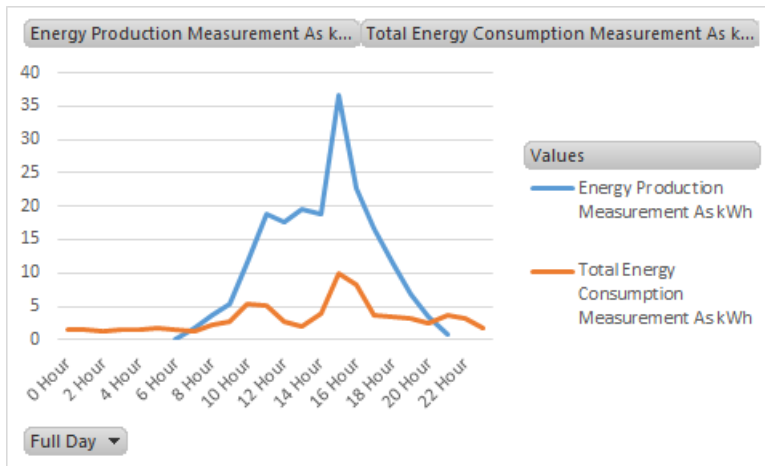


Figure 8.4.7: Total amount of kWh consumed and produced by the hours of a full day

If a researcher, who has defined a KPI, which is the energy production vs. the energy consumption, wants

to find out when during a day the ratio is below the set goal, the BI report seen in figure 8.4.8 can be used. The figure shows the KPI and a status, when the status is red, the value is below the set goal, and when the status is green the value is above the set goal. The grand total shows that the set goal is overall achieved for the KPI. It is seen from the KPI value for the grand total that the projected 185% production vs. consumption mentioned in the Context chapter is a very conservative projection, since the value was actually 259% during the competition.

Row Labels	Energy Production vs Energy Consumption KPI	Energy Production vs Energy Consumption KPI Status
⊕ 0 Hour		●
⊕ 1 Hour		●
⊕ 2 Hour		●
⊕ 3 Hour		●
⊕ 4 Hour		●
⊕ 5 Hour		●
⊕ 6 Hour	0,055594163	●
⊕ 7 Hour	1,549053356	●
⊕ 8 Hour	1,713483146	●
⊕ 9 Hour	1,935600579	●
⊕ 10 Hour	2,194889397	●
⊕ 11 Hour	3,628985507	●
⊕ 12 Hour	6,669192876	●
⊕ 13 Hour	9,928861789	●
⊕ 14 Hour	4,719298246	●
⊕ 15 Hour	3,678172487	●
⊕ 16 Hour	2,732299555	●
⊕ 17 Hour	4,509219089	●
⊕ 18 Hour	3,418701608	●
⊕ 19 Hour	2,081694402	●
⊕ 20 Hour	1,309148265	●
⊕ 21 Hour	0,187818621	●
⊕ 22 Hour		●
⊕ 23 Hour		●
Grand Total	2,592867756	●

Figure 8.4.8: KPI value and status for energy production vs. energy consumption

Chapter 9

Conclusion

This chapter concludes the thesis by drawing conclusions from the developed data warehouse and Business Intelligence solution.

The principal research question of this thesis was how to give the occupants and other subsystems of an EMBRACE house the insights of the functioning of the house, by measurements done by the control systems.

This question was based on the project goals, of which the main goal was inherited by the EMBRACE project, which was to save energy. The project goals decomposed from this were to provide insights by analysing data through finding interesting relations, drilling through data, slicing and dicing data cubes and the analysis should be intuitive and simple.

The EMBRACE house and the Solar Decathlon Europe 2014 competition was the context for this thesis. The proposed solution was to build a data warehouse and BI solution which was based on an OLAP cube and a star schema dimensional model. A BI application was also proposed for reporting insights to occupants.

The data warehouse was built using an ETL system which integrated four different operational source systems. These systems were the Hardware System Integration, Data Collection, Weather Data, and SDE Measurements. All the data from these systems were put into a star schema dimensional model, with multiple fact tables and dimension tables. The ETL system consisted of two types of ETL processing, one which was based on Entity Framework and another which was based on SSIS. The SSIS processing outperforms the EF-based ETL, but the EF-based ETL was more flexible.

The data warehouse was exposed through the OLAP system SSAS, which made query performance much better by pre-calculating aggregates and it provided hierarchical intelligence for the dimensions.

The OLAP cube was quickly set up because of the foundation, which was a star schema dimensional model. This meant that the dimensions and measures in the cube had a one-to-one relation with the dimension and fact tables in the star schema.

A BI application was made to make it possible for other subsystems in the cloud-based control system to get generated insight reports. This was mainly used by the App service, which is connected to a tablet application, which can visualise the summarised numbers from the insight reports.

The BI application also consisted of a relay, which made it possible to use the cube through the internet

and use it in analysis tools like Excel.

The final implementation was evaluated through a number of integration tests done through black-box testing, which tested the Data Collection service, the ETL system and the BI application. And the integration between these systems. All the integration tests succeeded, which means that the components of the DW/BI solution and the integration between these works as intended. To test the functional requirements of the DW/BI solution a number of system tests was done through black-box testing. It was shown that all the functional requirements are met for the DW/BI solution implemented. Which means that it is possible to gain knowledge about the functioning of the house, see how elements relate, see how elements of the house and outside conditions relate, perceive numbers as tangible elements, drilling through data to analyse it, slice and dice data cubes and get relevant and valid data. Besides system testing, acceptance testing was also conducted, which included performance testing. These tests showed that the processing times set in the non-functional requirements are mostly met by the DW/BI solution, especially by the SSIS package. It also showed that the SSIS package processing depends linearly on the number of records processed. CPU profiling of the ETL system was also done and it showed that during the first processing most CPU was used to process the SDE measurements and the date and time dimensions. For subsequent processing the loading of the SSIS package (not execution) uses the most CPU. It was noted that the execution of the SSIS package processing was not part of the CPU profiling.

A user acceptance test was also done during the competition, where the data warehouse and BI application was used to get insights about energy consumption. The analysis of the insights resulted in immediate actions to reduce the energy consumption. This showed a user acceptance toward the use of the DW/BI solution.

Lastly to evaluate the DW/BI solution a number of BI reports were made, which showed that valuable knowledge can be gained from the insights provided by the DW/BI solution.

Data warehousing is a concept of delivering data to end-users when and how they want it. This is usually the foundation for delivering Business Intelligence. The data warehouse makes it possible to get relevant data and the context for it by using dimensional modelling, which is the best way to model a data warehouse.

The thesis has shown that because BI solutions are quite generic it is actually possible to build a data warehouse and BI solution for an intelligent house context.

The typical components of a data warehouse and BI solution is access to a number of operational source systems, an ETL system, a presentation area (referred to as a data warehouse), and BI applications to analyse the data.

The data is loaded into a presentation area using the ETL system from the operational source systems. The BI application uses the data warehouse in the presentation area.

Data is often presented using dimensions and facts through an OLAP cube which is consumed by BI applications. This is the most efficient, simple and intuitive way to present data to end-users, which can imagine a tangible cube of data, consisting of dimensions and facts.

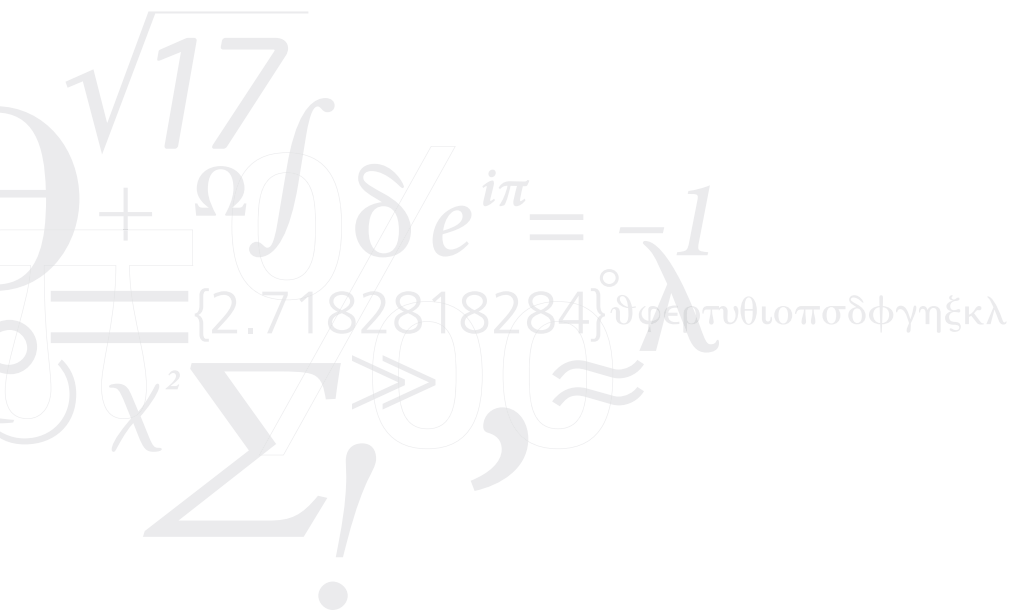
To make an efficient ETL system, specific ETL tools can be used, e.g. SSIS.

BI reporting can be used to easily gain knowledge by giving an outline and graphical view of the house

functioning and performance.

Retuning to the principal research question, the answer must be that, it is possible to provide insights of the functioning of the house to occupants and subsystems by using a data warehouse and a BI solution. The DW/BI solution integrates multiple operational source systems through an ETL system and exposes the data in a dimensional model through an OLAP cube, which can be used by a BI application. This allows occupants and subsystems to find interesting relations, drilling through data, slicing and dicing data cubes, which is done through an intuitive and simple way of analysing insights. Insights which are provided to help gain awareness of the energy consumption and thereby save energy. Which was the main goal.

This page intentionally left blank



Appendix A

Third Normal Form

A.1 First normal form

The first normal form rule considers the elimination of repeating groups or items in a single table, these items should be put into a separate table for each group of related items. Then the new items in the separate table should be identified by the primary key of the related item. Databases not adhering to the first normal form can usually be identified by the use of numbers after a column name, e.g. Device1, Device2. It can also be identified by the use of a one-to-many relationship, where the one and many side is in the same table. So instead of having multiple columns in a table for the same group of data, these should be put into a new table. E.g. instead of having a house table with multiple columns for each device, the devices should be put into a separate table. If this rule is not adhered it would mean that, when a new device should be introduced, the device would have new column in the house table, which would require database and application modifications. Following this rule offers the ability for the data to be added and removed more dynamically.

House	Device1	Device2	Device3
Embrace	PIR	Light	Power meter

Table A.1.1: Denormalised table

HouseId	House	Device
1	Embrace	PIR
1	Embrace	Light
1	Embrace	Power meter

Table A.1.2: 1NF normalised table

A.2 Second normal form

The second normal form rule considers the sets of values which apply to multiple records and multiple tables. The values which are used in more than one table or record, should be moved to a separate table or

stay in a single table, where they can be referred to by a foreign key. Databases not adhering to the second normal form can usually be identified by same values set into multiple tables. So instead of having the same values in columns for multiple tables, these values should be put into a separate table. E.g. instead of having the house name stored in both the house table and a device table. And where the house name might also be used in other tables like the recorded measurements from a sensor device. This information should be kept in the house table or a separate table for this information, and the device and measurement tables should refer to this house information using a foreign key. This offers the ability to get all houses from a defined device.

Id	House	System
1	Embrace	IHC
1	Embrace	PLC

Table A.2.1: 1NF normalised table for House entities

Id	House	Device	System
1	Embrace	PIR	IHC
2	Embrace	Light	IHC
3	Embrace	Power meter	PLC

Table A.2.2: 1NF normalised table for Device entities

Id	House	System
1	Embrace	IHC
2	Embrace	PLC

Table A.2.3: 2NF normalised table for House entities

Id	Device	HouseId
1	PIR	1
2	Light	1
3	Power meter	2

Table A.2.4: 2NF normalised table for Device entities

A.3 Third normal form

The third normal form rule considers the elimination of fields which are not dependent on the key. This means that values which do not functionally belong to the other fields in the table, would be moved to a separate table, this is done for two reasons: first it means that it is possible to identify records in a table based on the values, second it means that updating the values is only required once instead of multiple times in the same table. Databases not adhering to the third normal form can usually be identified by

values put into a table where the value is not dependent on the primary key of that table, but may be dependent on a primary key in another table. So instead of having values which are dependent on other keys than the primary key, these values should be moved to the table for their primary key. E.g. instead of having the system in the house table, the system which is not dependent on the house primary key, but the device primary key, should be moved to the device table. This offers the ability to get all houses where the device with the system variable is defined. And it ensures that the system is only needed to be updated once instead of multiple times in the house table. Adhering to the the third normal form is best practice, however in some applications it might not be practical to put all values which are used in more than one table into separate tables, because this might produce a large number of very small tables, which performance wise is not advisable. However if the values in these small tables change frequently, the should stay in a separate table.

Id	House	System
1	Embrace	IHC
2	Embrace	PLC

Table A.3.1: 2NF normalised table for House entities

Id	Device	HouseId
1	PIR	1
2	Light	1
3	Power meter	2

Table A.3.2: 2NF normalised table for Device entities

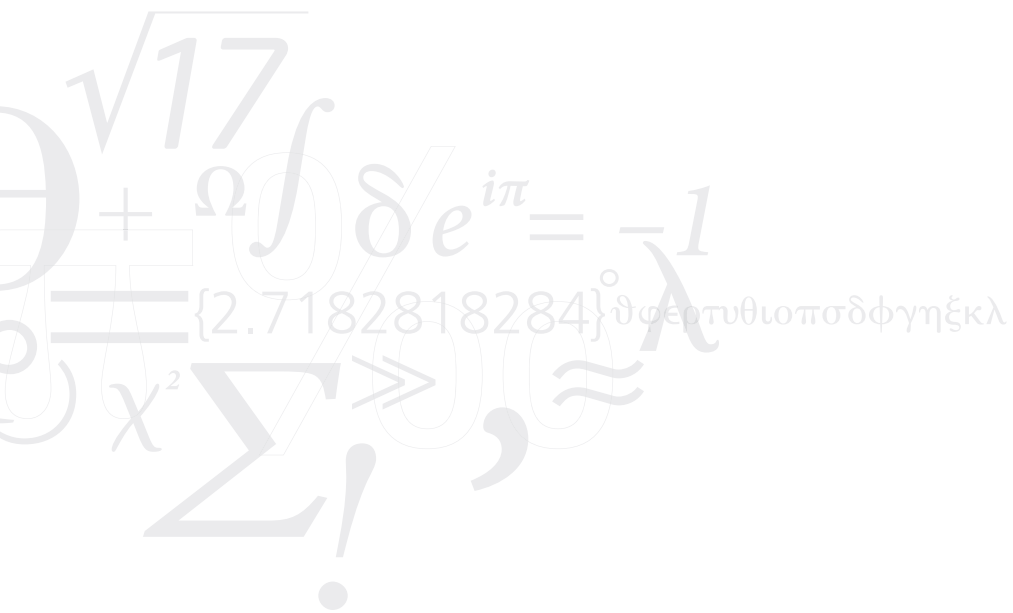
Id	House
1	Embrace

Table A.3.3: 3NF normalised table for House entities

Id	Device	System	HouseId
1	PIR	IHC	1
2	Light	IHC	1
3	Power meter	PLC	1

Table A.3.4: 3NF normalised table for Device entities

This page intentionally left blank



Appendix **B**

Additional dimensional modelling techniques

B.1 Facts

Additional dimensional modelling techniques for fact tables are given below.

B.1.1 Textual measures

Textual string values in fact tables should be avoided since most BI applications cannot do any proper analysis on these. Instead if the textual values are from a discrete range, these values should be put into a dimension table with a reference to this from the fact table. Then most BI applications can do way more, e.g. slice and dice a cube based on these values and thereby summarise data in the fact tables. Textual string values which are in free-form format, e.g. comments should be modelled in other ways, for a BI application to make any proper analysis. This could e.g. be to keep a count of the number of occurrences of a word from a discrete range.

B.1.2 Fact table types

B.1.2.1 Periodic snapshot based

Periodic snapshot fact tables are summaries of several events, e.g. from a transaction fact table. These summaries are done in a specified periodicity, this could e.g. be by day, week, year, etc. the grain is then defined by this period. Since these fact tables are for summaries, they usually include several facts or measurements. And since they are done in the specified frequency they will have a row for each tick, hence the fact tables tends to contain a vast amount of rows. Even if measurements where not done, then these will just be zero or null values. The periodic snapshot fact tables are usually calculated by the ETL, since these measurement values might not be in the operational source system. The number of rows sometimes means that a separate periodic fact table is created to hold data at a reduced frequency, and keep this after

a certain period of time of the first table. E.g. instead of keeping the fact table rows for years with data from a daily periodicity, the last three months could be kept in that fact table and a new fact table is then used to keep data with a quarterly periodicity.

B.1.2.2 Accumulating snapshot based

The last type of fact table is the accumulating snapshot fact table which is also a summarised fact table. These fact tables have measurements for specific events and usually models steps in a flow where each column defines the step. The columns could e.g. be foreign keys to a date dimension for when the event/step occurred, or foreign keys for the state at each step. The columns are mostly foreign keys to dimensions and a few numeric values. For this type of fact table most foreign keys to dimensions might be to "not applicable" or "not occurred yet" dimension rows. The rows of an accumulating snapshot fact table will usually be instances of the process it models. These fact tables usually only contain the "lag" measurement and counts as numeric values, which is the time between one step and another, also called duration, and the count for progress e.g. for how far in the process the instance is. These fact table rows are constantly being updated e.g. by the ETL system unlike the other two types of fact tables, where they are only processed once. This is done to fill the remaining columns as the process progresses. Since this type of fact table usually have several references to a date dimension, dimension roles are often used, to uniquely identify the differences between the dates. These date dimensions will also need a to be determined and/or an "unknown" dimension row, for when the columns are not populated yet, because the step has not yet occurred or is not done. This type of fact tables can be used if a process has definite steps. It is mostly used to model pipeline processes. And it can be used to identify bottlenecks, since these can be found by the lag calculations.

B.1.3 Centipede

A centipede fact table can occur when not using a junk dimension and when a multitude of dimensions are used for a single fact table. This should be avoided and can be avoided by using a junk dimension. This usually happens when dimensions are not combined properly. The reason they are called centipede fact tables is because they have a multitude of dimension foreign keys. It is often seen when dimension hierarchies are normalised and thereby in separate dimension tables, instead of a single denormalised dimension table. It can also occur when various flags and indicators are in separate dimension tables, instead of a junk dimension. This type of fact table should be avoided, since they will end up consuming a lot of space and the users will have a hard time joining the tables, since there are so many and the query performance will also deteriorate because of the many joins needed.

B.2 Dimension

Additional dimensional modelling techniques for dimension tables are given below.

B.2.1 Snowflake schema

Sometimes dimension tables are normalised using third normal form for many-to-one relationships, which is called snow-flaking because of the increase in joins needed to get the full verbose dimension table. This normalisation causes a decrease in performance and if the dimensions are presented in this way, it will also decrease the understandability of the dimensions. So this should be avoided. The reason it is called snowflake schema is because of the relationships to many separate tables from a dimension. The same information in snowflake schema dimensions can be represented in a single flatten denormalised dimension. Normalising a dimension table is usually a waste of time since the space required for these are significantly less than what is used by fact tables. A denormalised dimension table is both easier to use since multiple dimension tables does not need to be used when querying a fact table. This also improves performance because less joins are needed for the same results.

B.2.2 Outriggers

Dimensions can refer to other dimensions, these are called outrigger dimensions. Outrigger and snowflake schema are two different types of design, however using multiple outriggers can lead to a snowflake schema design, which should be avoided. Outriggers could e.g. be a floor dimension, that is referred to by a house dimension. If possible a fact table should include both dimensions and the reference from the dimension to the outrigger should be removed. It should be noted that dimension columns in a dimension table should be constrained to only have the columns or attributes which define a specific context for the end-user. E.g. it might be reasonable to have a device dimension table with a house and floor column. But it is just as reasonable to have separate house and floor dimension tables. Depending on the usage by the end-user the one or the other can be chosen. Outrigger dimensions are often used when multiple date dimension references are needed for a single fact table. Then the outrigger dimension would include a date role dimension, i.e. a date dimension in a view with a different name. An outrigger can also be used to reuse/repurpose a dimension for multiple usages from a dimension, as well as keep information for low cardinality data for a dimension, which could distract the usage of a dimension. This low cardinality data could e.g. be information about a house location, the town, the number of inhabitants and the yearly average number of mm of rain. This type of information would distract the usage of a house dimension, so a location dimension outrigger could be made to hold this information. Because of the same effects that a snowflake schema has on a dimensional model, these outriggers should be kept to a minimum.

B.2.3 Degenerate dimensions

There are situations when a separate dimension table is not needed. These situations happen, when a dimension only has a primary key, that can be used as a dimension, but since it does not make sense to make a dimension table for these keys, they are left in the fact tables. These dimensions are referred to as degenerate dimensions. These often represent a transaction number to identify the transaction from an operational source system, these numbers can be used as dimensions, but they have no relevant context other than that number. Degenerate dimensions can often be found in dimensional models where a di-

mension has almost the same number of rows as the fact table it is related to. This could be a sign that a degenerate dimension exists in the dimension table, and should be moved to the fact table.

B.2.4 Role playing dimensions

Sometimes a dimension can be used more than once in the same fact table for different purposes, this could e.g. be the date and time dimensions where the fact table contains different values for when a light was turned on and when it was turned off. When the same dimensions are used for different context they are referred to as role-playing dimensions. The fact table will have a foreign key to a view of the same dimensions with different table and column names, this makes them separate for the end-users, even though the same dimension tables are actually being used, behind the scenes.

Appendix C

SSAS Cube

C.1 Calculation script

An excerpt of the calculation script defined in the SSAS cube is given in Listing C.1.1.

```
1 CALCULATE;
2
3 /* Average duration */
4 CREATE MEMBER CURRENTCUBE.[Measures].[Dimmable Light Average Duration
   In Seconds]
5 AS [Measures].[Dimmable Light Duration In Seconds] / [Measures].[
   Dimmable Light Count],
6 VISIBLE = 1 , ASSOCIATED_MEASURE_GROUP = 'Dimmable Light';
7
8 /* Measurement averages */
9 CREATE MEMBER CURRENTCUBE.[Measures].[Humidity Average]
10 AS [Measures].[Humidity Measurement] / [Measures].[Humidity Count],
11 VISIBLE = 1 , ASSOCIATED_MEASURE_GROUP = 'Humidity';
12 CREATE MEMBER CURRENTCUBE.[Measures].[Indoor Temperature Average]
13 AS [Measures].[Indoor Temperature Measurement] / [Measures].[Indoor
   Temperature Count],
14 VISIBLE = 1 , ASSOCIATED_MEASURE_GROUP = 'Indoor Temperature';
15
16 /* Date named sets */
17 CREATE DYNAMIC SET CURRENTCUBE.[Date - Today]
18 AS STRTOMEMBER(' [Date].[Month Calendar].[Date].&[ '+FORMAT(Now(), '
   yyyyMMdd') + ' ]');
19 CREATE DYNAMIC SET CURRENTCUBE.[Date - Yesterday]
20 AS STRTOMEMBER(' [Date].[Month Calendar].[Date].&[ '+FORMAT(Now(), '
   yyyyMMdd') + ' ]').LAG(1);
21 CREATE DYNAMIC SET CURRENTCUBE.[Date - This Month]
22 AS STRTOMEMBER(' [Date].[Month Calendar].[Month].&[ '+FORMAT(Now(), '
   yyyy ') + ' ]&[ '+FORMAT(Now(), '%M') + ' ]');
23 CREATE DYNAMIC SET CURRENTCUBE.[Date - Last Month]
24 AS STRTOMEMBER(' [Date].[Month Calendar].[Month].&[ '+FORMAT(Now(), '
   yyyy ') + ' ]&[ '+FORMAT(Now(), '%M') + ' ]').LAG(1);
```

```

25 CREATE DYNAMIC SET CURRENTCUBE.[Date - This Year]
26 AS STRTOMEMBER( '[Date].[Month Calendar].[Year].&['+FORMAT(Now(), '
      yyyy')+']' );
27 CREATE DYNAMIC SET CURRENTCUBE.[Date - Last 12 Months]
28 AS STRTOMEMBER( '[Date].[Month Calendar].[Month].&['+FORMAT(Now(), '
      yyyy')+']&['+FORMAT(Now(), '%M')+']' ).LAG(12);STRTOMEMBER( '[Date
      ].[Month Calendar].[Month].&['+FORMAT(Now(), 'yyyy')+']&['+FORMAT
      (Now(), '%M')+']' ).LAG(1);
29 CREATE DYNAMIC SET CURRENTCUBE.[Date - Last 6 Months]
30 AS STRTOMEMBER( '[Date].[Month Calendar].[Month].&['+FORMAT(Now(), '
      yyyy')+']&['+FORMAT(Now(), '%M')+']' ).LAG(6);STRTOMEMBER( '[Date
      ].[Month Calendar].[Month].&['+FORMAT(Now(), 'yyyy')+']&['+FORMAT
      (Now(), '%M')+']' ).LAG(1);
31 CREATE DYNAMIC SET CURRENTCUBE.[Date - Last 3 Months]
32 AS STRTOMEMBER( '[Date].[Month Calendar].[Month].&['+FORMAT(Now(), '
      yyyy')+']&['+FORMAT(Now(), '%M')+']' ).LAG(3);STRTOMEMBER( '[Date
      ].[Month Calendar].[Month].&['+FORMAT(Now(), 'yyyy')+']&['+FORMAT
      (Now(), '%M')+']' ).LAG(1);
33
34 /* Time named sets */
35 CREATE DYNAMIC SET CURRENTCUBE.[Time - Midnight]
36 AS [Time].[Full Day].[Hour].&[0]:[Time].[Full Day].[Hour].&[1];
37 CREATE DYNAMIC SET CURRENTCUBE.[Time - Late Night]
38 AS [Time].[Full Day].[Hour].&[1]:[Time].[Full Day].[Hour].&[5];
39 CREATE DYNAMIC SET CURRENTCUBE.[Time - Morning]
40 AS [Time].[Full Day].[Hour].&[5]:[Time].[Full Day].[Hour].&[12];
41 CREATE DYNAMIC SET CURRENTCUBE.[Time - Noon]
42 AS [Time].[Full Day].[Hour].&[12]:[Time].[Full Day].[Hour].&[13];
43 CREATE DYNAMIC SET CURRENTCUBE.[Time - Afternoon]
44 AS [Time].[Full Day].[Hour].&[13]:[Time].[Full Day].[Hour].&[18];
45 CREATE DYNAMIC SET CURRENTCUBE.[Time - Evening]
46 AS [Time].[Full Day].[Hour].&[18]:[Time].[Full Day].[Hour].&[21];
47 CREATE DYNAMIC SET CURRENTCUBE.[Time - Night]
48 AS [Time].[Full Day].[Hour].&[21]:[Time].[Full Day].[Hour].&[23];
49 CREATE DYNAMIC SET CURRENTCUBE.[Time - This Hour]
50 AS STRTOMEMBER( '[Time].[Full Day].[Hour].&['+FORMAT(Now(), 'HH')
      +']' );
51 CREATE DYNAMIC SET CURRENTCUBE.[Time - Last Hour]
52 AS STRTOMEMBER( '[Time].[Full Day].[Hour].&['+FORMAT(Now(), 'HH')
      +']' ).Lag(1);

```

Listing C.1.1: Excerpt of calculation script

C.2 Dimension usage

The dimension usage defined in the SSAS cube is seen in Figures C.2.1, C.2.2, C.2.3 and C.2.4.

Measure Groups		[n]	[n]	[n]	[n]	[n]	[n]	[n]
Dimensions		Alarm Noise	Alarm Passive Infrared	Brightness	Device Group	Dimmable Light	Energy Consumption	Energy Production
Date	Date	Date	Date	Date		Date	Date	Date
Device	Device	Device	Device	Device	Device	Device	Device	Device
Device Grouping	Device Group	Device Group	Device Group	Device Group	Grouping	Device Group	Device Group	Device Group
Percent						Percent		
State	State	State	State					
Time	Time	Time	Time		Time	Time	Time	Time
Weather								

Figure C.2.1: SSAS cube dimension usage part 1

Measure Groups		[n]	[n]	[n]	[n]	[n]	[n]	[n]
Dimensions		Floor Temperature	Humidity	Indoor Temperature	Light Switch	Maintenance	Open Door	Open Window
Date	Date	Date	Date	Date	Date	Date	Date	Date
Device	Device	Device	Device	Device	Device	Device	Device	Device
Device Grouping	Device Group	Device Group	Device Group	Device Group	Device Group	Device Group	Device Group	Device Group
Percent								
State					State	State	State	State
Time	Time	Time	Time	Time	Time	Time	Time	Time
Weather								

Figure C.2.2: SSAS cube dimension usage part 2

Measure Groups		[n]	[n]	[n]	[n]	[n]	[n]	[n]
Dimensions		Passive Infrared	Power Outlet	Total Energy Consumption	Water Pressure	Water Temperature	Weather	SDE Air Quality
Date	Date	Date	Date	Date	Date	Date	Date	Date
Device	Device	Device	Device	Device	Device	Device		
Device Grouping	Device Group	Device Group	Device Group	Device Group	Device Group	Device Group		
Percent							Percent	
State	State	State						
Time	Time	Time	Time	Time	Time	Time	Time	Time
Weather							Weather Key	

Figure C.2.3: SSAS cube dimension usage part 3

Measure Groups						
Dimensions	[u] SDE Dishwasher Washing Ma...	[u] SDE Fridge Freezer Temperat...	[u] SDE Humidity	[u] SDE Oven Temperature	[u] SDE Power	[u] SDE Room Temperature
[u] Date	Date	Date	Date	Date	Date	Date
[u] Device						
[u] Device Grouping						
[u] Percent						
[u] State						
[u] Time	Time	Time	Time	Time	Time	Time
[u] Weather						

Figure C.2.4: SSAS cube dimension usage part 3

Appendix D

ETL

D.1 Data Warehouse entities

Data warehouse entities from the ETL implementation is seen in Figures 6.3.2, D.1.1, D.1.2, D.1.3, D.1.4 and D.1.5. Figure 6.3.2 is in the Design chapter.

Bool Facts		
<p>AlarmNoiseFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>LightSwitchFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>MaintenanceFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>
<p>AlarmPassiveInfraredFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>PowerOutletFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>OpenWindowFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>
<p>BrightnessFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>OpenDoorFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>	<p>PassiveInfraredFact</p> <p>Attributes</p> <ul style="list-style-type: none"> + Date : DateDimension + DateKey : int + Device: DeviceDimension + DeviceKey : int + Duration : double + Id : long + Measurement : bool + SwitchButton : SwitchButtonDimension + SwitchButtonKey : int + Time : TimeOfDayDimension + TimeKey : int <p>Operations</p>

Figure D.1.1: Bool record entities

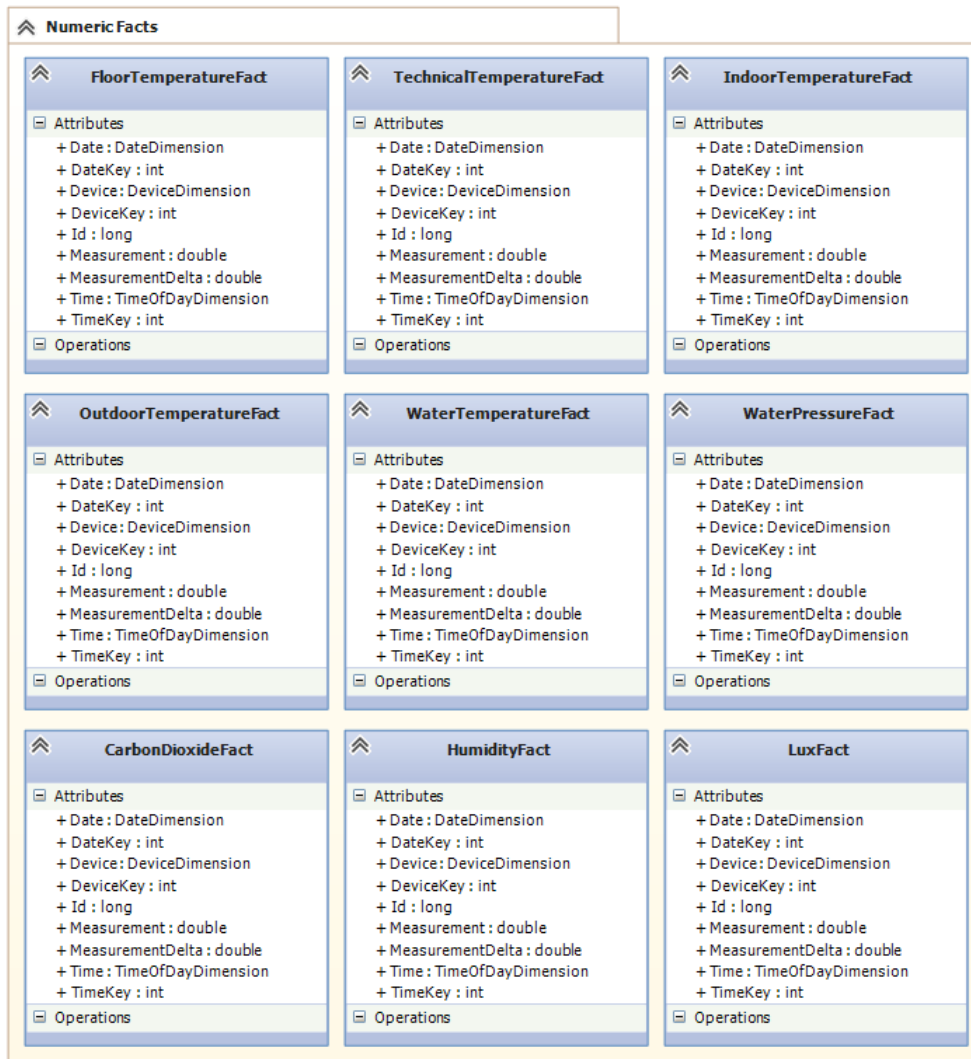


Figure D.1.2: Numeric record entities

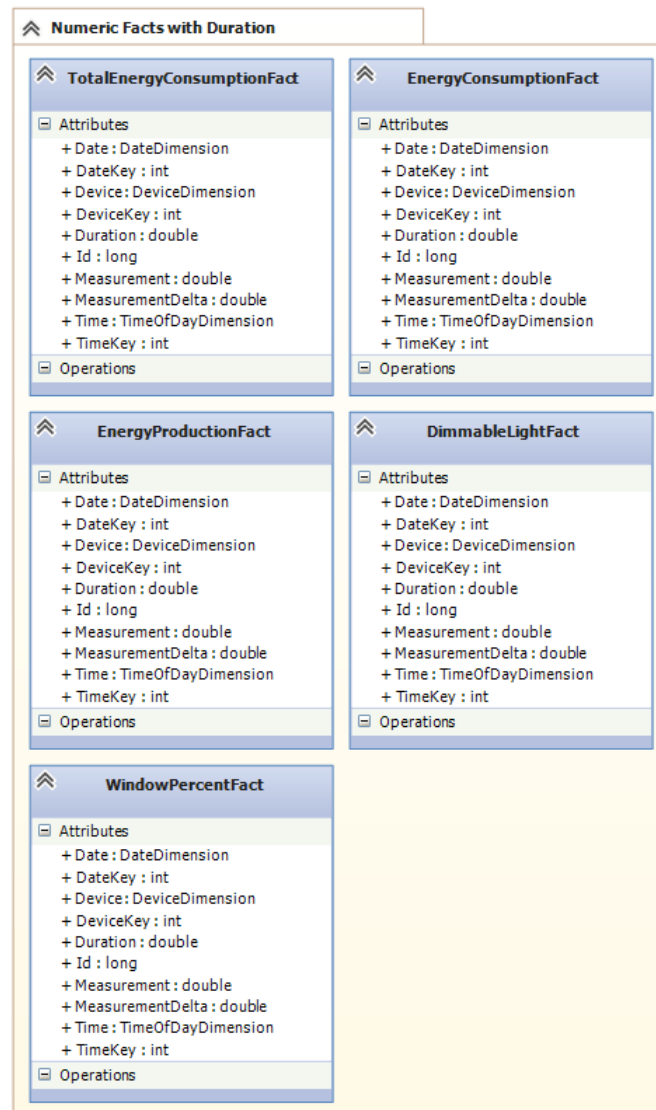


Figure D.1.3: Numeric record with duration entities

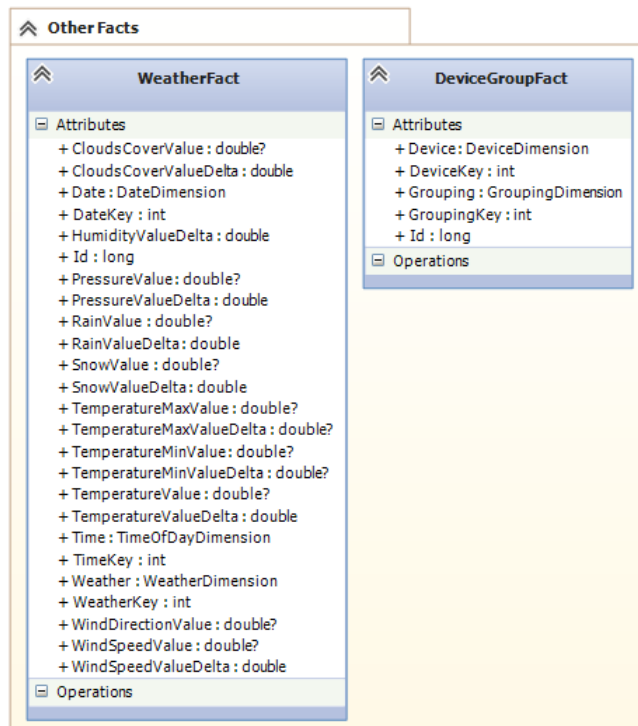


Figure D.1.4: Other fact entities

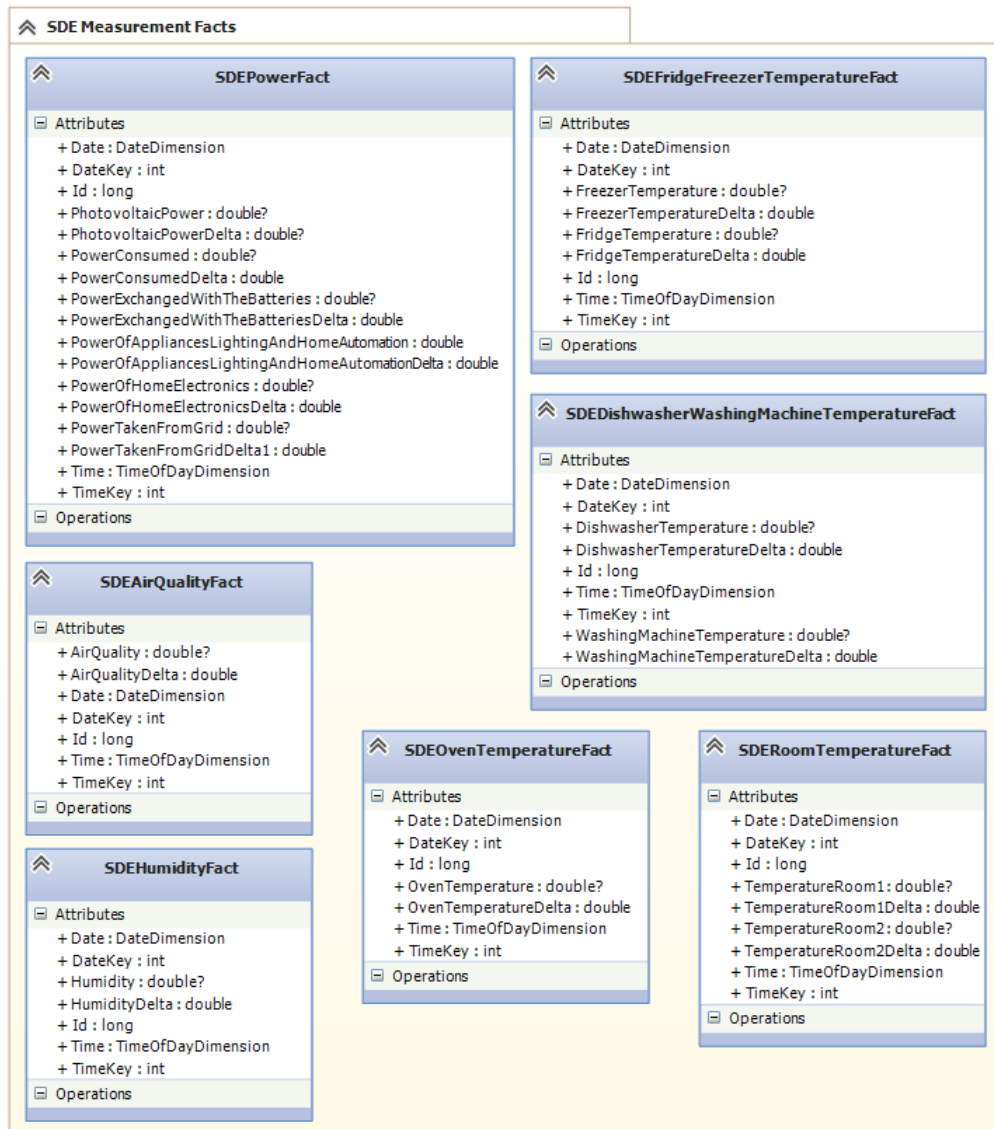


Figure D.1.5: SDE fact entities

Bibliography

- Solar Decathlon Europe 2014. Solar Decathlon Competition Web Site. *SolarDecathlon2014.fr*, 2014a. URL <http://www.solardecathlon2014.fr/en/competition>. Last visited: August 1, 2014.
- Solar Decathlon Europe 2014. Solar Decathlon Europe Web Site. *SolarDecathlon2014.fr*, 2014b. URL <http://www.solardecathlon2014.fr/en/>. Last visited: August 1, 2014.
- Valeria Anzolin and Jason Flakes. Solar Decathlon Europe. Flickr Images, 2014. URL <https://www.flickr.com/photos/sdeurope/sets/>. Last visited: August 1, 2014.
- Daniel. SSIS blocking, non blocking, and partially blocking transformations. *bidn.com*, 2010. URL <http://www.bidn.com/blogs/Daniel/ssas/1361/ssis-blocking-non-blocking-and-partially-blocking-transformations>. Last visited: August 1, 2014.
- DTU. DTU Solar Decathlon 2014 Web Site. *SolarDecathlon.dk*, 2014a. URL <http://solardecathlon.dk/wordpress/>. Last visited: August 1, 2014.
- Team DTU. Solar Decathlon Europe 2014 - Jury Report. *SDE Deliverables*, 2014b.
- Team DTU. Solar Decathlon Europe 2014 - Press Release. *SDE Deliverables*, 2014c.
- Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. John Wiley & Sons, Inc., third edition, 2013. ISBN 978-1-118-53080-1.
- Brian Larson. *Delivering Business Intelligence with Microsoft SQL Server 2012*. The McGraw-Hill Companies, third edition, 2012. ISBN 978-0-07-175938-0.
- Microsoft. Description of the database normalization basics. *Microsoft Support*, 2013. URL <http://support.microsoft.com/kb/283878>. Last visited: August 1, 2014.
- Microsoft. Multidimensional Modeling (Adventure Works Tutorial). *Microsoft MSDN*, 2014a. URL <http://msdn.microsoft.com/en-us/library/ms170208.aspx>. Last visited: August 1, 2014.
- Microsoft. Configure HTTP Access to Analysis Services on Internet Information Services (IIS) 7.0. *Microsoft MSDN*, 2014b. URL <http://msdn.microsoft.com/en-us/library/gg492140.aspx>. Last visited: August 1, 2014.
- Microsoft. SSIS Tutorial: Creating a Simple ETL Package. *Microsoft TechNet*, 2014c. URL <http://technet.microsoft.com/en-us/library/ms169917.aspx>. Last visited: August 1, 2014.

- Ryan Mulcahy. Business Intelligence Definition and Solutions. *CIO.com*, 2007. URL <http://www.cio.com/article/2439504/business-intelligence/business-intelligence-definition-and-solutions.html>. Last visited: August 1, 2014.
- U.S. Department of Energy. DOE Solar Decathlon: Solar Decathlon Europe Web Site. *SolarDecathlon.gov*, 2014. URL http://www.solardecathlon.gov/sd_europe.html. Last visited: August 1, 2014.
- Jennifer Lonoff Schiff. 8 Ways Business Intelligence Software Improves the Bottom Line. *CIO.com*, 2013. URL <http://www.cio.com/article/2384577/enterprise-software/8-ways-business-intelligence-software-improves-the-bottom-line.html>. Last visited: August 1, 2014.
- Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education Limited, first edition, 2014. ISBN 978-1-292-02615-2.