CHALMERS

Designing and implementing a web-based data warehouse solution for cost analysis

Master of Science Thesis in the Master Degree Programme Software Engineering and Technology

OSCAR HALLBERG
DAVID ERNSTSSON

Department of Computer Science and Engineering
Division of Software Engineering and Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010

Designing and implementing a web-based data warehouse solution for cost analysis

OSCAR HALLBERG
DAVID ERNSTSSON

Examiner: Sven-Arne Andreasson

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

Data warehousing is a term for the theory and techniques used for extracting, transforming and loading data from multiple sources and providing advanced analysis on the resulting information. A common representation of the data within the data warehouse is the multidimensional approach using facts and dimensions which can be designed and implemented by an OLAP solution. The facts correspond to the measurable numerical values of interest in the analysis while the dimensions are used for giving a context to the facts and built up by multiple levels within hierarchies.

During this master thesis a web application has been developed offering business intelligence analysis for telecom related invoice data. The development has been conducted within the agile Scrum methodology with two week iterations. The invoice source data has been extracted from an external system and then transformed into a more structural form which has been loaded into an OLAP cube running Microsoft SQL Server Analysis Services. Because the analyzed data is read-only within the cube techniques for pre-calculation of results across hierarchical levels of granularity are made possible which has been shown to be very effective performance wise.

This report describes different techniques and components used when designing and building the data warehouse as well as the graphical user interface developed resulting in the final business intelligence application. Different techniques for optimizing the performance are mentioned as well as main differences and comparisons with a normal relational database design.

The resulting application supports decision makers at potential customers with interesting analysis possibilities as well as providing fast responses to user requests. A comparison between the implemented multidimensional OLAP solution versus a corresponding relational database shows the response time in this case is highly significantly reduced and in this case with a factor greater than ten to one.

Keywords: OLAP, Data Warehouse, Multi-dimensional data, Business Intelligence Application, Cost Analysis, ASP.NET Framework Development

Sammanfattning

Data warehousing är ett samlingsnanm för den teori och de tekniker som används för att samla data från olika källor och utföra avancerad analys på denna data. En vanlig representation av datan i ett data warehouse är att dela upp den i fakta och dimensioner — även kallat flerdimensionell lagring — vilket kan realiseras m.h.a. en OLAP-kub. Faktadatan utgörs då av de mätvärden som analysen avser medans dimensionerna används för att gruppera och aggregera dessa mätvärden på olika nivåer.

Under examensarbetet har ett webbaserat system tagits fram som syftar till att erbjuda analysmöjligheter för telecom-relaterad faktureringdata. Arbetetet utfördes med den agila Scrum metodiken med två veckors iterationer. Faktureringsdatan hämtas från ett externt system och läses sedan in i en OLAP-kub som kör ovanpå Microsoft SQL Server Analysis Services. Genom att den analyserade datan endast skrivs till systemet periodvis så har tekniker för att i förväg kalkylera aggregererade mätvärden kunnat användas med framgång, vilket har visat sig vara mycket effektivt ur en prestandasynpunkt.

Rapporten beskriver de tekniker och metoder som har använts för att utvinna, omvandla samt modellera datan som lagras i OLAP-kuben samt utveckligen av det gränssnitt som används för analysen. Utläsningen av datan från OLAP-kuben jämförs även med motsvarande utläsning via en relationsdatabas. Resultaten visar på kraftigt förbättrad svarstid för analyser som körs mot den multidimensionella databasen jämförelsevis med relationsdatabasen.

Contents

List of Figures

1 Introduction

1.1 The problem

1.1.1 Background

Since 2001 the company Links has been developing and maintaining a web-based system for a large telecom operator. The system is called STAX and was built to help the customer's large customers (CC) to organize and administrate their MAS agreements they have with the telecom operator. More specifically the system holds information about the subscriptions associated to an agreement such as their physical and organizational belongings, the products on each subscription and their corresponding fixed costs etc. The system also provides functionality for ordering new subscriptions as well as administrating the data associated with the subscriptions and their products. Worth mentioning is that the system does not contain any information about the traffic — such as calls, text messages or similar — for a given subscription.

For some of the agreements in the system, invoice data is created periodically by a report server. Since only fixed costs are stored in the system, the variable traffic-related costs are retrieved from an external system owned by the telecom operator. These costs are then mapped to the subscriptions in the system and the CC:s can use the web interface in order to extract the invoice data as well as filtering it by date and organizational belonging.

Lately, Links has identified a need of a more sophisticated analysis of the invoice data created by the system. As for now, the data cannot be filtered on different types of costs, comparisons of invoices are not supported and the presentation of the data is done through Microsoft Excel-files. Aside from this the performance of the extraction is rather poor, for some agreements the data can take up to 30 minutes to generate.

During the fall of 2009 an idea about a new system, offering high-performing analysis possibilities and graphical presentation of the invoice data, started to evolve at Links. A basic requirement specification as well as some use cases were compiled and have provided the foundation for the work described in this report.

1.1.2 Problem description

Based on the initial documentation provided by Links, the task was to design and implement a first version of a web-based system that provides functionality for analyzing and comparing costs provided from the already existing system described above. The costs should be able to analyze with respect to time, organizational belonging and type of cost, such as fixed- or mobile traffic, fees etc. In order to allow for fast retrieval of data it was decided that the tools and features within Microsoft SQL Server Analysis Services — due to its support for aggregations and multidimensional data storage – should be considered when building the system. Some informal requirements were that the system should be generic, easily extendible and that it should be fast. Generic in such a way that the functionality fits a wide range of user needs; extendible in such a way that data from other telecom operators could be added without too much effort; fast in such a way that the output should be generated in measures of seconds, not minutes. The working title of the system is Brix.

1.2 Goals

In the documentation provided by Links, several different use cases were described and they covered anything from cost analysis to user administration. These use cases were prioritized and it was decided that the task should focus on the most important ones in first hand. The goal of the work carried out was therefore to implement the functionality described in the use cases dealing with cost analysis, which includes building an infrastructure that allows for future modifications of the system. More specifically, the goal was to build a data warehouse solution that is fed with data from STAX and to implement a web user interface that provides tools for analyzing this data according to the use cases. The company aimed to have a first version of this system ready for demonstration to customers before the end of May 2010.

When this first version of the system is complete, Links would then assign resources in order to implement the other use cases.

1.3 Limitations

As already mentioned in the previous section, no focus was put on the use cases that didn't correspond to cost analysis. Since the scope of the described system covers everything from building a web user interface to designing a data warehouse solution it was decided that an external interaction design company would be hired for dealing with the user-friendliness and graphical look of the system. Hence this work will not be considered nor described within this report. However, the actual implementation of the user interface was still to be made and will be taken into account in this report since it contains areas of importance for the performance of the system.

Due to the relatively short timeframe of the project it was also decided that a test engineer at Links would support the authors of this report in different activities related to testing and verification. These activities will therefore not be considered in this report.

1.4 The Company

Links Got AB (Links) is a fairly small company with 10 employees and has their office located in Gamlestaden, Göteborg. Links was founded in year 2000 and has since then been delivering IT services and management support mainly targeted towards companies within the telecom industry. Links is specialized on Managed Services (MAS) which briefly described is a practice where a company offers their customers functionality instead of a technical solution. For example, instead of selling telephone switches the company sells the function *telephony*.

Links is a certified Microsoft Gold Partner and develop their applications entirely upon products on the Microsoft stack, such as the Windows operating system, Microsoft.NET platform, SQL Server, BizTalk etc.

2 Theory

2.1 Business Intelligence

“Business Intelligence (BI) refers to computer-based techniques used in spotting, digging-out, and analyzing business data, such as sales revenue by products or departments or associated costs and incomes” [3]. The product developed discussed in this report is indeed a good example of a Business Intelligence application. As the definition implies BI tools are there for offering decision makers with the data needed for performing relevant analysis and giving solid support when making decisions. It is quite possible the same data exists elsewhere but BI systems strives for offering the functionality wanted and needed by the user. The functionality given should give a common ground for all the users or a “one truth” as well as good options for analysis. This together with increased performance and easiness of use should save time for the users as well as offering better information. BI tools often use data retrieved from a data warehouse but not all data warehouses are used for BI and not all BI systems relies on a data warehouse.

2.2 Data Warehouse

A data warehouse system is an organization’s repository for storing data about the organization but also the means to retrieve and analyze such data. Organizations typically have several to many more or less standalone systems for handling all their needs. An organization may have different systems for handling employees, sales data, customer relations, budget etc and as a whole these systems are generally not well coped at answerring simple questions of analytical nature. The essence here is that information needed for answerring interesting questions like "How much did we sell to customer C? Did he pay in time? Was C happy?" may not be easily gathered from the sytems.

One of the key problems thus with these different independent systems is that even though the data needed for user analysis certainly is available somewhere in the collection of systems there may be no easy way to actually retrieve it. Another issue is that while the underlying data source systems contains data and information that may be relevant to that specific system the data fulfills no purpose when looking at the systems as a whole.

A data warehouse can bridge these problems and make data appear consistent despite any differences in the underlying data sources. On top of this only the relevant information for analysis can be chosen to improve both the space needed as well as maintainability. The technique of extracting, transforming and loading (ETL) plays a vital part when planning and designing the usage and role of the data warehouse [20]. With a properly designed data warehouse users within the organization can then analyze relevant and consistent data creating a de facto standard for the organization as a whole. This is possible even though the actual sources may have no implemented connection between each other.

2.2.1 Storage Approaches

There are two major approaches for the storage of the data in the data warehouse: the normalized and the dimensional approach.

Normalized approach The normalized approach can be thought of as a data warehouse being made in a relational database. Tables are created and grouped by categories such as customers, products, finances and similar, meaning it is fairly easy and straightforward to add information into the database. However it might be difficult to join the data coming from different sources into meaningful information and also for a user to access this information without a very good understanding of the actual structure of the sources of data.

Dimensional approach The dimensional approach is fairly different, here numerical data such as sales or products ordered are partitioned into *facts* while the needed reference information for giving context to the facts are called *dimensions* [5]. As an example, for finding out how many products and for what amount a certain customer placed orders during a certain time interval one would use the *fact table* 'Sales' with the facts/*measures* 'number of products ordered' and 'sales amount' as well as dimensions for customer, product and date/time. Further the members in a dimension are usually aggregated on different levels for improving performance and usability. The dimension for stores with member attributes specifying the location of the stores would likely have aggregated data on suitable levels such as 'country' and 'state'. Advantages to this approach includes that a user lacking insight into the underlying structure finds such an implementation more intuitive to understand and use as well as the retrieval of the data tends to operate very quickly because of the pre-calculated results. However, managing the data from different data sources while maintaining integrity of the facts and dimensions may be complicated and heavy design and implementation work could be needed. Also updating and modifying the data warehouse structure to correspond to the changes in how the organization does its business involves further maintenance.

There are two different query languages used for extracting information depending if data warehouse is built in a normalized or dimensionalized way. SQL (Structured Query Language) is used for queries against the relational database and has been around since the early 1970s while MDX (MultiDimensional eXpressions) was first introduced in 1997 as part of a Microsoft specification for Online analytical processing (OLAP).

To further distinguish between a data warehouse and traditional operational systems the information from business transactions may often be stored into dozens or more tables in a relational database. This puts emphasize and focus on maintaining data integrity as well as fast insertions and updates which can be achieved because only a small portion of the data is changed during a transaction. Data warehouses are built for a completely different purpose, namely supporting analysis of the already existing information. Thus the main requirement apart from the data being correct is usually improved reading speed, actually changing the data would have to be made in the underlying data and then the state of the warehouse would have to be refreshed. The increased data retrieval performance is mainly accomplished through denormalization of the data into the dimensional model as well as storing the same data multiple times on different levels of granularity, called aggregations. To summarize, there are several conclusions regarding the benefits as well as disadvantages with using a data warehouse.

2.2.2 Benefits

- A data warehouse provides a common model of the data of interest for all analysts and decisions makers throughout the organization. This makes it easier to analyze and/or report data that may have its origin in multiple independent underlying systems.

- Existing inconsistencies are identified when designing the data warehouse and prior to the actual loading. This further supports a general model of the information within the organization.

- Even if the data in the underlying source is deleted or changed it is possible to keep the old information inside the data warehouse.

- Because the data is stored separately it can be retrieved without affecting and slowing down underlying systems.
- Since a data warehouse is simply a repository with data being read-only the design on both high and low level can be optimized for performance speed.

2.2.3 Disadvantages

- The data from the source system(s) needs to be carefully restructured and transformed to be properly divided into the design of the data warehouse.
- Due to data being first loaded and transformed from the underlying system(s) there is always some latency, the only guarantee is that the data is correct as of the latest time of update.
- Because a data warehouse only provides a new view of the already existing data it comes with an extra cost for possibly limited new functionality. When changes to the organization and/or the data are made the data warehouse may have to be updated in order to reflect the changes [2].
- Functionality can often be duplicated in the subsystems as well as in the data warehouse. Also functionality that may have been better off being implemented in data warehouse is created in the underlying systems and vice versa.

2.3 OLAP

Online analytical processing (OLAP) is a database approach to rapidly answer multidimensional analytical queries. The multidimensionality corresponds to the dimensional approach of a data warehouse and performs better when retrieving analytical information than the normal relational database mainly because of pre-calculated results [4]. The data inside an OLAP database are logically represented of one or more OLAP cubes which in turn are built up by dimensions and facts. As the name would imply a simple OLAP cube can be thought of as a normal three dimensional cube with three business related dimensions —i.e. product, customer, date— on the axes and each cell in the cube a business measure/fact, a numerical value that is specified by a coordinate along the three axes. The name cube implying three dimensions is however chosen for imaginative reasons, in reality the number of actual dimensions within a cube has nothing to do with the number three.

As mentioned earlier in 2.2.1 an important and useful notion is how a dimension can be aggregated on different levels. The dimension 'Time' that intuitively offers a context of when a business measure occurred would in most cases include a *hierarchy* consisting of the levels 'year', 'quarter', 'month' and 'date' and in that order. A company selling furniture consisting of a time dimension as well as a dimension Products; consisting of a hierarchy furniture group->furniture id and a measure Sales could then let a query *slice* —which can be thought of as applying a filter— down through the cube on the axes/dimensions. The OLAP database would then calculate the sales for the cell(s) that has been sliced down to.

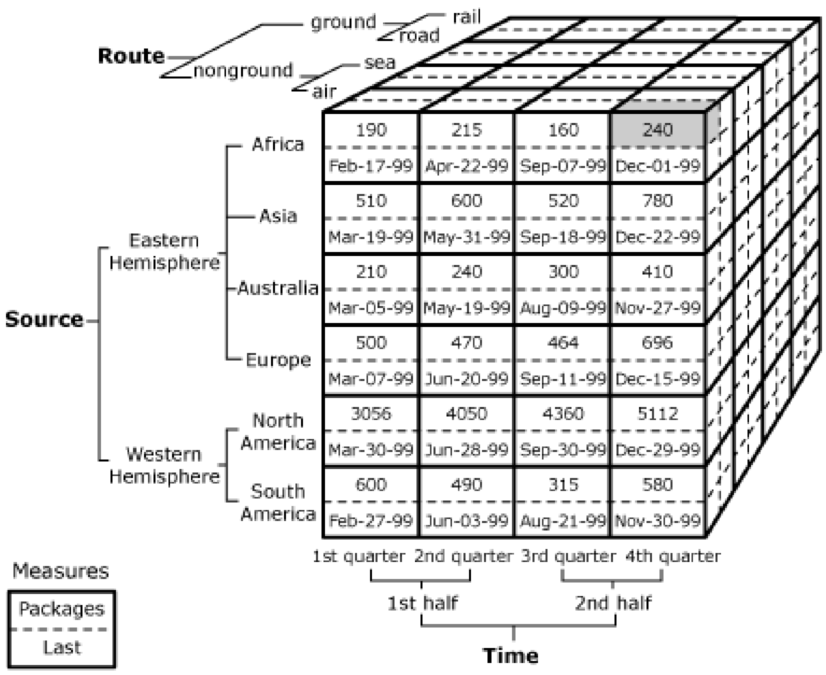


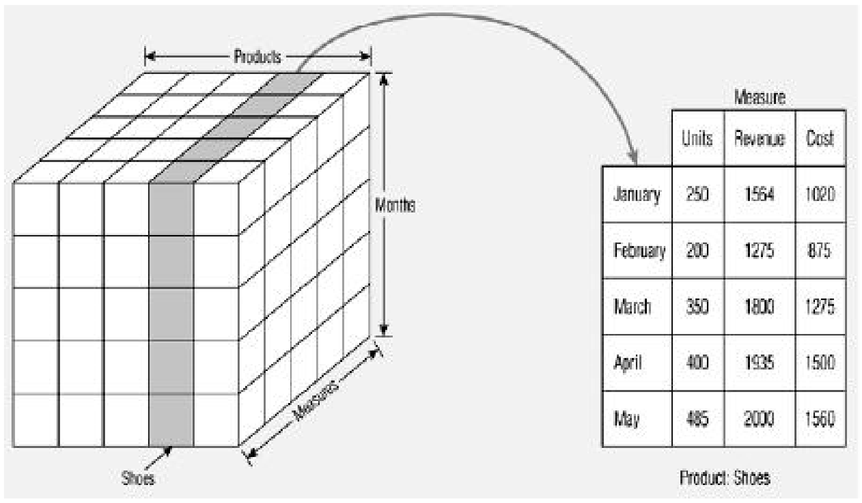
Figure 1: Cube showing possible levels of aggregations with measures inside cells. ©Microsoft.

The main benefactor for the improved reading performance of OLAP compared to a normal relational database is how results can be pre-calculated along the aggregated levels. A common analysis is the year-to-year change of sales for all the stores in a given country. In a normal relational database retrieving the results would demand server calculation of all the individual sales in all the stores at runtime and then compare the two years with each other. A typical OLAP database on the other hand pre-calculates results on different aggregated levels at loading time offering rapid responses to analytical queries even if the number of underlying fact rows may be huge.

The main query language used against the OLAP database is called MDX (MultiDimensional Expressions) and can be thought of as the equivalence of SQL with relational databases. The way MDX queries typically work can be thought of as firstly specifying the scope which is a subset of all the information within the cube that will act as a temporary subcube for the query. Then doing a *drill-down* on the dimensions within the cube the query retrieves the result for typically one or more measures as columns with a set of members of dimension as the rows. As a basic example it could be of interest to analyze how the amount of sales of a certain product in a specific store fluctuates during a 12-month period. A user would then chose to slice on a given store and product and set the resulting matrix to show the measure sales as the column together with the 12 months as rows. The 12x1 values of the cells in this resulting matrix would then correspond to the amount of sales for the specified product during each month. The query could then easily be modified with adding more measures such as net sales or number of products sold on the column axis as well as doing the calculations on a grouped collection of products rather than a single member.

2.3.1 OLAP Types

The data in an OLAP database can be stored in three fundamentally different ways and each of them deserves mentioning. MOLAP (Multidimensional OLAP) is the classic and most famous one and is usually referred to — including this report— as simply OLAP. As mentioned above it stores data in optimized multidimensional array storage instead of a relational database. This requires pre-computing and loading of data —processing — into the cube before it can be used by a user. ROLAP (Relational OLAP) works directly with storing the data in traditional relational databases. It uses both more relational-like tables containing all the base data as well as other tables



	Units	Revenue	Cost
January	250	1564	1020
February	200	1275	875
March	350	1800	1275
April	400	1935	1500
May	485	2000	1560

Figure 2: A two dimensional slice of a cube. ©Microsoft.

created for storing the aggregated information. HOLAP (Hybrid OLAP) is a mix of the two above which can be useful if there are too many members on the lowest levels of hierarchies which is better stored in a relational database while parts of more aggregately data can be stored as a MOLAP for better performance results.

Main advantages of MOLAP include increased performance related to optimized storage and indexing as well as needing less storage space due to compression techniques. The reason why it can perform so nicely is however also the biggest drawback and potential problem. Because results are pre-calculated at loading time there is the possible bottleneck of this simply taking too long to be feasible and smooth or in worst case slower than new data gets added. A general solution if/when this problem occurs is to choose not to pre-calculate everything but rather deciding which aggregated data that is more important and used more frequently. This subset of all potential aggregations would then be pre-calculated in a feasible amount of time leaving the rest to be calculated at run-time if queries request the information.

ROLAP can easier and possibly better scale up to a large number of dimensions with lots of members and a tremendous amount of fact rows [10]. However pre-calculation of results is more difficult to implement efficiently and often skipped leading to worse analysis performance than MOLAP. A

ROLAP solution is also more limited to the underlying database when it comes to offering the user with useful specialized business intelligence functions. HOLAP naturally tries to encompass the best of the mix using either MOLAP or ROLAP when it gives the best performance and scalability.

2.3.2 Aggregations

Aggregations and the corresponding pre-calculated data is one of the key performance boosters with OLAP due to the fact that to retrieve result from a high level the server does not have to do real-time calculations of every single member on the lowest level and then add these together but instead the result has already been calculated and stored when the data was processed. This basically means that the performance boost compared to non aggregated storage increases with dimensions consisting of multilevel hierarchies where each level consists of a limited number of members. Less members but more levels means a high number of aggregated results that is calculated fast and provide that fast cube performance [1]. On the other hand large number of members means pre-calculations may take too long to be feasible along all levels as well as the space needed to store the results may be too much. Without being able to pre-calculate the results the information wanted by user has to be resolved at run-time by retrieving information from the data-source which obviously goes slow and with worse performance than not using a MOLAP at all but instead ROLAP.

To further boost performance analysis services comes with a feature called Usage based aggregations [21]. This feature allows the system to log queries to the server and what parts of the cube that was used. This information both informs the administrator what data is most frequently used and at what level as well as providing support on what aggregations that would be most likely give the best effect to pre-calculate. Administrator can then set how much space that is allowed to be allocated or to how high percent of all queries that is wanted to be pre-calculated and system will offer automatic support of this. This feature is often important due to the fact that pre-calculations of every level may not be feasible and by taking advantage of this logged information of user behavior a suitable balance between performance and pre-calculations can be made.

2.3.3 Partitions

When designing the cube and preparing it for a relatively high future load of inserted data it may be crucial to group the information into different physical partitions. Well-designed partitions means information related to each other and often asked for in the same MDX query will be within the same partition. Example of such data would be all the sales for all stores within a specific country during a certain year. The effect of partitions are primary two: First of all it greatly decreases time of processing when only partitions effected of new data needs reprocessing. Secondly data related will be physically together meaning more efficient caching and increased responding time [15]. As well as this there are further more performance tweaks available to the developer including optimizations in the design as well as the MDX queries themselves [19].

2.3.4 Processing

Generally with OLAP there are not really any specific best practices and there are often many options for accomplishing the same task. As an example given earlier with a given set structure the same wanted output could be accomplished from more or less unlimited different MDX queries. There are several different high structural ways of selecting the subcube —slicing down in the dimensions to get a smaller/sub cube from the whole cube— in the query as well as dozens of different ways of specifying different member(s). One should obviously aim for performance but also strive for ease of understanding and consistency between the different queries.

Processing of the cube is the art of actually getting the information from the data source(s) into the analysis server [9]. Now while a relational database simply stores raw data and with the possibility to whenever wanted add new rows or change existing ones —as long as no constraints are broken — this is never so simple with a cube. In the simplest case the OLAP designer specifies a data source view mapping the relational tables and columns of interest into the fact tables and dimensions of the cube. To make the data ready for the user to view the administrator then processes the cube thus transforming the source data into pre-calculated results that may be stored multiple times on different aggregated levels.

While pre-calculated results certainly boosts reading performance it does have the drawback that the information the user sees is only accurate and valid as of when the latest processing was made. Any new added facts means new calculations have to be made along all the aggregations that will be affected. Such a processing is called incremental processing and involves firstly processing the new rows only and then merging together the old and new one while caching old cube for user to still be able to read. Changing existing rows though is another matter due to the fact the information has already been processed and the connection to the relational row is after that lost. This means to accurately show changed or deleted fact rows one would have to reprocess the entire cube —or at least the effected partitions— to reflect these changes.

It is possible to have a more or less real-time OLAP solution that with quite little latency reflects the current state of the underlying data source. While this is a preferred solution in situations with frequent changes to the data and the need to seem like a real-time system it comes with a price of automatic polling and less control as well as performance decrease while updating. For the current system the updates to the relational database occurs on an automatic known schedule and thus all that is needed is to start the incremental processing of the cube after the input to the data source has been completed by the Data Extractor, as described in next chapter.

2.4 MDX

2.4.1 MDX Introduction

The following will just briefly explain the core concepts of an MDX query as well as highlighting any main differences to SQL, for more detailed information there exists several interesting books as well as articles available online [16] [7] [11]. As a recap and summarization of information earlier given in this report MDX queries are run against an OLAP cube. A cube can be thought of consisting of two main concepts:

- Measures are the numerical values used in the analysis such as sales, number of sales or cost. They correspond to a given cell's value inside the cube.
- Dimensions are the categories or reference tables that the cube is built from. A dimension often consists of a hierarchy with different levels for the user to specify the granularity of the request. The levels in turn consist of a number of *members*. A dimension Time would typically

be broken down to a hierarchy with levels Year, Month and the finer grained Date with members for the whole years on the year level and Date level having every day of these years as members.

A very basic MDX query typically will be following a code skeleton such as:

```
SELECT axis specification ON COLUMNS,
axis specification ON ROWS
FROM cube name
WHERE slicer specification
```

With a cube consisting of —amongst others— a Date and Store dimension as well as a measure for sales the user could be interested in the following basic analysis:

```
SELECT
   { [Measures].[Store Sales] } ON COLUMNS,
   { [Date].[2002], [Date].[2003] } ON ROWS
FROM Cube_Sales
WHERE ( [Store].[USA].[CA] )
```

The above is a basic example of an MDX query, the actual language in this form is actually somewhat intuitively understandable. However what it actually outputs as well as why and how this is done may be trickier to grasp. First the above query specifies the measures and dimensional members to output on the axes— typically measures are specified on the columns with the wanted dimensional members as rows. In this example the matrix will consist of the members 2002 and 2003 on the x-axis with the measure sales of the stores as the single member on the y-axis. Cube-Sales is the name of the cube built up from these dimensions together with the numerical values given from the fact rows in the data source. The WHERE clause above states that only the stores of California —a child of the member [Store].[USA]— is of interest and only the sales from these stores will be calculated. Although this two dimensional output matrix is a typical example it is quite possible to retrieve results on more than two axes as well as specifying much more complex statements.

2.4.2 SQL simple comparison

A possible mistake for a developer comfortable with SQL would be to think of MDX as simply another similar query language that works in roughly the same way. However, while it looks quite similar to the same to specify the following simple query it really is quite fundamentally different on a lower level.

```
SELECT [a measure] on COLUMNS
FROM [a cube] WHERE [some kind of condition]
```

The SQL statement such as

```
SELECT [Id] FROM [TableSales] WHERE [Sales]>5
```

specifies exactly what information to retrieve and then from which table and what conditions that has to be fulfilled for row to be included. An OLAP cube on the other hand offers a multidimensional context built up from the information given in several/many relevant relational tables. The wanted output is the calculated measures/facts from the cube and thus the simple

```
SELECT [Measures].[Sales] on COLUMNS from [Cube Sales]
```

outputs a single numerical field corresponding to the total sum of the sales of the entire collection of data residing within the cube. WHERE and similar clauses is used as a way to define the scope, namely defining what smaller part of the entire cube that is to be used in the rest of the query. If the member(s) of a certain dimension are not specifically defined then the entire collection of members is part of the scope. Such as the example above where every member in every dimension is part of the scope.

As such, the default view of a cube that can be queried can somewhat be thought of as an SQL query that joins every related table together to the one containing the measures of interest. Specifying the exact columns in SQL together with the function sum of the wanted measure and some grouping would give a similar output as a simple MDX query with two axes.

2.4.3 Main differences compared to SQL

While the two scripting languages may seem somewhat the same at a first glance there are some very important but yet simple characteristics of MDX that cannot easily be translated to SQL. As mentioned earlier the cube may aggregate results multiple times on different levels on a dimension's hierarchy. The dimension Store could for example be built with the levels Country, State, City and Store from a table with the columns CountryName, StateId, CityName and StoreId. A common dimension like Time could similarly include a hierarchy such as Year-> Quarter-> Month of Quarter-> Date. Recollecting the very first query from above outputting the sales for 2002 and 2003 from all stores in California it is simple enough to write with MDX but would require a construction with higher complexity in SQL. Firstly one would have to join together the three tables with the conditions of retrieving rows only for the two specified years and the state and then group over the years together with a sum of the sales.

Albeit the SQL query discussed above would be quite a bit longer involving different sub-queries it would still be rather doable to construct in SQL and fairly straight forward. However, the above uses a normalized hierarchy which works very well to work with and query in a relational database. Sometimes a parent-child hierarchy is preferred —when number of levels may be different and/or be subject to change at any time— and this does not work nearly as well in a relational database. A cube on the other hand automatically creates levels corresponding to the parent-child hierarchy and it can then be used just like a hierarchy coming from a naturalized relational table. In fact it is even easier start using due to the fact the cube automatically recognizes the relations while in a normalized hierarchy they would have to be somewhat manually specified.

As can be seen in addressing parent-child hierarchies with SQL and doing calculations is not really an easy task and even if divided into multiple queries it still is not intuitive and straight forward. As a simple example can be used the dimension Post Type which contains a parent child hierarchy of several levels of granularity. A root node named 'Traffic Costs' contains amongst several others the children 'Data Traffic' and 'Abroad Calls' which in turn may or may not have more children. The rows in the fact table Posts all include a reference to typically a leaf node at the lowest level of the Post Type hierarchy. A user may want to slice/filter and include the data that belongs to Traffic Costs or any of the leaves below it and then group the result on the leaves one level below Traffic Costs. However, this means first joining down recursively through the hierarchy—which is no easy task— and then group the result on a level lower than the first one specified. Finding an intuitively understandable and easy solution would be tricky and most likely involving multiple queries. This compared to an MDX query to the cube which will do all that by default by first specifying the member Traffic Costs as a member of the scope and then using an existing function for specifying its children as one axis of output.

As mentioned earlier there are also several other important differences in the way the two languages behave on a logically lower level. SQL can only specify output on a two-dimensional axis (columns and rows) while MDX can choose output on many more dimensions—64 is the maximum definable in MDX. Conceptually SQL uses the where clause to filter the data while MDX provides a slice of this data and while this may seem similar they are not equal in concept.

2.4.4 Equal result-Different statements

Member Definitions and Member functions There are usually several different ways of addressing a specific member as well as multiple possibilities to define a set including many members. On top of this one can choose to slice out the data that is to be part of the query scope in several ways. The dimension Date that includes the two hierarchies Year, Quarter, Month of Quarter, Date and Year, Month, Date can relate to the same member with these four examples:

- [Date].[Month 3 of Quarter 2 2009]
- [Date].[June 2009]
- [Date].[June 23 2009].Parent
- [Date].[Year-Quarter-Month-Date Hierarchy].[Month of Quarter].[Month 3 of Quarter 2 2009]

Of the above only the last example specifies a hierarchy which shows that MDX automatically finds members even without explicit definition. The concept here is how all of the above actually relates to the same member which got one specific Id that could also have been used if it had been known, like [Date].[1234]. [Date].[June 23 2009].Parent is an example of a member function and although this particular specifies the parent of a member many others exists.

Another example of a member function would be to rewrite the top example to the following:

```
SELECT
   { [Measures].[Store Sales] } ON COLUMNS,
   { [Date].[Year].Members } ON ROWS
FROM Sales
WHERE ( [Store].[USA].[CA] )
```

The above retrieves the sales in California for every year available in the Date dimension. Yet another example is the Set function Descendants which can be used like: Descendants([Date].[Year-Quarter-Month-Date Hierarchy].[2005]) which returns a set containing every member drilled down to the lowest level namely all quarters, months and days of 2005.

Defining the query scope While one could define the members to slice on for defining the scope in the WHERE part this is mostly not advisable for performance reasons. Although the result may be cached the database would in the top example still slice both 2002 and 2003 individually with the stores in California before calculating the result. A more efficient statement performance wise is to move the actual slicing so that it is made before any calculations and look ups is to be made.

```
SELECT
   { [Measures].[Store Sales] } ON COLUMNS,
   { [Date].[2002], [Date].[2003] } ON ROWS
FROM (SELECT [Store].[USA].[CA] ON COLUMNS FROM Sales)
```

This outputs the same result as the earliest query but the way it works is slightly different. The above creates a subcube from the whole cube which will be the scope in which everything else works. The subcube can of course be limited by many more dimensions than just stores and it improves performance over slicing every member every time with the WHERE clause. Apart from this there exists more ways for defining the scope in which a statement exists although the above are the fundamentally simplest to use.

3 Technology

Throughout the work several different tools, components and frameworks where utilized and this chapter gives a brief background and a description of them.

3.1 Application tier tools and components

3.1.1 Microsoft .NET Framework

Microsoft .NET Framework is a platform upon which developers can build software applications that executes on computers running the Windows operating system. Except the fact that Java is platform-independent, the .NET Framework has many similarities to the Java platform. They both contain two major components; one function library containg classes which developers can utilize in their applications and a virtual machine where the applications written for the platform are executed. In .NET Framework, code is compiled into an intermediate language called CIL (formerly MSIL) and resides in assemblies which could be either libraries (DLL) or processes (EXE). The assemblies are loaded and executed within the Common Language Runtime Virtual Machine and referred to as *managed code*.

The .NET Framework is language independent, i.e. applications written for .NET Framework can be written in any language that conforms to a specification called the Common Language Infrastructure (CLI). Thus there exists several languages using different paradigms that could be used when programming applications for the .NET Framework, among them are C, VB.NET, F etc.

At the moment, version 4.0 is the latest stable release of the .NET Framework and was released in April 2010.

3.1.2 Visual Studio 2010

Visual Studio is an Integrated Development Environment (IDE) created by Microsoft and mainly targeted to developers building applications upon the .NET Framework.

3.1.3 ASP.NET

Within the .NET Framework, two different frameworks might be used when developing web applications; ASP.NET WebForms and ASP.NET MVC. The former was shipped with the very first release of .NET Framework and was targeted to developers from the desktop-side. Windows Forms, which is the corresponding framework for developing desktop applications, share many concepts with ASP.NET WebForms.

With ASP.NET WebForms, Microsoft introduced a number of mechanisms that aimed to turn web pages into stateful components. Among these mechanisms was *server controls*, that abstracts the rendering of the HTML for the developer and provides an event-driven approach in order to respond to user interaction, as well as helpful functions for reading and updating the state of a web page.

Criticism was aimed towards these mechanisms due to the overhead involved when maintaining a state over a stateless media, namely the HTTP protocol. This criticism gained in strength as light-weight frameworks like Ruby-on-rails became more popular and in 2008 Microsoft released a new, alternative, framework called ASP.NET MVC.

In ASP.NET MVC aims to give the developer more control over the rendered html and the state and is to be seen as a complementing, not competing, framework.

When selecting between those two frameworks a number of factors were considered. From a performance point of view, web pages developed using ASP.NET MVC gives a more fine-grained controls of the rendered and also saves the server from maintaining a state by default [18]. On the other hand, ASP.NET WebForms can be configured so that state management is disabled. Eventually, ASP.NET WebForms was selected due to mainly two reasons:

- The developers at Links have more experience within ASP.NET WebForms.
- The components within Telerik RadControls for ASP.NET, among them server controls for creating charts, are supported.

3.1.4 ASP.NET WebForms

Web pages developed in ASP.NET WebForms are called web forms and generally contains of two different parts, the code-behind file and the markup-file.

Most often these are located in different files. The markup-file contains the HTML as well as the declarations of *server controls*, which could be for example graphical components such as a button or a hyperlink but also more complex controls exists. The code-behind file contains the server-side logic of the web page.

Similar to the Java swing library, web forms are event-based. Each server control exposes a number of events that could be handled from the server-side logic, for example a click on a button could be handled in order to perform a specific operation.

Web forms and server controls are stateful, which means that ASP.NET maintains a state at the client that is persisted between different loads, or *postbacks*, of the page. This feature is realized by serializing the state of each server control on the page and adding the resulting string, called *viewstate*, to a hidden form field that is embedded to the response sent to the client. On a postback this string is sent back to the server and by deserializing it ASP.NET can recreate the state of the user interface. This mechanism comes in handy when for example a dropdown list is to be filled from a database. The database only needs to be queried on the first load of the page while subsequent postbacks can recreate the list by using the viewstate. However, this obviously also increases the data amount sent between the client and server.

The entire web form in itself is a server control as well and exposes a number of events that relates to something called the *page lifecycle*. This is initiated when a HTTP request is made to the web page and contains a number of stages which are briefly described here.

- First, in the request stage, ASP.NET determines whether the requested page needs to be parsed and compiled or if a cached version of the page can be delivered to the client.
- In the initialization and load stages, the page is parsed and the server controls are created and, if a postback is made, their states are recreated from the viewstate.
- Finally, in the render stage, each server control including the web form itself is rendered into HTML and appended to the response sent to the client.

3.1.5 Asynchronous javascript and XML - AJAX

AJAX is the name of a collection of technologies for making asynchronous requests using client-side scripting. By using AJAX developers can build dynamic interfaces that loads data from the server as it is needed and updates only the part of the display that needs to be updated, thus optimizing the amount of data loaded. Also, slow-running tasks can be performed asynchronously without affecting the UI responsiveness. Some examples of well-known interface components that often utilizes AJAX technology are progress bars, autocomplete text boxes and interactive maps.

Even though AJAX is a fairly new term, the technologies behind it have been around since the early days of web development. A central component of AJAX that makes the asynchronous server communication possible is the XmlHttpRequest-object, which must be supported by the client's browser in order for AJAX to work. Currently all modern browser supports it.

Since different web browsers has slight differences on how to use the XmlHttpRequest-object, implementing an AJAX engine can be tedious and needs to keep up the pace as new versions of web browsers are released. Luckily there exist a number of javascript frameworks that provides a stable and well-tested AJAX engine as well as utility functions for using it.

3.1.6 Telerik RadControls For ASP.NET AJAX

Since the data were required to be displayed in different types of charts, a third-party product suite called Telerik RadControls was utilized in the project. This product suite contains a wide range of server controls that could be used in ASP.NET solutions and among them are a set of controls rendering charts of different types as well as a framwork supporting the implementation of AJAX functionality.

Two controls were used more frequently than others in the project, the RadChart control and the RadXMLHttpPanel.

RadChart The RadChart component is used in order to visualize data in a chart. Several different chart types are supported — among them pie charts, bar- and line diagram — and each chart provides a excessive API in order to customize the look, such as dimensions and colors, of the chart.

The charts are generally loaded with data by creating one or more chart series, where each chart series item can contain one or more data points and are rendered as images before sent to the client.

RadXmlHttpPanel The RadXMLHttpPanel component is used in order to utilizing the ASP.NET callback mechanism, which as opposed to postbacks doesn't cause the web form to execute its entire lifecycle. The server-side performance is thus increased while on the other hand the client state must be managed explicitly.

3.1.7 LINQ

LINQ (Language integrated query) is a query language integrated into the .NET Framework that can be used to query data in any collection implementing the LINQ interface. For example, the same LINQ query can be used for an in-memory array of objects as well as for an SQL Server database.

Wrappers for many other query languages has lately been released for LINQ, so called LINQ providers, among them LINQ for XML, LINQ for objects, LINQ for SQL.

3.1.8 Telerik OpenAccess

Telerik OpenAccess is an object/relational mapper (O/RM) that contains functionality for retrieving and writing data into persistent state. The OR/M is used in order to generate a set of classes, or entities, representing the data in the database. Typically one entity per database table is created, while any foreign keys in the tables are reflected by relationships among the entities. Though, OpenAccess can also identify more complex associations such as inheritance and many-to-many relationships. OpenAccess fully supports LINQ and is controlled through an interface residing within Visual Studio.

3.2 Data tier tools and components

3.2.1 Microsoft SQL Server

Microsoft SQL Server at its core is a relational database server but it also comes bundled with many different tools and services including analysis services offering an OLAP solution. SQL Server uses several techniques for handling issues such as concurrency, transactions and buffering for providing a reliable database with good performance. In this project extensive work has been carried out relying on different parts of the SQL Server 2008 R1.

3.2.2 SQL Management Studio

SQL Management Studio includes both graphical and scripting tools for the managing of all the components within the Microsoft SQL Server. The tool was used throughout the entire lifetime of the work and extensively used for both exploring of content as well as configuring and managing of both the relational database as well as the analysis services.

3.2.3 SQL Server Profiler

The profiler can trace and monitor all communication made to the server showing how it actually resolves the queries internally. This is especially useful for performance monitoring of the cube. With the profiler it is amongst other things possible to see the duration of every query executed against the cube as well as any internal usage of —which is preferable from a performance point of view— cached data.

The profiler can trace and monitor all communication made to the database showing how the server resolves the queries internally. This is especially useful for performance monitoring of the cube. With the profiler it is amongst other things possible to see the duration of every query as well as any internal usage of —which is preferable from a performance point of view— cached data.

3.2.4 Microsoft Analysis Services

SQL Analysis Services (SSAS) includes a group of capabilities for OLAP as well as Data Mining; the method of extracting and finding patterns from available data. It supports all three different OLAP storage modes — MOLAP, ROLAP and OLAP — both on a dimensional level as well as on a lower partition level with some consistency limitations. SSAS supports several different APIs for communication depending on the level of operation as well as the programming environment. Below follows a list of a few such noticeably frameworks:

- XML for Analysis (XMLA), low level API based on XML. Is the industry standard for interaction with OLAP. XMLA can be used for both administrative tasks —such as managing structure of cube and processing of data— as well as accessing existing data for analysis. Not used extensively in this project as such, rather administrative tasks are performed with tools and frameworks which in turn rely on XMLA. For analysis purposes MDX scripts are sent alone —for simplicity and ease of use— to the server rather than embedding them within an XMLA statement.

- ADOMD.NET is a .NET based API for running queries against the OLAP. On a lower level it uses XMLA for interaction with analysis services.
- AMO is the administrative correlation of ADOMD.Net for managing the OLAP database and works in a similar fashion within the .NET framework.
- DDL (Data Definition Language) is an XML based language with commands for defining the OLAP objects and used for data mining.
- MDX is the OLAP query language comparable with SQL for relational databases. More detailed information about MDX can be found later in the report.

3.2.5 Business Intelligence Developement Studio

Business Intelligence Developement Studio (BIDS) customizes Visual Studio —2008 as latest supported version— providing an IDE for developing OLAP solutions together with SSAS. BIDS offers tools for both the managing and exploring of the database as well as the reporting services in the SQL Server. For the administrative tasks the user may opt to work both purely with the graphical tools available as well as modifying the underlying XMLA scripts. With the environment users can create a fully working OLAP database from a data source as well as optimizing performance with specifying aggregations and partitions. Any changes made by developer can then be chosen to be deployed to the server when ready for use. BIDS was used to build the OLAP solution from scratch including defining all the dimensions as well as which aggregations to pre-calculate and the interface chosen to be exposed for outside queries.

4 Method

In large, the work went through three major stages. One initial stage, consisting of theoretical studies and a first analysis of the problem, one stage where the actual planning and execution of the project were performed and one final stage where the project was handed over to Links development team for further implementation.

4.1 Initial stage

During the first two weeks of the work an initial phase took place where a theoretical understanding of the problem was gained. This was achieved by literature studies of the theory behind Data Warehouse solutions and OLAP cubes and by taking an online video-based course describing the tools within Microsoft SQL Server Analysis Services. Furthermore, meetings and interviews with people from Links were held in order to gain insight to the purpose and desired functionality of the desired system.

In order to allow for a direct and efficient communication with the employees at Links, two work stations were installed at their office where the work was carried out.

4.2 Project planning and execution stage

Within this stage - that took place in weeks 3-18 - the actual work was performed. For most software projects, regardless of which methodology that is applied, this includes the the phases planning, design, implementation, testing and deployment.

Initially, when the scope and limitations of the project was defined together with Links, it was decided that external resources were needed in order to fulfill the company's goals within the desired timeframe. Therefore, test-related activities were decided to be delegated to the company's test engineer and were scheduled later on in the project. Also, the creation of the system's Look-and-Feel was assigned to an external company.

Due to the circumstances described above — as well as the fact that some planning activities, such as elicitation of requirements, had already been performed — the emphasis during this stage was put on the design and implementation activities.

4.2.1 Project methodology

Initially there were many uncertainties in the project, the initial documentation only briefly described the system and there were many things in it open for interpretation. Aside from this a number of pilot Customers were supposed to be involved in the process later on, so it was estimated that requirements and directives were likely to change. It was therefore decided that an agile methodology should be applied. For a couple of years, Links has been applying Scrum in their projects and this methodology was therefore the natural choice for this project.

Scrum Central roles in Scrum are the Product Owner and the Scrum Team. For this project, the authors of this report formed the Scrum Team while the Product Owner was represented by an employee at Links. A Scrum project is performed in a number of iterations, called sprints, with a recommended length of approximately 1-4 weeks. Each sprint contains the phases *Pregame*, that includes planning and high-level architecture, *Game*, where development-related tasks such as design, implementation, unit-testing and documentation are performed, and *Postgame* where system testing, integration and user documentation is considered.

Requirements are stored in a Product Backlog, prioritized by the Product Owner. When planning an iteration in the Pregame phase the Product Backlog Items that should be included in that Sprint is selected during a Sprint Planning Meeting. The Scrum Team then disassembles these items into one or more Sprint Backlog Items. The Sprint Backlog contains more technical descriptions and should describe a task that takes around 4-16 hours. If a tasks is estimated to take longer time to complete it should be disassembled further. Each day, the Scrum Team has a short daily meeting, where any problems and uncertainties are discussed and hopefully resolved.

4.2.2 Planning the iterations

The uncertainties of the project made planning for longer periods difficult, so it was decided to use a rather short iteration length in the beginning of the project. Another reason for using a short iteration length was to allow for rapid changes and to minimize work done in case of misunderstandings. A length of two weeks was agreed upon to start with.

Below is a short summary of the performed iterations and their corresponding goals.

Iteration 1 In this iteration the goal was to fill the system with the data needed for implementing the analysis use cases.

In the Pregame phase, an understanding of the source system and its relational data model were gained and a list specifying the entities and attributes needed from the source system were created. A relational database, similar to the source system's database, was also designed in order to be able to store the data.

Since the system in its deployed state doesn't have any direct access to the source systems, the Game phase focused on reading the specified data over the network. The initial source system already exposed a web service from where XML-based reports could be retrieved over the HTTP-protocol. Thus, a report containing the data specified in the Pregame phase was created by a developer at Links and a console application responsible for polling the web service and reading the received report into the database was developed.

Iteration 2 The second iteration focused on two things. Firstly, a first version of the OLAP cube was modeled upon the relational database in order to get a more practical knowledge of the OLAP-specific tools and query language, MDX. Secondly, recalling the requirement that the system should support several data providers, the relational model was extended and generalized in order to allow data from other source systems to be used in the system.

The output from the Pregame phase was thus a revised domain model, where some concepts very specific to the initial source system were generalized. The initial OLAP cube as well as the console application developed in the first iteration was also modified according to the changes in the domain model.

Iteration 3 The third iteration targeted the overall design of the system. The different components and their correlations for the full system were identified and modeled. In the end of this iteration, the final delivery of the graphical sketches were done and different third-party components and frameworks were evaluated — among them Telerik RadControls and asynchronous requests — in order to verify that the sketches could be realized.

The third iteration targeted the overall design of the system. The different components and their correlations for the full system were identified and modeled. In the end of this iteration, the final delivery of the graphical sketches were done and different third-party components and frameworks were evaluated — among them Telerik RadControls and asynchronous requests — in order to verify that the sketches could be realized.

Iteration 4-8 The forthcoming iterations mostly concerned the design, testing and implementation of the components of the system, such as the interfaces to the relational database, the OLAP cube etc. The user interface and a business logic layer, abstracting the business functionality from the presentation layer, were also developed.

Thoroughgoing tasks Through the early iterations a number of meetings, or rather brainstorm sessions, were held together with the external company that delivered the graphical sketches and the flow of user interactions. During these the original use cases regarding cost analysis were discussed and many similarities were found among them. Instead of separating the use cases dealing with drill-downs and comparison, common functionality in them were identified and eventually the use cases were merged together into one single use case.

4.3 Final stage

During the last two weeks the project was handed over to Links in order for their development team to continue building on the application. Necessary documentation was provided and several meetings where held where the concepts and architectures used in the system were described.

5 Analysis

The initial documentation provided by Links contained a number of use cases and a brief requirements specification. Although this documentation gave the reader an overall picture of the described system many things in it were left unmentioned and based on implicit assumptions. An analysis of the documentation and the existing source system, STAX, was therefore performed during the work in order to clarify the parts that were subject to implementation. Eventually this analysis resulted in a domain model of the system.

5.1 Use cases

The use cases are described in detail in Appendix A.

5.1.1 Login use case

The login use case is rather straightforward and was selected for implementation since the cost analysis requires a logged in user.

5.1.2 Cost analysis use case

Many similarities were found in the use cases dealing with cost analysis and together with the staff at Links, as well as the employees at the external interaction company, those use cases were replaced by one single use case, referred to as the cost analysis use case. Despite the name of the use case not only costs are concerned in the analysis but also other invoice data such as number of calls etc.

The full cost analysis use case contains several steps and alternative paths and is attached in Appendix X. However, for the reader's convenience a brief description of the main path is hereby given natural language.

The use case requires an authenticated user that is authorized to the cost analysis module and is initiated when the user selects the cost analysis option in the application menu. Roughly speaking, the use case contains three major parts; (1) definition of filters, (2) a drill-down through the type hierarchy and eventually (3) a listing of the subscriptions and organizations associated to the filtered invoice data.

In the first step, the system loads a filtering section where the organizations and invoice periods available to the user are displayed in a hierarchy.

The user selects a desirable organization and a period for analysis. If appropriate the user also selects an earlier period for comparison.

Secondly, the system displays aggregations for amount, number of calls and call length for the given organization and period, grouped by cost category (fixed traffic, mobile traffic or fees). The total number of subscriptions and — if a comparative period was selected — a comparison between the aggregated amounts for the periods is also displayed for each category.

The user then selects a category of interest and the system displays a similar view as before, but with the costs grouped by the sub-category, or call groups, of the selected category. Similarly, these costs could be further grouped by their call type in subsequent steps.

In the last step, when the costs could not be grouped by type any further, the subscriptions for which matching costs are found are displayed to the user. These subscriptions can be grouped by organizational level such as their cost locations or company in order to get a picture of the organizational distribution of the filtered costs. By clicking on an item in the list more information about the subscription or organization is displayed.

5.2 Identifying concepts and their relationship

Some concepts and attributes was directly extracted from the use case, among them user, module, subscription, organization etc. However it is not clear how the invoice data is represented and how the different measures relate to each other. An analysis of the source system's data model was therefore performed.

5.2.1 Analysis of STAX data model

In STAX the invoice data are calculated periodically by a report server and stored in a SQL server database. The lowest level of invoice data is the invoice row which is related to a subscription in the system, as shown in the figure below.

Figure 3: Model of invoice-related concepts in STAX

When comparing the cost analysis use cases to the data model of STAX is it seen that the measures needed are all available in the invoice rows, which thus forms one of the main concepts of the analysis domain model. The number of subscriptions for a certain set of invoice rows is also available by counting the distinct subscriptions to which the rows are associated. Organizational belonging of an invoice row is given by its subscription and is divided into a hierarchy consisting of the levels company, customer and cost Location. However, for the type and period dimensions that is used in the analysis there are no corresponding hierarchies defined in STAX data model.

5.2.2 Modeling the period dimension

The smallest period length used in STAX is one month and a user is likely to want to group these months by quarter or year. At first glance it seems like a trivial task to divide these into a hierarchy since a month is always fully contained within a specific quarter and, similarly, a quarter is fully contained within a year. One problem is that organizations might choose to start their financial, or fiscal, year at different times and thus their "quarter one" might start in March instead of January. This makes grouping more complex since a period might require more than one parent period and the analysis must consider the different organizations fiscal years in order to aggregate the data.

The strategy applied here was to not organize the periods into a hierarchy but instead realize the grouping of periods by using the start and stop dates that are associated with each invoice. Also, instead of aggregating the measurements by period they are aggregated by their invoice. The aggregated value for a period could then be calculated by finding all invoices whose period is within two dates and summing their contributions. The advantage of this approach is that periods of any lengths could be aggregated and analyzed, not only months, quarters and years.

5.2.3 Modeling the type dimension

In the input data, the type of an invoice row is entirely identified by its textual description. These descriptions have no relation to each other and could either come from STAX itself or from other systems owned by the telecom operator. The number of different descriptions in the source system is in the magnitude of ten thousand and some examples of them are displayed below.

- Mobile Sub (New subscription)

- Online Turbo 3G
- To 077 number

The use case specifies how the user drills down through a three-level hierarchy in order to filter the data by their type. The following levels and types of costs are mentioned:

- Categories level consisting of the three types fees, fixed traffic, mobile traffic
- Call group level, consisting of types such as onetime fees, international calls, datatraffic, messaging services etc.
- Call type level, consisting of types such as SMS, MMS, Received abroad calls, abroad calls etc.

To complicate things further some of the types at lower levels were applicable for more than one category, international calls can for example be a sub type of both fixed and mobile traffic. Also, not all types are divisible into three levels, some might contain more and some might contain fewer levels. As an example, the cost type "Onetime fees", which is a child of the category "Fees" does not have any child types and thus contains only two levels.

With this knowledge taken into account it was decided to split the type dimension into two dimensions, one for the top-level category of the costs and one for the underlying type of the costs. In other words the invoice rows are associated to both a type as well as one of the three categories. By doing so the measures can, for example, be grouped both by international calls in general but also further drilled down by their category, fixed or mobile traffic. To allow the sub type dimension having a dynamic number of levels it was decided to model it as a parent-child hierarchy.

By using a set of mapping rules – maintained by a system administrator where each known textual description is associated with a category and a type — the invoice rows from the input data can be mapped to a member of each dimension.

5.3 Generalizing the concepts

From a business perspective one of the aims for the system was to support the analysis of telecom-related invoice data from other source systems — or data providers — and not only STAX. In other words, the system should be modeled as generic as possible within reasonable limits in order to minimize the development needed when new data providers are connected.

In the context of analyzing telecom-related invoice data — which is the main purpose of the system — the measures costs, call length and number of calls are all applicable for a generic case. If needed, additional measures could pretty easily be added by adding new attributes to the invoice row concept. Also, it was decided not to touch the modeling of periods, categories and types since these were considered generic enough; a period is a globally well-defined concept and the categories and types are defined in the system itself and will be used for any provider.

However, two major problems were identified from a generic point of view; the modeling of the organizational hierarchy as well as the mapping rules applied when assigning a category and a type to the invoice rows. The organizational hierarchy in STAX is not very generic and other providers might use different hierarchies in order to define organizational belonging.

The organizational hierarchy containing the levels company, customer and cost location was therefore replaced by the two concepts organizational level and organizational node in order to allow different providers specifying their own number, and labels, of levels.

Not only the hierarchical levels of the organizations are likely to differ between providers, but also the attributes of the organizations and the subscriptions. During the drill-down described in the use case only the hierarchy and the names of the organizations are used, but a user is also supposed to be able to view information about a specific subscription or an organization — such as owner, address or similar — later on in the use case. However, the only attribute that can be considered valid for any provider is name which is why any other attributes are left unmentioned in the domain model. In order to allow for storage of other attributes a provider-specific implementation needs to be done, as described in the Implementation chapter.

In order to generalize the mapping rules described in previous section this concept was associated to the provider concept, thus allowing each provider to define their own mapping rules.

5.4 Strategies for slowly changing dimensions

Periodically, the system is fed with invoice data containing all information needed by the system, such as the invoice rows and any related dimensional data such as subscription and organizational belonging. Even though the invoice rows represent new data to the system the related data, i.e. the dimension members, might already reside in the system. This data might change over time — referred to as *slowly changing dimensions* — and a strategy for how to handle these versions of dimension members is needed.

One approach is to treat each unique version as a unique member in the dimension. The problem with this approach is that the connection between the versions is lost, thus no comparison between the versions can be done, which is required in the use case. Some kind of custom mechanism needs to be implemented at a higher level in order to group different versions.

Another approach is to keep the latest version as a member, i.e. when a new version appears the old is written over in the dimension. However, this will destroy the historical data which might be unwanted when analyzing older periods of time.

The dimensions subject to change are the type and organizational belonging. The periods are immutable in that sense that a member is defined by its own value. For example, "March 2010" cannot change to "February 2010" or similar. An analysis of the cost analysis use case was performed in order to identify the cases when changes within these dimensions could cause a problem. Each case was discussed with employees at Links in order to develop a strategy for how to handle them.

1. Organizational belonging of costs Each post in the system is associated to a subscription which in turn are associated to an organizational hierarchy. The attributes of the posts could be aggregated on different levels within the organization. When an organization moves within the hierarchy, it was important to ensure that any previous posts associated to the organization are not reflected in the new hierarchy. For example, if cost location moves from company A to company B in January 2010 this means that posts belonging to the cost location only should be included in the aggregations of company B if they regards a period after January 2010. Any previous costs should still belong to company A.

The solution to solve this was to create a dimensional member if and only if the organization hierarchy is changed. Other changes such as name or other information are updated to reflect the last known version of the members. In order to maintain a connection between two members that has been assigned to a new hierarchy the concept organizational node and subscription were extended with a relation, predecessor. An old version of the member is this associated to the new version of the member.

2. Organization filtering The organizational filter displays the names and the organizational belonging of the organizations authorized to the user and is loaded in the first step of the use case. The problem here is how to handle the display of organizations that change name or that has been moved within the hierarchy. Since the filter is loaded very first in the use case and does not depend on the periods selected it was decided to display only the current state of the moved organization, meaning that previous states of the organizations are hidden.

Due to the solution in the previous case the organizations might exists in different versions. This is solved by only displaying members for which no predecessor exists and thus only the current state of the hierarchy are displayed.

3. Type of costs Similar to the previous case, the types of costs are also organized into a hierarchy which might be reorganized over time. A major difference compared to the organizational dimension is that the type hierarchy is defined in the system itself and not by the providers. It was therefore decided that a change in the type hierarchy would be reflected to all costs in the system, even those retrieved before the change took place. This strategy doesn't require any versioning since the members of the type dimension only exists in one version, the latest.

5.5 Domain model

The analysis resulted in the domain model described in this section.

5.5.1 Concepts

Provider A provider is an actor that feeds the system with data. All data in the system are directly or indirectly related to a provider. In the current state of the system only one provider exist, namely STAX.

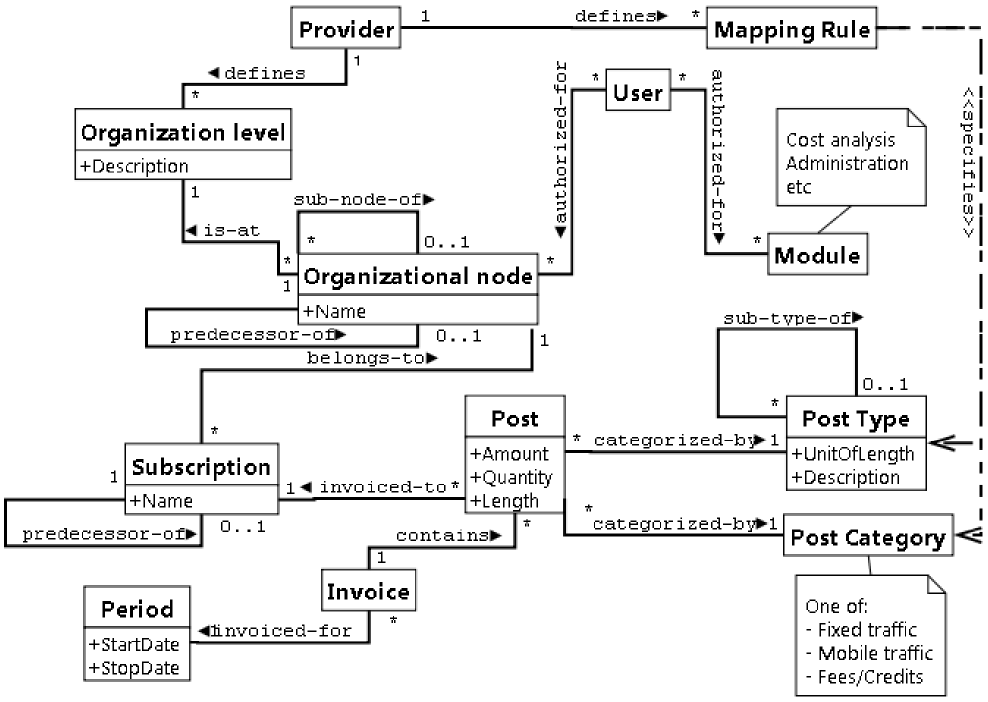


Figure 4: Diagram displaying the concepts and their relationships in the domain model

User A user is able to login to the system. Each user is authorized to analyze data for one or more organizational nodes and has access to one or more modules of the application.

Module A module is a sub-part of the system that provides the interface and functionality needed for a specific use case. Examples of modules are Cost analysis, Administration etc.

Organizational level An organizational level belongs to a provider and represents a level within the providersŠ organizational hierarchy. For example, STAX defines the organizational levels company, customer and cost location.

Organizational node An organizational node could be for example a company, a sub-division of a company or an employee. It is connected to an organizational level and could have a parent organizational node.

Subscription For cost analysis purpose the subscription could be seen as the lowest level of the organizational node hierarchy. The reason to separate the subscription concept from the organizational node is that other parts of the system are likely to use the subscription concept for other purposes.

Period A period simply defines a start and a stop date and represents a time interval.

Invoice An invoice belongs to a period and is associated to a number of invoice rows, or posts.

Post A post represents an invoice row and defines the measures of interest when performing a cost analysis. A post is directly or indirectly associated to the dimensions period, organizational belonging, category and type.

Post category A post defines the top-level category of posts and could be one of fixed traffic, mobile traffic or fees.

Post type A post type is used to describe the sub-type of a post. A post type could have a parent post type in order to structure the types within a hierarchy.

Mapping rule A mapping rule could be anything that maps the provider-specific categorization into a post category and a post type. For example, for STAX these rules defines the mapping between the textual descriptions into the concepts of the domain model. The mapping rules are applied as the system is fed with data.

6 Implementation

The implemented system contains a wide range of components and several third-party components and frameworks were used during the work. This section aims to explain the functionality, connection and implementation of these components for which some are described in detail while some are only briefly described.

6.1 Overall architecture

6.1.1 Deployment model

The system in its deployed state is built upon a three-tier architecture which involves a data storage tier, an application tier and a client tier. As for most web applications, the client tier resides in a web browser and uses web technologies such as HTML, CSS and javascript in order to provide an interactive user interface. The browser communicates over the HTTP protocol to a web server running on Internet Information Services (IIS) in which the application tier resides. The data in the system is located on a separate database server and contains a relational database running on Microsoft SQL Server 2008 and an OLAP Cube — where pre-calculated aggregations from the relational database are stored in order to allow for fast reading performance — running on Microsoft SQL Server Analysis Services. The web server and the database server are connected to the same LAN in order to provide a high-speed communication link between them. Both the web server and the database server are run upon the operating system Windows Server 2008.

Except from the IIS web server another application resides in the application tier. This application is called the Data Extractor and is responsible for periodically polling the data providers for data and — if any data exists — store this data within the data tier. The Data Extractor uses a command-line interface which makes it easy to schedule using the built-in task scheduler in Windows.

6.1.2 Application tier architecture

The application tier is developed upon the .NET Framework and uses a layered architectural style consisting of the layers data access, business logic and presentation. In this architectural style — which is based on the design pattern Separation of Concerns — each layer focuses on a specific area within the application and exposes its functionality by public interfaces [17]. The main reason for using layers is to promote reusability. For example, in the future it might be of interest to develop a user interface targeted to mobile clients. In order to achieve this only the UI layer needs to be rebuilt while the lower layers can be reused.

The layers resides in separate assemblies and interact to each other using a top-down approach which means that a layer only interacts with the layer immediately below itself and that a lower layer never calls a upper layer.

Each layer has access to a separate assembly containing a set of classes — referred to as business objects — that represents the data in the data tier. The business objects and the layers of the application are further described later on in this chapter.

An exception of the layered architecture is that the data extractor bypasses the business logic layer and in some cases also the data access layer. This is done in order to retrieve and to persist data that are not part of the business objects model, such as logging information and retrieval history.

6.2 Relational database design

The modeling of the relational database followed from the domain model described in the Analysis chapter. When going from a domain model to a relational schema there exist a number of rules-by-thumb that can be used [6]. Basically, each concept in the domain model is realized by a database table with a primary key and each attribute is converted to a column. For the relationships in the domain model different strategies are applied depending on the cardinality of the relationship. One-to-many relationships in the domain model are realized by using foreign keys, which are made nullable if the one-to-many relationship is optional. Many-to-many relationships are realized by using a cross reference tables.

These rules-by-thumb where thus applied for all concepts in the domain model. However, the modeling of the provider-specific fields associated to subscriptions and organization were still to be done.

6.2.1 Realizing the provider-specific attributes for organizations and subscriptions

In the domain model the concepts Organizational Node and Subscription only have one single attribute, name. This is enough in order to fulfill the filtering and drill-down parts of the use case but eventually the user might want to view the details for a specific item. The problem here is that these details might look different for different providers and different organizational levels.

One solution to this problem would be to retrieve the needed details from the provider by request, for example via a web service. This would not require any extra storage in the system and the providers can have full control over the data sent. However, this solution was not accepted due to the possible performance overhead when sending data over a network.

Instead, it was decided to create provider-specific tables where detail data associated to the affected concepts are stored. More specifically, each provider has at least one table containing subscription details and one table containing organizational node details, and each row in these tables has a reference to the subscription or organizational node it represents. If the subscription or organizational node details contains any one-to-many relationships additional tables can be created.

This solution is good for two reasons. First, it separates the data for which aggregations and calculations are performed from the data needed for presentation of details. Thus the related dimensions in the OLAP cube are unaffected as new providers are added to the system, i.e. no change of MDX-queries or dimension definitions needs to be done. Secondly, it allows the providers to use whatever relational model they want for their detailed data since no assumptions are made for it.

For the initial data provider, STAX, three provider-specific tables were created.

STAXSubscriptions contains the detail fields needed to display a subscription from STAX, such as installation address. Each row has a foreign key referencing the subscriptions table.

STAXSubscriptionProductRows contains the product rows associated to each subscription and has fields for product name, quantity and price.

STAXOrganizations contains the detail fields needed to display a company, customer or a cost location from STAX. Since the hierarchy of the organizations is merged into a single row this table contains a lot of redundant data but also allow for fast reads since no joins are needed in order to retrieve the full hierarchy of an organization.

6.3 OLAP Cube

6.3.1 Cube Design

The design of the relational database supported a relatively easy transition into an OLAP solution. The data were structured into tables that roughly could be translated into the fact table and dimensions modeled in the cube. The fact table Posts consist of the actual numerical data that is to be analyzed as well as the references to the dimension tables for the context the data is subject for. This fact table contains all the recorded rows from the corresponding database table although some additional meta-data is not added to the cube. The cube is built up from the following data view of the relational data:

- Subscription

  The lowest organizational like dimension. The members can be roughly thought of as the users of telephones in the organization. Not every fact row does reference a subscription however meaning it needs to be its own dimension instead of joined with Organization.

- Organization Contains the structural view of the CC's internal organizational structure.

- Post Type Parent-child dimension for grouping the fact rows on different types of calls as well as several different levels of granularity.

- Post Category Simple dimension with the members Mobile Traffic, Fixed Traffic, Fees.

- Invoice Acts as a time dimension giving a context of for what period the facts reference.

- Facts Posts The fact table of the cube and here can be found the facts for calculations —including amount, number of calls etc.

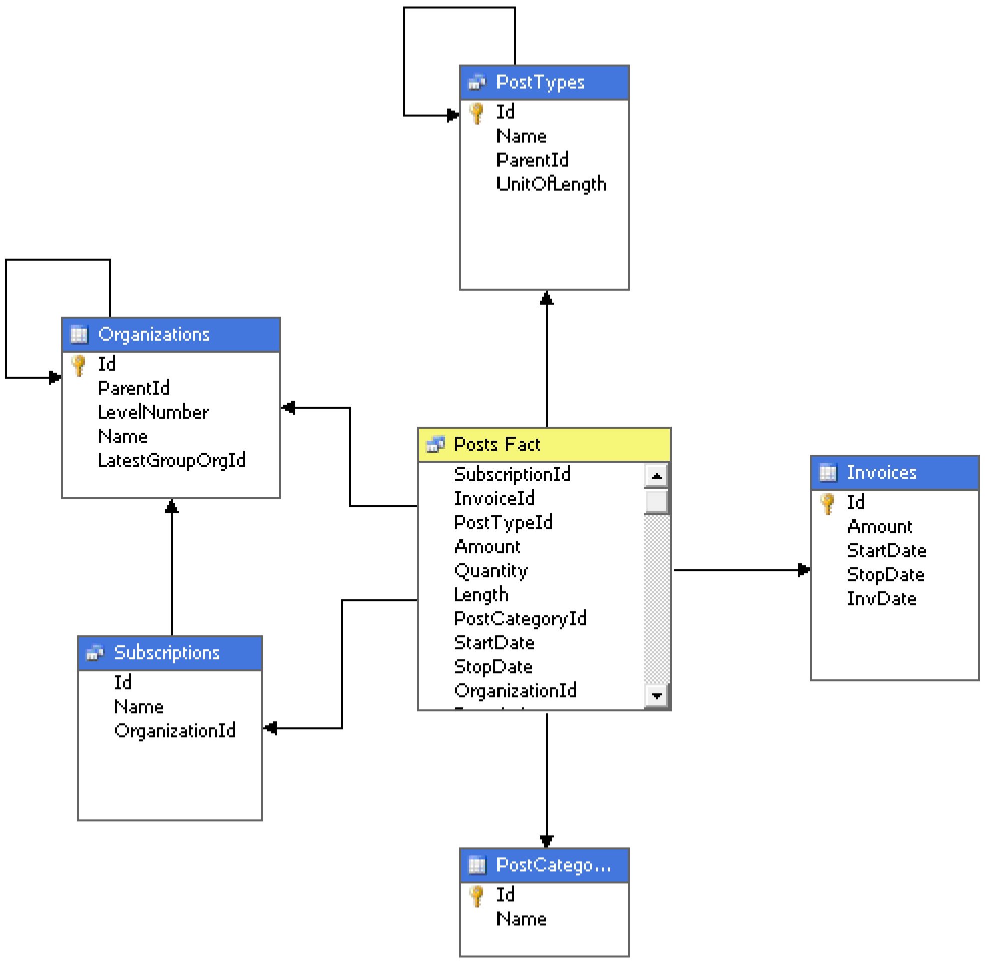


Figure 5: Model of the Cube

6.3.2 Aggregations

The data in the system does not in fully conform to the wanted properties for a preferred data source to gain the perfect OLAP performance boost. Specifically both the Subscription and Organization dimensions contain a fairly high amount of members that increases in a somewhat logarithmic way with respect to number of fact rows. While the dimension Subscription does contain a huge amount of members —possibly ranging in the rank of millions— it is not as frequently used in analysis as other the dimensions. The user may only opt to analyze the costs grouped by subscription at the latest last level of the application. In combination with this a user only has need to view a small —but hopefully the most interesting— portion of all possible subscriptions at the same time which further decreases execution time. In the case of a user actually wanting a full report of all subscriptions the duration of retrieval is not as important. Still, the high amount of members and being rarely used means that if the processing time gets unfeasible and/or storage load becomes too great the dimension Subscription is certainly a likely candidate for run-time analysis and calculations.

The Organization dimension consists of three different levels of granularity with the top level including a fairly limited number of members and levels below exponentially increasing with the total number of members at the very bottom ranging in numbers of possibly hundreds of thousands. Although the higher levels will be more frequently used all layers can be accessed at every stage in the program. This means performance could be vital even at the bottom and a longer processing time and bigger storage space may be acceptable in this case. To improve performance and processing time the data can be divided into physical different partitions on the hard drive. For example creating partitions for the members or groups of members that are known to be related to each other on the top level of Organization would mean only partitions affected by newly inserted data would need to be reprocessed. Also users query only a part of the full cube (all partitions) which slightly improves performance.

6.3.3 Processing

It is possible to have a more or less real-time OLAP solution that with quite short latency reflects the current state of the underlying data source(s) [8]. While this is a preferred solution in situations with frequent changes to the data and the need to appear like a real-time system it comes with a price of automatic polling and less control as well as performance decrease while refreshing the state. For the current system the updates to the relational database occurs on an automatic known schedule and thus all that is needed is to start the incremental processing of the cube after the input to the data source has been completed by the Data Extractor, as described in next section.

6.4 Data extraction

Too feed the system with data a console application called the *Data Extractor* was implemented. In general the application performs three tasks, each one of them is executed by providing the command-line interface with an appropriate flag.

1. Poll the provider for to see if there are any new or updated invoice data to retrieve, referred to as *header retrieval.* 2. Poll the provider for the actual invoice data needed, parse it and store it in the relational database, referred to as *detail retrieval.* 3. Trigger a refresh of the OLAP cube.

For the two first tasks, the application uses a web service method exposed by STAX in order to retrieve the data. The web service method returns a tabular data set represented as a XML-document where each node corresponds to a row and each attribute corresponds to a column. Two different types of data sets are used; one that contains header information for the invoices on a given agreement in STAX and one that contains the actual invoice details for a specific invoice. The latter thus contains the data of interest while the first one is used in order to know for which invoices details should be retrieved.

6.4.1 Header retrieval

The header retrieval is performed for a given set of agreements and for each agreement a data set is retrieved from the web service. This data set contains the identifier and a couple of aggregated values — such as total cost and number of rows — for all invoices on the given agreement. The data extractor maintains this information, together with a timestamp, a retrieval status and a processing status, in a log table that is used in order to control for which invoices details are requested. On retrieval, each row in the data set is iterated and the log table is searched for an entry with the same invoice identifier. Either of the following takes place:

- If no entry in the log table, the row is inserted, assigned retrieval status "not retrieved". - Otherwise, if one or more entries are found, compare the aggregated values of the last retrieved entry to those of the row. If changes are detected, the row is inserted and assigned retrieval status "not retrieved". If also the found entry has retrieval status "not retrieved", delete this entry from the log table.

Note that in any case that data is written to the log table by during the header retrieval the current timestamp and the processing status "not processed" are assigned to the row.

The reason to include the aggregated values in the header information is to detect possible changes to the invoice data in STAX. In some cases, invoices might be re-generated within STAX and a change in this data will most probably be reflected in the aggregated values. By using this strategy these changes are eventually reflected in the system.

6.4.2 Detail retrieval

The actual invoice data is retrieved by iterating the entries of the log table for which the retrieval status are set to "not retrieved". For each entry the web service is queried using the invoice identifier of the entry. When the invoice details are inserted into the database the corresponding entry in the log table are assigned the retrieval status "retrieved".

The received data set contains all invoice data that is needed, which includes related data such as organizational belonging, subscription information etc. This means that the data set is heavily denormalized and contains a large portion of redundant data.

The mapping of the denormalized data into the relational database follows a recursive pattern. Each concept is assigned a custom helper class, called extractor, which is responsible for extracting the fields of the data set relating to that concept and to create a corresponding row in the appropriate database table if no such row already exists. In order to not violating any foreign key constraints the extractors must thus first ensure that any related data are extracted. The rows of the data set are thus sent one by one to the extractor representing the lowest-level concept, namely Post, which in turn starts by invoking the extractors for the related concepts Type and Subscription before it continues execution.

For the extractors associated to the Category and Type concepts the provider-specific mapping rules are applied. The extractors reads the textual description of the invoice row and uses a lookup table in order to map it to a category and a type. In the case when no mapping exists the post is assigned to a special type and category representing unknown descriptions, until it is appropriately mapped by a system administrator.

In order to check whether the data already exists in the database is performed by using the numerical identifiers coming from STAX, which are stored in the provider-specific tables described earlier. In some cases, recalling the versioning strategy that was described in the Analysis chapter, other fields are also used in order to determine whether the data has changed or not. The checks for existence requires a large amount of lookups to be performed and in order to avoid polling the database with SELECT-commands each extractor maintains a bit-array where the identifiers forms the indices. The bit-arrays are filled on first request, which of course could take a number of seconds, but was in the long term more efficient since the lookup time was heavily reduced.

6.4.3 Refreshing the cube

The refresh, or processing, of the cube is performed by first checking whether or not any non-processed invoices exists in the log table. If so, the AMO API is used in order to trigger an update of the related dimensions needed before processing of the fact rows. What really happens is firstly creating a temporary new empty partition with the new fact rows being added and processed within this. In the next step this temporary small cube is merged together with the existing big one with adding together the calculations across the aggregated levels. The outcome is a new ready-to-use repository in a much shorter time than it would have taken to reprocess the entire cube from scratch. The primary objective here is to avoid having to perform a full reprocess of the cube since this may take too long time rendering the system either unusable or with decreased performance for an unnecessary amount of time. The current solution supports this with adding new fact rows to the cube taking a relatively short period of time.

6.5 Business objects

The business objects are used in order to represent the data in the data tier and can be seen as the carriers of information between the layers of the application. The components of the data access layer are responsible for translating the data in the data tier into business objects and vice versa. The business logic and the presentation layer might then work with these objects and does not need to depend on any storage-specific representation of the data.

There are two types of business objects, business entities and OLAP objects. The business entities represents data from the relational database and are automatically generated by the object/relational mapper (O/RM) Telerik OpenAccess, but might be manually extended and customized if appropriate. The business entities exist in both a connected and a disconnected state. The former is used within the data access and business logic layer where a persistence context, or scope, is available. The scope is managed by Telerik OpenAccess and keeps track of changes made to the objects. Disconnected, or detached, entities are used in the presentation layer where no scope is available.

The other type of business objects, OLAP objects, represents data retrieved from the OLAP cube. These objects have a very basic structure and generally contain numerical values of the aggregations retrieved from the cube together with a label describing the dimensional member for which the aggregations are performed. Since it makes no sense to write this data to the OLAP cube these objects are made read-only.

The two OLAP objects referred to in this reports are PostRow and CompareRow, both containing the measures amount, number of calls, call length and number of subscriptions together with a label describing the row. The compare row also contains an additional value and is used to represent a difference in amount compared to a given period.

6.6 Data access layer

The data access layer has two major functions. First, it provides an interface to the components in the data tier by offering functionality for querying and persisting data. Secondly, it acts as a translator between the data from the data tier and the business objects. The data access layer is separated into two modules, one accessing data from the relational database and one accessing data from the OLAP cube.

6.6.1 Relational database access layer

By using Telerik OpenAccess not only the business entities are automatically generated but also the entire relational data access layer, supporting query languages such as OQL and LINQ as well as lambda expressions on the fly. All methods for retrieving and persisting data using Telerik OpenAccess are performed using an instance of an object scope, which lifetimes are decided by the business logic layer since a business method might require several operations to be performed within the same scope. Below is two examples, one showing how to query data and one showing how to persist data using Telerik OpenAccess.

- Query using lambda expression and Telerik OpenAccess

  ```
  //Retrieve the modules authorized for the given user
  IList<Module> modules = context.Extent<ModuleRequest>().
     Where(mr=>mr.User.Equals(u) && mr.IsAccepted).
     Select(mr=>mr.Module).ToList();
  ```

- Code that demonstrates how to persist data using Telerik OpenAccess:

  ```
  // Initiate a transaction
  context.Transaction.Begin();

  // Create a new user
  User u=new User();
  u.Name="Test";
  context.Add(u);

  // Persist the changes
  context.Transaction.Commit();
  ```

The OLAP data access layer is used for querying the cube for the wanted information as well as converting the results into objects that can be understood also by a higher level. Given the filter that is to be applied to the cube together with the class and method called the layer then puts together a valid MDX string which is used against the analysis services. It then parses the result and creates objects holding the information wanted, such as the name of a post type, total amount and number of calls for the periods wanted. The filter itself consists of lists of the organizations to be part of the scope, the level of analysis, the invoices for setting the period etc.

The internal structure itself is as a whole strongly component based in idea. One class is responsible for the cube connection while another knows the actual interface of the cube. On a finer grained level another group of classes puts together the different parts of the MDX query depending on what has been requested. These parts are then translated by a component into the resulting query. This means that the functionality is well encapsulated and the components not connected with the nature of the specific cube or the information coming from other layer could be used in other systems as well.

OLAP Base OLAP Base is the only class that actually knows about the interface — containing name/id of dimensions, measures, hierarchies etc — exposed by the cube. Apart of defining objects and variables describing the exposed members the class also contains common stored procedures and using these definitions for converting objects coming from the service layer into the corresponding OLAP representation. For example a method would return a valid MDX set of the representation of a list of organizations given by the in parameter of an ORM collection of organizations. Typically used by classes needing to be able to be able to communicate with the analysis server.

IMdxQueryProvider Classes implementing the interface must expose a method that returns an object consisting of a valid MDX query. These fine grained classes typically extends the OLAP Base class for understanding the structure of the cube as well as getting access to methods assisting in more low level work. Implementing classes would then use the information about the wanted analysis coming from the service layer to construct the different object like parts of the final statement. These parts are then put together to the valid string to be queried by an MdxQuery object.

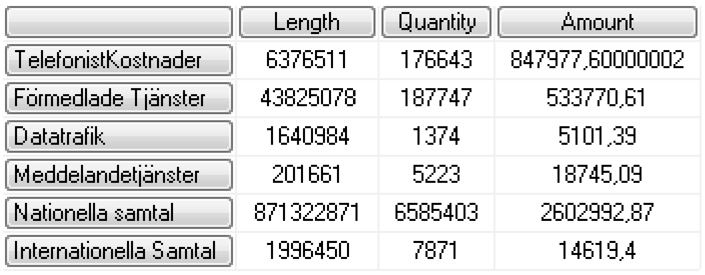
MdxQuery While the other classes responsible for creating the MDX queries preferably use as little MDX specific syntax as possible this class does the exact opposite. All it really does is exposing methods for adding different parts of an MDX statement. Typically a collection of strings— representing the measures requested in the analysis— are used as in parameters to the AddColumns method. Furthermore, methods for adding members to the rows, sub scope, where clause etc. are exposed as well. The class then uses all this info to put together a final valid MDX query which can be run against the cube.

6.6.2 Examples of MDX queries constructed and result received

```
SELECT
  {
    [Measures].[Length],[Measures].[Quantity],[Measures].[Amount]
  } ON COLUMNS
 ,NON EMPTY
    {
      [PostType].[PostTypes].[TrafficCosts].Children
    } ON ROWS
FROM
(
  SELECT
    (
      {
        [Organization].[Organizations].&[99136]
       ,[Organization].[Organizations].&[99139]...(and more)
      }
     ,{[Invoice].[InvoiceKey].&[3710]}
    ) ON COLUMNS
  FROM [Cube Posts]
)
WHERE
  [PostCategory].[PostCategoryKey].[Fixed Traffic]
```

6.7 Business Logic layer

The task of the business logic layer is to provide an interface to the business methods of the system. The business layer applies the facade pattern and is split among different *managers*, each one providing functionality needed for a specific module of the system. Thus there exists one manager for user



	Length	Quantity	Amount
TelefonistKostnader	6376511	176643	847977,60000002
Förmedlade Tjänster	43825078	187747	533770,61
Datatrafik	1640984	1374	5101,39
Meddelandetjänster	201661	5223	18745,09
Nationella samtal	871322871	6585403	2602992,87
Internationella Samtal	1996450	7871	14619,4

Figure 6: Resulting matrix from above query

authorization and one manager for cost analysis. Common functionality used in these managers are either encapsulated in internal helper-classes or provided by an abstract base class called the BaseManager, from which every manager derives.

Except from encapsulating the business logic in the system, each manager is responsible for maintaining data integrity, i.e. users should only be able to read and write data that they have access to, and therefore needs to be run in the context of a user. This could be implemented by forcing the caller to provide user credentials with each call, but in order to avoid excessive code the service layer stores the user credentials in memory at successful login by using the built-in HttpContext maintained by ASP.NET [12].

When calling the relational data access layer, the managers are responsible for providing an open and valid database scope. Following the guidelines from Telerik it was decided to initiate one scope per HTTP request and then dispose it as the request finish [13]. Since the scope is not accessible for upper layers any business entities returned from the business layer must be detached from the scope and, similarly, attached to a scope again scope when passed from the presentation layer.

6.7.1 Managers in the application facade

BaseManager All managers share some common functionality, for example they all need access to the current user and the database scope. For convenience, this functionality is encapsulated into a abstract base class called BaseManager. The abstract member of this class is a property telling whether or not the current user is authorized to the module associated to the manager. In the constructor, the base manager evaluates this property and makes sure that an exception is thrown if appropriate. As shown later, the presentation

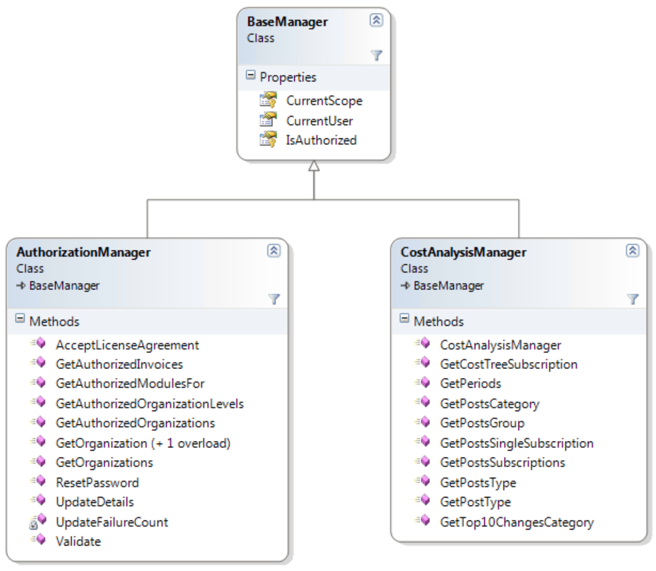


Figure 7: The managers in the business logic layer

layer performs a similar check but it was decided to keep this extra layer of security in order to allow for reuse of the business layer.

Authorization manager The authorization manager contains functionality that should be accessible by any user. Among them are authenticated-related functions such as user validation, reset of passwords as well as functions for retrieval of authorizes modules and organizations.

Cost analysis manager The interface to the cost analysis facade was identified directly from the steps in the cost analysis the use case.

For the retrieval of periods, a method called GetPeriods() is used in order to return the periods for which invoice data exists for the current user. The business object Period is used in order to represent the data.

In the current state of the category and type hierarchy contains of at most three levels. Thus, three methods exist, each returning invoice data grouped by a specific level in this hierarchy. These methods are very similar and in large parts share the same signature. All methods take a set of organizations, a period and an optional period for comparison as input. The two methods returning data from the second and third level of the hierarchy takes additional arguments specifying the parent type or category to be filtered on.

The methods create a dictionary with the periods as keys and a collection of PostRow-objects as the values. The OLAP data access layer is queried for data for each given period and the retrieved collection of PostRow-objects are added to the dictionary together with the period before the dictionary is returned to the caller. Comparisons between the periods can then be performed with respect to any value in the PostRows, since all data is returned and associated to the corresponding period.

The retrieval of subscriptions and organizations are performed in a similar way with one the major difference, the comparisons of the periods are performed within the OLAP cube and the data returned is not divided by period. The reason for this is that only a limited number of rows are returned and in order to select what rows it was decided to use the ones for which the amount has grown most. For performance reasons this sorting must be performed by the OLAP cube. Therefore another type of business object, CompareRow, is returned.

6.8 Presentation layer

The presentation layer is built by using the ASP.NET web forms framework and contains the logic for rendering the user interface of the application. The graphical look and the interaction flow was developed using CSS, HTML and javascript and followed the graphical sketches that were provided from the external interaction company.

6.8.1 User authentication and authorization

The user authentication and authorization is implemented by utilizing a couple of built-in mechanisms and features that ASP.NET provides. First of all, a number of built-in server controls are used in order to create a user interface and functionality for such things as logging in, resetting and encrypting passwords etc. These server controls uses an abstract base class called a membership provider, for which an implementation is done. On successful login, the membership provider creates an authentication ticket, stored in a cookie at the client's web browser, which is sent to the server on each request in order to verify that the user is authenticated. If no authentication ticket exists, or if it is expired, the client is automatically redirected to the

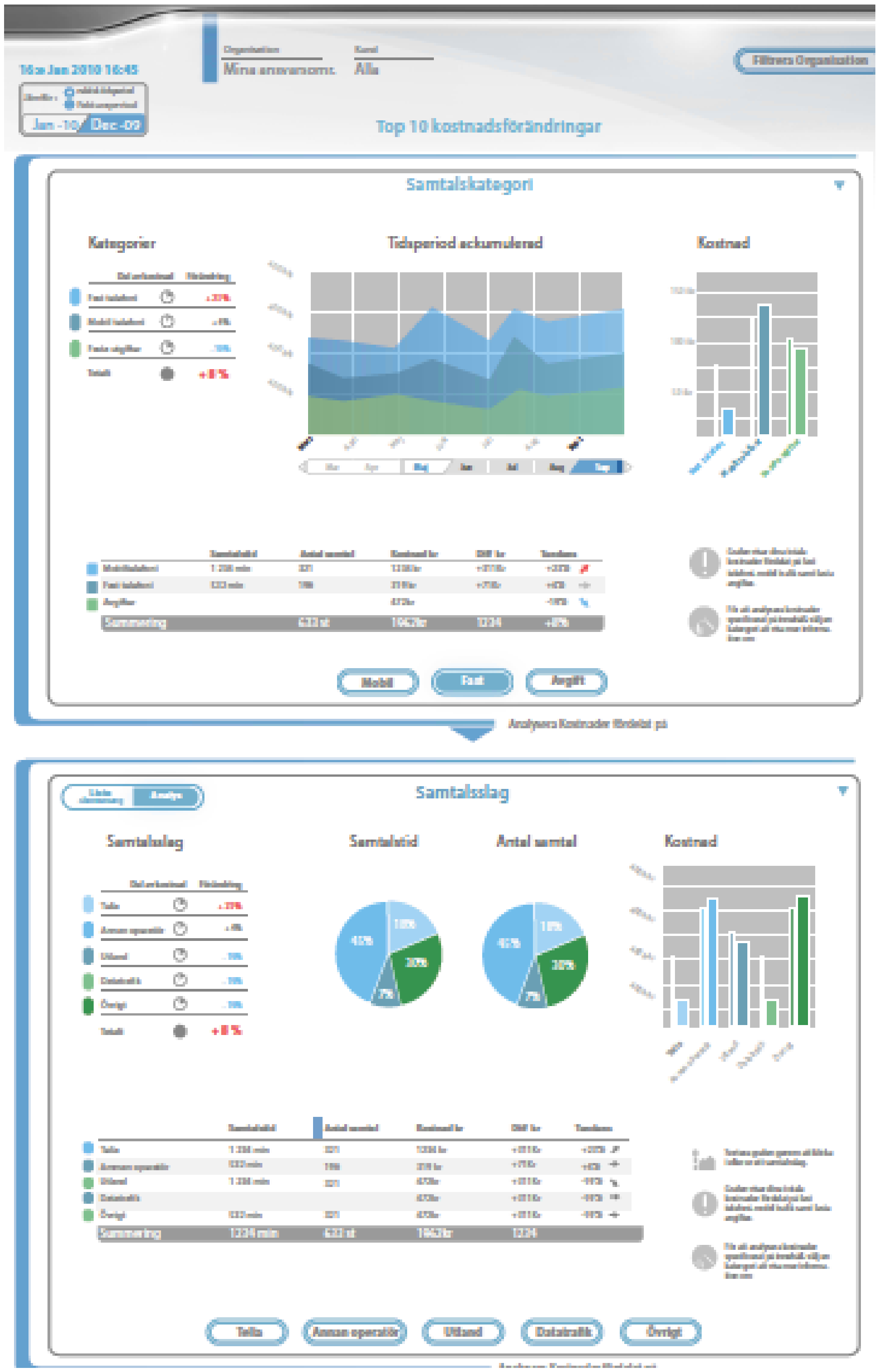


Figure 8: The graphical sketch of the analysis web page

login page. In the custom implementation of the membership provider basically three methods are implemented by hand; one verifying that a given username and password is valid, one resetting the password for the user and one increasing the number of failed login attempts for a specific user. The remaining logic is performed automatically by the base class. For the custom implementation, the authorization manager class in the business logic layer is used.

Using the membership provider keeps non-authenticated users out of the system, but since different users are supposed to have access to different areas, or modules, of the system a role-based access control is needed. This is achieved by mapping each module to a specific role and by using another security-related feature of ASP.NET, the role provider. Similarly to the membership provider, the role provider is an abstract class providing the base functionality for role-based security. The custom implementation of the role provider are responsible for retrieving the names of the modules that a specific user are authorized to access and the role provider are then automatically performs a check in each request whether or not the user are authorized to the requested content. A configuration file is used in order to specify what content that is accessible by what roles. All web forms associated to a specific module is then added to this folder and is then automatically protected for non-authorized users.

Since both the membership and role provider uses a configuration file changes such as shortening the expiration time of authentication tickets or granting access to a specific module can be performed without the need of redeploying the system.

6.8.2 State, postbacks and asyncronous requests

During the drill-down of the cost analysis use case, the user should see the old choices made, i.e. the entire use-case must reside on one single web page. In other words, as the user navigates through the type hierarchy, the server must be contacted and render the new data as the old resides in the same place as before.

In ASP.NET, the classic way to achieve this to use the built-in postback mechanism. The advantages of this are that it is built in and rather trivial to use. The downside is that we need to render the entire web page again, even the parts that are already rendered, and send the full content back to the client. An alternative is to use an ASP.NET AJAX postback instead, where only a small part of the web page is rendered by using. This reduces the data amount sent to the client.

This disadvantage of both these alternatives is that they both cause a full postback to the server. The entire life-cycle of the page is executed, and the viewstate is processed and sent back to the client. A third alternative is to use a mechanism called ASP.NET client callback instead, causing only a small part of the page life cycle to be executed. This approach still sends approximately the same amount of data to the client, but reduces the amount of server-processing.

When comparing these alternatives the callback mechanism turned out to be the far most efficient one from a performance perspective [13]. Since client callbacks requires a relatively large amount of client-side scripting to be written a component, called RadXMLHttpPanel, is used to utilize the implementation.

Thus, each updatable part of the cost analysis web page is put into a RadXMLHttpPanel, or simply a panel. The panels might contain regular server controls and/or HTML-content. Typically the panels contain a data table and a number of chart controls as described in the Technology chapter. Each panel are associated to two client-side event handlers; one post request handler that are invoked after a client callback has been performed and one error handler that are invoked in cases when a HTTP error occurs. In order to trigger a client callback a function within the client-side API of the component is invoked and passed a string argument that is sent to the server through a HTTP POST request. As the response is retrieved from the server, the content of the panel is updated and the post request handler is invoked.

Since the server cannot access the client states of the UI components when, their state must be explicitly sent together with the HTTP POST request in order for the server to know which data to query for. Thus, before a panel is updated the states of each UI component of interest are concatenated into a comma-separated string containing key/value entries corresponding to the parameters needed by the server.

The HTTP POST requests triggered by the panels cause an event-handler to be executed by the server. Each panel has a custom event handler which is responsible for retrieving the requested data from the business layer and then to update the server controls within the panel accordingly. The arguments sent to the business layer are set by parsing the comma-separated string that was passed with the request.

6.8.3 Combining scripts and CSS files

From a maintenance point of view there is of importance to modularize the CSS-files and javascript-files used in the application into several files. On the other hand web browsers perform better the smaller the number of request are, so one large request is generally faster than several small since CSS-files and JS-files are not downloaded in parallell [14].

The presentation layer utilizes mechanisms provided by Telerik that merges the CSS-files and JS-files into one single file. This file is generated on first request and then resides in the server memory.

7 Performance testing

Through the development process the performance of the system and its components were continuously evaluated in order to support the selection of technologies and patterns used. This part will describe how the tests were performed as well as presenting some of the.

Maintainability is often achieved by using well-known design patterns and by dividing the system into independent, specialized components, also known as *separation of concerns*. However, since these design patterns mainly considers the reusability of the system they are in many cases associated with a performance cost [17]. The aim of the work was to create a reusable and flexible system while it at the same time should be able to deliver the response at an acceptable speed.

The purpose of the performance testing was twofold. Except from providing support for the selection of technologies and design patterns used, the purpose was to identify where the bottlenecks in the system are located when heavy loads are generated.

7.1 Web performance testing

In order to achieved useful test data it is necessary to develop a performance test plan, where the following questions are considered:

- What should be tested? Which test cases do we need?
- What measures should be done?
- How to generate the load?

When developing the test cases it was decided to target the tests to the system as a whole, rather than the individual components. By using this approach not only the individual components are implicitly tested, but the interaction between them is tested as well. So, in order to achieve realistic test cases the presentation tier of the system —- the web application — was targeted.

In order to simulate a visit to the web site, a way to generate and send HTTP requests to the web server is needed. Visual Studio Ultimate version can create so called web test, which does exactly that. It is as simple as pressing a "record"-button, browsing the web site in a web browser and then pressing "stop". During the recording, Visual Studio logs all requests made and saves them into the web test. When running the test later on Visual Studio generates the very same HTTP Requests to the server as during the recording.

The web test follows the main path in the cost analysis use case and when recording it, the following actions were performed:

- Login to the system.
- Load analysis page.
- Select category "Mobile traffic".
- Select call group "International calls"
- Select call type "Received international calls".
- Select the first subscription in the list.

One problem with this web test is that it is not parameterized. When re-running the test it will always use the same user credentials and the selected drill-down path as well. Running 50 instances of this test simultaneously is not very realistic and might probably give a better result than if 50 different users — selecting different paths were using the web site, due to caching involved at different levels of the system.

In order to solve this problem, a plugin were developed for each step in the test case. A web test plugin could handle different events in the execution of the test and data can be shared between the different steps in the web test by using a globally accessible context. Most importantly two of the exposed events are used; the pre-request event, which takes place before the web test generates the request, and the post-request event that occurs when the response from the web server is retrieved. The pre-request handlers are used in order to generate random input data and then modify the appropriate request parameters sent to the server. The post-request handles are used in order to generate the data for which the randomization is performed.

More specifically, in the plugin for the first step where a user is logged in to the system, a pre-request handler is used in order to randomize among a set of pre-defined username/password combinations. The appropriate request-parameters are then modified and upon request sent to the server.

For the steps in the performed during the drill-down, the outputted sub-types are identified in the post-request handler by parsing the HTML-response. One of the sub-types is selected randomly and added to the web test context. In the pre-request handler for the next step the appropriate form parameter values are then set by using the sub-type specified in the context.

By doing this parameterization we achieve a more realistic scenario when running many instances at once.

7.1.1 Load generation

In Visual Studio a web test can be executed in isolation or within a load test. In a load test the appropriate web test is selected and parameters — such as number of users, running-time etc — is defined. The load test can thus be used to run the web test in many instances simultaneously and when done a detailed summary is displayed describing the response-times, throughput for each one of the steps performed in the web test.

7.1.2 Measures

The main measure of interest was the throughput of the system, i.e. the number of requests that could be handled per second.

7.2 MDX performance compared to SQL

As a way to more fully illustrate the performance difference between the OLAP solution and using SQL to extract the information directly from the relational database a number of tests were performed. The tests were designed from real usage of the application with the MDX statements translated roughly into similar resulting output in SQL. The result clearly shows some interesting differences and how superior pre-calculated aggregated data can be for analysis in comparison to a normal relational database.

7.2.1 Testing environment

Both the MDX and SQL statements were inserted and executed by SQL Management Studios with SQL Profiler monitoring the duration of the execution. All tests were run several times to realize if any unexpected spikes would occur although none were observed and every resulting value was close to the average. Worth nothing is the relational database was in fact somewhat designed for an easy transition to the OLAP system and the cube consists of aggregated data on the recommended default levels. This should give a solid reasonable basis for the tests.

7.2.2 Design of tests

The goal of the testing was to highlight any main differences in the length of response from the MDX and SQL queries depending on the type of request. There were basically three connected areas of interest that the tests were designed to give interesting feedback for:

- Proof of concept that the multidimensional model performs better for the current analysis system.
- Showing —if any— the major differences of how aggregated data gives different response times than requests for data at a lower level.
- Both the relational database and the analysis server comes with a built-in cache for quickly answering new queries related to information retrieved earlier. In what way does this affect the results?

The MDX queries used for the tests are those given by a sample run of a user using the application for analysis from the top to the bottom. All four levels shows the result for the measures Amount, NrOfCalls, Length, NrOfSubscriptions with the rows on each level corresponding to an increasing amount of members. The first level features only three different PostCategories and the next two levels relates to two different levels in the parent-child hierarchy PostTypes with still a somewhat limited number of members —the lower level in the order of dozens. The lowest level displays individual subscriptions which currently rank to several hundreds of thousands of rows inserted into the database. The number of organizations equals to about forty two thousand and the total number of Post fact rows is just above five million.

Important to note is that the translation of the MDX queries into an SQL counterpart does not give exactly the same output but what is wanted is the essence of performing roughly the same calculations. Specifically the main difference is the SQL will not group the resulting calculations on the correct level of PostType but this is a simple calculation in any case. As well as this the multidimensional analysis for the subscriptions calculates the difference between two periods while the similar SQL query does not. However this should not affect more than a slightly increased absolute value of the duration. Further for all other statements the Invoice dimension is used for stating the single period/time of interest while the set of organisations are always on a high level within the hierarchy.

In total there were four different requests during the test which in syntax were all somewhat similar. For comparison reasons most of the query for the second request can be found below, both for MDX as well as SQL.

- MDX query for analyzing on fixed traffic for given period and organizations

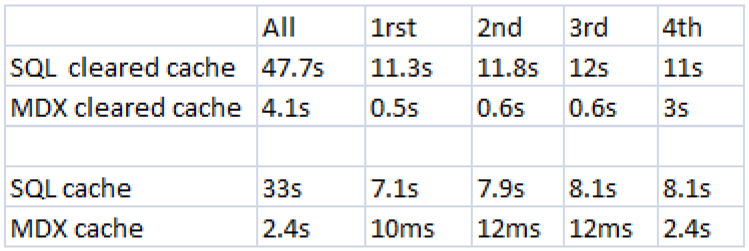
```
SELECT
  {
    [Measures].[Length]
   ,[Measures].[Nr Of Calls]
   ,[Measures].[Amount]
   ,[Measures].[Number of Subscriptions]
   ,[Measures].[PostType Id]
  } ON COLUMNS
 ,NON EMPTY
    {
      [PostType].[PostTypes].[Traffic Costs]
     ,[PostType].[PostTypes].[Traffic Costs].Children
    } ON ROWS
FROM
(
  SELECT
    (
      {
        [Organization].[Organizations].&[1234]
       ,[Organization].[Organizations].&[1235]...
      }
     ,{
        [Invoice].[InvoiceKey].&[1234]
      }
    ) ON COLUMNS
  FROM [Cube Posts]
)
WHERE
  [PostCategory].[PostCategoryKey].[Fixed Traffic];
```

- The translation into sql

```
SELECT PostTypeId, SUM(Amount), SUM(Quantity), SUM(Length),
COUNT(Distinct SubscriptionId) FROM Posts
WHERE OrganizationId IN (
 SELECT Id FROM Organizations WHERE ParentId IN (
  SELECT Id FROM Organizations WHERE ParentId IN(1234,1235)
 )
)
AND InvoiceId IN (3710) AND PostCategoryId=1
AND PostTypeId IN (
 SELECT pt1.Id FROM PostTypes pt1
 INNER JOIN PostTypes pt2
 ON pt1.ParentId=pt2.Id INNER JOIN PostTypes pt3
 ON pt2.ParentId=pt3.Id WHERE pt2.ParentId=42
)
GROUP BY PostTypeId
```

7.2.3 Results

All four tests were individually executed with both cleared cache before test as well as letting the cache warm up a little with a few similar queries and then execution



	All	1rst	2nd	3rd	4th
SQL cleared cache	47.7s	11.3s	11.8s	12s	11s
MDX cleared cache	4.1s	0.5s	0.6s	0.6s	3s
SQL cache	33s	7.1s	7.9s	8.1s	8.1s
MDX cache	2.4s	10ms	12ms	12ms	2.4s

Figure 9: Table over results from MDX vs SQL comparison

The results above does in fact clearly indicate several interesting notions. The first and most obvious being that the time needed for the analysis server is highly significantly lower than that of the relational database for every type of tested request. Further the impact on performance from data being aggregated with different granularity is also significant. The relational database which does not do any aggregations responds within a similar length of time to all requests while the OLAP behaves differently. The MDX query retrieves results for the aggregated and rather limited number of members on the first three levels much faster than the lowest level displaying results for subscriptions —a relatively high number of members— go a lot slower. Lastly the cached effect shows this even further: the relational database saves only a seemingly fixed linear time while the cube actually responds pretty much instantly to the first three queries while taking almost as long as before for the last one.

8 Discussion and Conclusion

8.1 Discussion

A concern in the beginning of the work was whether or not the OLAP cube would be able to offer the functionality needed. By spending the first weeks on getting a good theoretical base in this area the problems and possibilities related to data warehousing was pretty clear early in the project.

8.1.1 OLAP Access Layer

The architecture could of course have been designed in many different ways with varied levels of encapsulation and extensibility. Also the communication and queries themselves could be set up in different ways. The adomd framework supports the use of olap objects such as dimension or a cube but really using these for smoothly running different queries does not really help anything in terms of complexity, one still need to know what one is doing and then there is no need. Not to mention the difficulty to properly debug while learning If using existing components one do not understand. Quite another possibility though would be to use reports or customized expressions on a pre-set basis that would then require parameters as input. This may certainly be a viable option if cube structure is not very likely to change a lot and most or all of the required queries are already specified and set. However, in this case with an agile development of both the system as well as the cube it would be far from optimal due to the small changes that would be needed would take far longer on the lower level of implementation.

8.2 Conclusion

The work that was performed resulted in a web application running on top of a data warehouse and offers all functionality that was stated in the initial cost analysis use cases. The final product we beleive is a good example of a business intelligence tool that offers consistent and useful analysis options together with a good response time with which users can base possible decisions on.

One of the key factors for the project to be finished within the desired timeframe was that the development of the web interface and the underlying infrastructure utilized a large number of third-party components and frameworks. This minimized the hours spent on implementation work and focus could therefore to a large amount be directed to the problem domain rather than technical details.

During the process, the performance and extendability was throughously considered. However, neither of these are easily measured. The performance of the system might be tweaked in many ways but the main focus was to acheive a scalable system, for which reason it was architected as a 3-tier solution. However, the response time for which invoice data is retreived is drastically improved when comparing to the intial system. We can conclude that the largest contributor to this performance gain was the using of an OLAP cube, which was not very surprising since calculating data run-time cannot beat the performance of precalculated data, no matter of CPU power etc.

There are several different important building blocks needed for developing an OLAP solution and all vital for a good behaviour of the system. Much work has been focused on writing efficient MDX queries and the overall design of the cube as well as an automatic processing of the data. While we are content with the final outcome further work with design and optimization could still be made. Because of too many members of one or two dimensions the data from the underlying system was not completely conformant for the optimal performance increase for a multidimensional architecture with precalculated results. However the user always starts the analysis of data from the top meaning the system in most cases responds rapidly in a way that would not be possible with a normal relational database.

Future developement of the application should focus on both new analysis functionality for end user as well as optimizing performance. Continous performance testing and redesign of the cube together with it's access layer is likely to be needed in the future as well. Extended functionality and increased performance must however not restrict the current scalability of the system due to the fact requirements are expected to change in the future as well.

References

[1] Matt Carroll. Sql server best practices article. Technical report, Microsoft, March 2007.

[2] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1):65–74, 1997.

[3] Peter Evans. Business intelligence is a growing field. `http://www.databasejournal.com/sqletc/article.php/3878566/Business-Intelligence-is-a-Growing-Field.htm`, 2010.

[4] Mark Gschwind. Choosing olap or aggregate tables in an aggregate strategy. `http://www.information-management.com/specialreports/20030218/6364-1.html`, February 2003.

[5] P. Hylandy H. Hasan. Using olap and multidimensional data for decision making. Technical report, Faculty of Commerce, University of Wollongong, 2001.

[6] Jennifer Widom Hector Garcia-Molina, Jeffery D. Ullman. *Database Systems: The Complete Book.* Prentice Hall, October 2001.

[7] Edward Melomed Irina Gorbach, Alexander Berger. *Microsoft SQL Server 2008 Analysis Services Unleashed.* Sams, December 2008.

[8] Justin Langseth. Real-time data warehousing: Challenges and solutions. `http://www.databasejournal.com/sqletc/article.php/3878566/Business-Intelligence-is-a-Growing-Field.htm`, August 2004.

[9] Michelle Wilkie Mary Simmons. Best practice: Optimizing the cube build process in saső 9.2. Technical report, SAS Institute, Cary, NC, 2009.

[10] MicroStrategy. The case for relational olap. `http://www.cs.bgu.ac.il/~dbm031/dw042/Papers/microstrategy_211.pdf`.

[11] MSDN. Key concepts in mdx. `http://msdn.microsoft.com/en-us/library/aa216772(=SQL.80).aspx`.

[12] Microsoft Developer Network. Msdn library. `http://msdn.microsoft.com/en-us/library/ms123401(v=MSDN.10).aspx`.

[13] Telerik Developer Network. Telerik developer blogs. `http://blogs.telerik.com/`.

[14] Yahoo Developer Network. Best practices for speeding up your web site. `http://developer.yahoo.com/performance/rules.html`.

[15] Oracle White Paper. Climbing to the olap summit with oracle warehouse builder 10gr2. Technical report, Oracle, January 2006.

[16] Sethu Meenakshisundaram Matt Carroll Denny Guang-Yeu Lee Sivakumar Harinath, Robert Zare. *Microsoft SQL Server Analysis Services 2008 with MDX*. Wrox, March 2009.

[17] David Hill S.Somasegar, Scott Guthrie. *Microsoft Application Architecture Guide (Patterns and practices), 2nd edition*. Microsoft Press, November 2009.

[18] Chris Tavares. Building web apps without web forms. `http://msdn.microsoft.com/en-us/magazine/cc337884.aspx`, March 2008.

[19] Elizabeth Vitt. Microsoft sql server 2005 analysis services performance guide. Technical report, Microsoft, February 2007.

[20] The Data Warehousing and Business Intelligence division @ Vivan. The evolution of etl-from hand-coded etl to tool-based etl. Technical report, Vivan Technologies, June 2007.

[21] III William E. Pearson. Optimizing microsoft sql server analysis services: Optimization tools: Usage-based optimization wizard by. `http://www.sql-server-performance.com/articles/biz/optimizing_usage_based_wizard_p1.aspx`, April 2004.

9 Appendix A - Use cases

9.1 Login use case

1. The systems prompts the user for a username and a password.
2. The user enters a username and a password.
3. The system validates that the username/password combination belongs to a valid user and that the user has accepted the appropriate licence agreement.
4. The system saves the timestamp of the successful login and loads the start page of the system.

Alternative path – Not valid username/password combination

3. If the username belongs to a user in the system, the system logs the failed login attempt with a time stamp. The user is then notified that the login attempt failed.

Alternative path – Licence agreement not yet accepted

3. The system displays the details of the license agreement.
4. The user accepts the license agreement.
5. The system marks that the user has accepted the licence agreement.
6. The use case continues from step 4 in the main flow.

9.2 Cost analysis use case

Precondition The user is logged in has access to the cost analysis module.

Main path

1. The user selects to analyze costs.

2. The system displays filtering options for period, compared period and organization. Only organizations and periods authorized to the user are displayed. By default, all organizations and the latest period are selected. If a period exactly one year before the latest period exists this is selected as compared period.

3. The user selects a period, one or more organizations and (optionally) a comparing period.

4. The system displays a summary of the invoice data matching the previous selections, grouped by the categories Fixed traffic, Mobile traffic and Fees. The following data is displayed:
   - Category name
   - Call length (tt:mm:ss) (*)
   - Number of calls (*)
   - Number of subscriptions
   - Amount (two decimals)
   - Difference between the amounts for the selected period and the compared period. (**)
   (*) For the category ŞFeesŤ no value are displayed in this field.
   (**) Visible only if a compared period is selected in step 3.

5. The user selects one of the categories ŞMobile trafficŤ or ŞFixed trafficŤ.

6. The system displays a summary of the invoice data matching the previous selections, grouped by call group. The following data is displayed:
   - Call group name
   - Call length (tt:mm:ss)
   - Number of calls
   - Number of subscriptions
   - Amount (two decimals)
   - Difference between the amounts for the selected period and the compared period. (*)
   (*) Visible only if a compared period is selected in step 3.

7. The user selects one of the call groups.

8. The system displays a summary of the invoice data matching the previous selections, grouped by call type. The following data is displayed:

- Call type name
- Call length (tt:mm:ss)
- Number of calls
- Number of subscriptions
- Amount (two decimals)
- Difference between the amounts for the selected period and the compared period. (*)
(*) Visible only if a compared period is selected in step 3.

9. The user selects one of the call types.

10. The system displays a list of subscriptions that has associated invoice data matching the previous selections. The following data is displayed:
- Subscription name
- Call length (tt:mm:ss)
- Number of calls
- Amount (two decimals)
- Difference between the amounts for the selected period and the compared period. (*)
(*) Visible only if a compared period is selected in step 3

    The list is sortable and the data can be grouped by the subscriptionsŠ organizational belonging.

11. The user selects one of the subscriptions.

12. The system displays information about the selected subscription, such as organizational belonging, installation address and products. The system also displays the invoice summary for the selected subscription where all costs for the selected period and Ű if selected Ű the compared period are displayed.

Alternative path – Not enough data

2. The system finds no organizations or periods authorized to the user. The system displays a notification to the user saying that there is not enough data in order to continue the analysis.

Alternative path – The user selects to analyze fees

5. The user selects the category ŤFeesŤ.

6. The system displays a summary of the invoice data matching the previous selections, grouped by the fee type. The following data is displayed:
   - Fee type name
   - Number of subscriptions
   - Amount (two decimals)
   - Difference between the amounts for the selected period and the compared period. (*)
   (*) Visible only if a compared period is selected in step 3.

7. The user selects one of the fee types.

8. The use case continues from step 10 in the main path.

Alternative path – Subscriptions are grouped by organizational belonging

11. The user chooses to group the subscription list on organizational belonging by selecting a desired organizational level, for example cost location.

12. The system displays a list of organizations that has associated invoice data matching the selections mad in step 1-10. The following data is displayed:
    - Organization name
    - Call length (tt:mm:ss)
    - Number of calls
    - Amount (two decimals)
    - Difference between the amounts for the selected period and the compared period. (*)
    (*) Visible only if a compared period is selected in step 3

13. The user selects an organization.

14. The system displays appropriate information about the selected organization.