Techniques for semi-automatic generation of data cubes from star-schemas

Mattias Hinnerson
Abstract

The aim of this thesis is to investigate techniques to better automate the process of generating data cubes from star- or snowflake schemas. The company Trimma builds cubes manually today, but we will investigate doing this more efficiently. We will select two basic approaches and implement them in Prototype A and Prototype B. Prototype A is a direct method that communicates directly with a database server. Prototype B is an indirect method that creates configuration files that can, later on, get loaded onto a database server. We evaluate the two prototypes over a star schema and a snowflake schema case provided by Trimma. The evaluation criteria include completeness, usability, documentation and support, maintainability, license costs, and development speed. Our evaluation indicates that Prototype A is generally outperforming Prototype B and that prototype A is arguably performing better than the manual method current employed by Trimma.
Acknowledgements

First and foremost I would like to say a big thank you to my two supervisors: Adam Holmström and Michael Minock, who have guided me through this master thesis with much thoughtful advice. I would also like to say a big thank you to the company Trimma that let me work at their office, lending me a desk to work at and a laptop to work with during this thesis. I would also like to give a thank you to Trimma’s employees, who helped me during this thesis in many different ways, for example by being survey participants in the evaluation.

Secondly, I would like to thank my parents, grandparents, sisters, and my sister’s son for giving me support during this thesis’ trials. Lastly, I would also like to thank all my friends that gave additional support.
Contents

1 Introduction 1
   1.1 Organization of this thesis 1
   1.2 About Trimma 2
   1.3 Thesis Focus 3
   1.4 How it could work in a (semi)-automated method 3
   1.5 Problem statement 3
   1.6 Restrictions 4

2 Background 5
   2.1 History of Business Intelligence 5
   2.2 Business Intelligence structure and parts 6
      2.2.1 OLTP (Online Transaction Processing) 6
      2.2.2 Data Warehouse 6
      2.2.3 ETL (Extract, Transform, Load) 7
      2.2.4 Structure of a data warehouse 7
      2.2.5 OLAP Cubes 8
      2.2.6 OLAP (Online Analytic Processing) 10
   2.3 Software that Trimma use: 10
   2.4 Available Tools for Cube Creation and management 11
      2.4.1 Microsoft API AMO 11
      2.4.2 BIML 11

3 Approach 13
   3.1 Example Schemas 13
      3.1.1 Case 1: A star schema with five dimension tables 14
      3.1.2 Case 2: A Snowflake Schema 14
   3.2 Prototype Architectual Design 14
   3.3 Evaluation Criteria 17
6.3 Future work ................................................................. 48
References ................................................................. 49

A Appendix ................................................................. 53
A.1 Implementation case 1: Star Schema ......................... 53
A.2 Implementation case 2: Snowflake Schema ................. 54
A.3 User Survey ............................................................ 55
1 Introduction

The world we live in gets more globalised every day, opening up for new business opportunities for companies to expand to other markets, but also creates a higher risk for competition [1, p. 31-32]. This has led to that companies have started to use computerised support for improved decision making. This phenomenon appeared already in the 70s, where it in 2006 became widespread under the name Business Intelligence (BI). BI provides easier access to data and gives better insight on the development of a business, leading to more well informed and better decisions [2, p. 19]. Constructing a BI system can be very challenging, expensive and require years of work. This because of all its complex parts that both should be secure (since the system might contain sensitive strategical data), and integrated into many other systems such as databases, legacy systems, etc. [1, p. 40,42].

This master thesis will focus on a particular part in the construction of new BI systems: the creation of Online Analytical Processing (OLAP) cubes. We will investigate various techniques that could speed up the creation of cubes. Two of these will be selected for prototype implementation and then be compared and analysed according to an evaluation criteria.

1.1 Organization of this thesis

Introduction gives an explanation of what this thesis is about, a description about the company Trimma, how they create cubes today, and a problem description with a problem statement.

Background gives an explanation of what this thesis is about, a description about the company Trimma, how they create cubes today, and a problem description with a problem statement.

Approach describes the approach of this thesis, telling how the evaluation will be performed to answer the thesis’ problem statement, and also what criteria that exist.

Implementation contains a description of the two selected techniques, where we describe syntax similarities and differences.

Evaluation describes the performed evaluations conducted on each method according to the list of criteria stated in chapter four. It also has an overview of the results at the end of the chapter.

Discussion and Conclusion contains a discussion, states the answer to the thesis’ problem statement, and also propose possible future work.
1.2 About Trimma

This master thesis is done in cooperation with Trimma. Trimma is a software company that was founded in Umeå in the year of 2003. Today they have several offices around Sweden, still having their head office in Umeå. Their focus is solutions for operation management and business intelligence that helps other companies to run their businesses more efficiently. One of the programs that they develop and advertise is the application named INSIGHT (INSIKT in Swedish), which is based on standard techniques from Microsoft. Many big companies use their software, for example, MAX hamburgers, LKAB, The Swedish Consumer Agency (Swedish: Konsumentverket), etc. Their software supports the whole process from setting up goals and planning, to making analysis and performance management. By using INSIGHT, the customers of Trimma can get statistics about their own business so that they can monitor their development and better plan for the future.

The tool Trimma uses today for building OLAP cubes is Microsoft Visual Studio. The content inside Trimma’s data warehouse look different and contain different types of data in every customer project, but they always follow a particular pattern. The process that happens when Trimma get a new customer can briefly be described as:

1. The customer decides to buy Trimma’s product INSIKT.
2. Trimma build a data warehouse, which is a database system with particular content with a model that describes the demands and needs of the customer in an efficient manner. The needs of the client are somewhat known to Trimma since the base often follows a particular pattern. For example, if Trimma get a new customer in an area they have had customers before, and the customer wants a system for financial tracking of their own business, then Trimma have a clue of how to build up the data warehouse. This includes star-/snowflake schemas (i.e. internal data warehouse structures), and indirectly the cubes, in such a way that the customer’s needs are mainly covered. However, if they get a new customer that has entirely different demands or in a field completely new to Trimma, then they would have to do more manual work.
3. Provided that the customer’s operational source systems are standardised and with a known structure by Trimma since earlier projects, the ETL-process (data extraction) and data warehouse with star schemas gets semi-automatically generated.
4. The result so far is a star schema with facts and dimensions that are built according to Trimma’s standard formats and conventions.
5. The next phase is to create one or several cubes from these star schema database. Today this work (defining cube structure) is done manually by consultants and requires hours of clicking, drag and drop and completing wizards in Microsoft Visual Studio.
6. When Trimma have created definitions of how the cube should look like and where data can be extracted from the star schemas, The definitions are loaded onto a Microsoft SQL server where the built-in cube engine gathers information and creates the cube(s) with desired aggregations.
7. The web application INSIKT gets connected to the created cube(s) and the customer’s data can now be analysed.

https://www.trimma.se/
1.3 Thesis Focus

A brief overview of the central parts of a BI system is shown in Figure 1. These will be described in more detail further into the report, but briefly, ETL is the process of fetching analytic data from customers. The output is stored in the Data Warehouse, as a collection of tables with relations formed like a star- or snowflake schema. This data can, later on, be transformed into data cubes. It is this, among others, that Trimma build. These parts need to be constructed for every new customer that Trimma get. This requires a lot of consulting services and time. Trimma has focused on simplifying the construction of star schemas and ETL function. However, still many manual instances of cubes from the star schemas need to be managed (see * in Figure 1).

![Diagram of ETL function and Data Warehouse](image)

**Figure 1:** The focus of this thesis will be the link between star/snowflake schemas and cubes.

1.4 How it could work in a (semi)-automated method

The semi-automatic generation would preferably be done in a way that when the prototype starts, you give it as input, a name to a database source and a fact table. It should then automatically be able to understand the schema structure from the fact table, set up definitions for where data should be collected, define the cube structure and lastly generate a cube upon this. Note though that it is up to the database engine to load and create the cube defined by the prototypes’ specified definitions and structure. It is also preferable if the method could be used as a plug-in inside Microsoft Visual Studio. If it can be automized in this way, then it will save Trimma many hours of manual work every time they get a new customer or whenever new cubes need to be built. If it works, then they would also be able to accept more customers since the startup process can be optimised and done faster.

1.5 Problem statement

The aim of this thesis is to find out what techniques exist that could more automatically generate the otherwise manual work of creating OLAP cubes. Then it is necessary to determine how well each of those perform regarding completeness, usability, and effectiveness. All this should lead to an answer for which tool is the preferable choice. All these problem-questions can get summarised in the following problem statement:

- Out of all available techniques for OLAP cube generation, can we determine which technique that is the best choice regarding completeness, good usability and high efficiency?
1.6 Restrictions

This thesis will not involve inventing an entirely new technique for autogenerating database cubes, but instead it will contain an evaluation of existing tools. We will not create a whole database engine for creating a cube, but rather we define where the engine should extract data, how it should be interpreted, and how the internal structure should be inside the cube. It is then up to the cube engine to act on our set up definitions and extract the actual data.

This thesis will focus only on the generation of Online Analytical Processing (OLAP) cubes. Hence, the evaluation is based only on that area and does not include, for example, *SQL Server Integration Services* (SSIS) [3]. How the evaluated techniques will perform is solely measured for the OLAP aspect, and the results will not show how they behave in other aspects and fields. Also, this thesis will only take into account that the cubes are created from scratch from a star/snowflake- schema. It does not measure how each technique performs with configuring and changing existing ones.

Lastly, this thesis will only take into account schemas which have one fact table inside it. We will not include additional fact tables within the same schema.
2 Background

2.1 History of Business Intelligence

The idea to analyse one’s surroundings for better business decisions appeared already in 1865 where Richard Millar Devens described the banker Sir Henry Furnese’s work as business intelligence (BI) [4]. Although computers did not exist, Furnese managed to, via manual methods, get deeper understandings of the issues (such as political ones) in his surroundings and used it to make better decisions [5]. People later on accused Furnese to be a corrupt financier [6], but the idea of BI was born.

In 1958, Hans Peter Luhn, who often is considered the father of BI [5], described the objective of business intelligence system as: ”The objective of the system is to supply suitable information to support specific activities carried out by individuals, groups, departments, divisions, or even larger units”[7, p. 2]. This area continued to be developed and in the 1970s, static two-dimensional reporting systems started to exist with no analytic capabilities [1, p. 33]. These were run on mainframe computers and were often referred to as black holes since they were good at collecting data, but retrieving data for analysis were much more challenging [1, p. 39]. This because of their complicated file structure, resulting in that a complex aggregated information retrieval took a long time to execute. These things changed in the late 70s where solutions came up to solve this problem, where the company Teradata created a database management system specifically for decision support[1, p. 65].

In the 1980s, Personal computers (PC) became widely available. This lead to increased digital storage which became more spread out than in mainframe computers. Resulting in the problem called Islands of data. New software was created to fix this, called distributed database management systems, which upon querying, fetched the data directly from files at the organisation’s PCs and put it in one place, transformed it and created an answer to a user’s question. However, this was not efficient [1, p. 65]. Instead, a change happened in the 90s where it went back to the ideas of the 70’s to copy data into a central location to later perform analysis on it. This technique resulted in the birth of Data warehouses [1, p. 66], which are a central part of BI today [8].
2.2 Business Intelligence structure and parts

When talking about Business Intelligence, there are several parts involved in the system. A general overview can be seen in Figure 2 below.

![Figure 2: A simplified example of a Business Intelligence system.](image)

2.2.1 OLTP (Online Transaction Processing)

Traditional databases belong here, which are accessible using queries for insertions, updates and deletions. These queries only touch a small part of the database [9, p. 1102]. OLTP databases handle the ongoing operations of the business, for example, a store purchase which should calculate a reduction in the store’s inventory. They are designed for efficient transaction processing but are not optimised to conduct analyses on [1, p. 39], this because they are not built for analysis and do not typically store historical data. They mostly only care about the current state of the business [10, p. 2].

2.2.2 Data Warehouse

A data warehouse is usually a copy of the original database where data may have been fetched from multiple OLTP sources, which gets stored in one single place. The storage process of data happens periodically and is usually static and not modified afterwards. A copy is needed since running a long transaction directly on the OLTP database would stall normal OLTP operations (such as recording sales) and would be very inefficient. It is, therefore typical that this copying of data happens over night to not disturb the daily operations of the business [11, p. 465]. One could say that data warehouses represent a coherent picture of the business conditions at a particular time (a time point), where all earlier time points also get saved and stored [1, p.40, 66]. The amount of stored data in the data warehouse is usually large and is restructured to support analytical work better. There are many business rules of how to use, transform, encode, calculate and store data in the data warehouse [12, p. 55]. The warehouse’s ETL function usually handles the storage process [9, p. 1103].
2.2.3 ETL (Extract, Transform, Load)

Plays a significant role in the data warehouse. In short, this is the process that moves data from external OLTP sources into the Data Warehouse. It consists of a work area, instantiated structures, and processes [10, p. 19]. The processes is often divided into three parts:

- **Extract**: Reads data from one or many sources, which potentially could be of many different kinds (OLTP databases, spreadsheets, external files, etc.) [12, p.54]. Data is extracted into the ETL system for further transformation [10, p. 19].

- **Transform**: Since extracted data can have different formats, we need to transform it into a form that is suitable for storage in the data warehouse. This stage can also include aggregation and cleansing of data [12, p.54]. Cleansing, according to Kimball and Ross [10, p. 19-20], means that it corrects misspellings, resolves domain conflicts, handles missing elements, or parse data into a standard format.

- **Load**: Load over the transformed data into the data warehouse’s dimensional models [10, p. 20].

2.2.4 Structure of a data warehouse

The logical representation of data in the data warehouse can be seen as a multidimensional model that is either built up like a star schema or snowflake schema, or an OLAP cube. The logical design for these is similar, but they differ in physical representation [10, p. 8]. Ralph Kimball and Margy Ross suggest in their book “the data warehouse toolkit” that, when constructing a data warehouse, one should first build star-/snowflake schemas hosted in a relational database, then build OLAP cubes upon these[10, p. 9]. It is common that a Data warehouse has multiple star- or snowflake schemas. A **Fact table** is in the center of the schema. Tables that are linked with a relation to the fact table is called **Dimension tables**.

The difference between a Star- and a Snowflake schema is that, in a snowflake schema, the dimension tables could have additional **Sub-tables** which describes the dimension with additional attributes. These attributes gives extra possibilities for filtering the result of a query. An example could be when sending a query, asking for the amount of sold menu items at a restaurant, by only showing data where the restaurant is located in a specific region. A graphical representation of a star schema and snowflake schema is shown below.

![Figure 3: A star schema (a) and a snowflake schema (b), containing a fact table (purple), dimension tables (dark green), and dimension sub-tables (light green).](image-url)
**Fact Tables:** Contains foreign key connections to the dimension tables, but also measures, which are dependent attributes that a user may want to ask the system [11, p. 467]. An example of a measure is amount, or sales.

**Dimension tables:** Contain tuples of attributes that explain an event [9, p.1108-1109]. These dimension tables can get huge, having up to 100 attributes in its dimension. Additional sub-tables may also exist to dimension tables (coloured light green in Figure 3), giving it additional attributes. Every dimension table has at least one primary key to which the fact tables’ foreign keys are joined to. These dimension tables create the constraints to queries via their attributes. If a user, for example, wants to see sales by brand, a brand must exist as an attribute in a dimension table [10, p. 13,15]. Hence, attributes in dimensions are used for filtering and grouping the facts. [10, p. 40]. Attributes can be combined into hierarchies.

**Hierarchies:** Describes the drill-down paths for users to see more detailed data [13]. Several hierarchies can exist in a dimension table. A hierarchy can be divided into levels with more or less detail, where the levels contains more or less precomputed, aggregated data. An example of two hierarchies for date are Year-Month-Day and Year-Week-Day [10, p. 48]. We then get that numbers are shown distributed as Years, Months/Weeks, or Days.

### 2.2.5 OLAP Cubes

Cubes are a multidimensional storage model where each axis in the cube is a dimension (category), which can be compared with other dimensions [11, p. 466]. OLAP cubes differ from star schemas in the way that it has special indexing techniques designed for dimensional data, and that it pre-calculates aggregations (or summaries) of data into the tables, which could give increased query performance [10, p. 8]. If we have a cube where the dimensions are Menu items, Restaurants, and Time, a point in that three-dimensional cube where each axis (dimension) intersects will be a specific menu at a particular restaurant at a given moment in time. This example is shown below.

![Figure 4: An example of a cube, with three dimensions: Restaurants, Menus, and Time](image-url)

With cubes, users get the opportunity to twist, turn, compare and analyse a large amount of data in many different ways. For further clarification of these concepts, consider the following example.

A snowflake schema for a restaurant is given in Figure 5. There we have one fact table that is built up using three dimensions tables: restaurant (coloured yellow), time (coloured blue), and menu (coloured green). These dimension tables are connected to the fact table via Foreign Keys (FK). The fact table also has an “Amount” measure. Note that the restaurant
dimension has three hierarchies, while time and menus only have one. Every dimension has at least one hierarchy which is a plain list of all tuples. Additional hierarchies as in the restaurant dimension enable that the user could, for example, see how much they have sold of a particular menu item at a specific restaurant on a specific date. One could also limit the shown results by only seeing restaurants of a certain size or from a particular region.

An example of a cube derived from this snowflake schema is shown in Figure 5. Here it exists entries for three restaurants, three menu items, and three dates. It also has extra rows, columns, and depth. One extra row exists called Total which contains the sum of each sold menus at all restaurants on each date. Example: on the date 2016-09-21, 2000 pizzas were sold in the Stockholm restaurant, 1000 in Umeå, and 3000 in London. Everything sums up to 6000 pizzas in the total row. When the user queries for the total amount of sold pizzas on that date, the value, which is already precomputed, only has to be fetched from one cell in the cube.

**Figure 5:** Schema and Cube example.
2.2.6 OLAP (Online Analytic Processing)

In short, OLAP is a method for end-users to analyse data [9, p. 1102]. There are several different variations of OLAP systems where each has pros and cons. The most common ones are MOLAP, ROLAP and HOLAP [1, p. 95-96].

**MOLAP:** Is a shortage for *Multidimensional Online Analytical Processing*. This method is built upon the usage of multidimensional data cubes for storage and analysis. These cubes contain both raw data and precomputed aggregated data [11, p. 467]. This aggregated data could, for example, be the sales for a specific menu item at all restaurants. The storage medium for cubes is optimised multidimensional arrays [1, p. 96].

**ROLAP:** Stands for *Relational Online Analytical Processing*. It has a design for analysis through a multidimensional model, but it does not require precomputed data such as aggregations. Instead, When the end-user has a question to the system, an SQL query is created to directly ask the relational model (i.e. the star-/snowflake schema) for an answer to that question. Using ROLAP gives that less storage of information is needed. The storage medium for ROLAP is relational databases [1, p. 95-96].

**HOLAP:** Stands for *Hybrid Online Analytical Processing* and it is a combination of MOLAP and ROLAP [1, p. 96].

2.3 Software that Trimma use:

The database structure that Trimma use today is Microsoft’s solution for OLAP cubes named *SQL Server Analysis Services* (SSAS) [14]. Currently, they build the cubes manually in Microsoft Visual Studio where the system has a MOLAP structure. When building, they declare the cube definitions, and where and how to extract data, the rest is handled by the Cube engine. Thus, the cube engine deals with how data is stored inside the cube. Possibilities exist, though, for example, to configure the cube engine to select if MOLAP or ROLAP should be used, if cubes should be saved on disk or RAM, etc. In this thesis, We will only work with cubes that get saved to disk.
2.4 Available Tools for Cube Creation and management

After a thorough search for available techniques for a more automatic creation of SSAS cubes, these are the tools that were found.

2.4.1 Microsoft API AMO

*Microsoft API Analysis Management Object* (AMO) is a complete library under the .NET Framework version 2.0 with classes that enables the user to create, modify, and delete Analysis Services objects such as cubes, dimensions, mining structures, and databases. The whole library is built on managing an instance of Microsoft Analysis Services [15].

According to Microsoft, their AMO is built for automating repetitive tasks that are usually performed on a weekly, monthly, or quarterly basis, where they even mention as an example to automate the creation of cubes [15]. Extra features that AMO have is a cleaning service, where data older than a specified time limit can be set to be removed, some security features such as user permissions, and automatic backup management.

The part of AMO that is used for SSAS cubes is the AMO OLAP Classes, which includes both the Cube class and the Dimension class. There are also possibilities to use the advanced classes, for example for creating alert-actions when the user browse parts of the cube, particular views of a single cube, and proactive hashing that helps with the balance between MOLAP and ROLAP data storage [16]. The advanced classes might not be interesting for this thesis, though, only the OLAP classes are of interest.

2.4.2 BIML

BIML stands for *Business Intelligence Markup Language* and is an XML dialect that handles many aspects around Business Intelligence Programming. The primary programming language used in BIML is XML, but code written in C# or VB can also be included into the BIML code [17]. BIML is created for easier being able to build data warehouses and Business Intelligence solutions. The aim, according to Scott Currie [18], is that this software should eliminate a lot of clicks and drag procedures, to more automate the generation of SQL scripts, SSIS-packages and SSAS cubes.

BIML works in the way that one writes XML based source code, often with the help of XML code editors. One example of such a code editor is the Varigence’s Mist IDE [19]. The BIML compiler should then be able to read the source files and output SQL scripts, SSIS files (DTSX, DTPRO), or SSAS files (cube, dim, dsv). These SSAS files should, later on, be able to be used with Microsoft SQL server [18].
3 Approach

Three ways of building database cubes were found. One is doing it manually, as Trimma does it today. Two other options are using API AMO or BIML. For this thesis, we will build two Prototypes, where the output of both will be stored on a Microsoft SQL Server 2014.

Prototype A: Will be built using Microsoft API AMO inside Microsoft Visual Studio IDE. This prototype will be more of a direct method where the prototype will communicate with the SQL server every time an object (i.e. data source, data source view, dimension and cube) is created. This also includes that the objects get processed, which means that the database engine fills the objects with data.

Prototype B: Will be constructed using BIML inside the Mist IDE, but Microsoft Visual Studio will still be needed to load the SSAS cube files onto the database server. This will be more of an indirect method where object configuration is created via BIML code. The BIML Compiler will output these as SSAS files, which then could be uploaded to the runtime SQL server via Visual Studio.

The prototypes will be constructed to work in the general case taking in only a data warehouse reference, a fact table name, and a reference to where the generated cube should be stored. They will then produce a cube upon this information. The prototypes will have to follow a few rules which we will describe in this chapter. We will also show what schemas that will be used as a reference when building the prototypes, together with a checklist specifying what one should expect from a good cube generation tool.

3.1 Example Schemas

When building the two prototypes, two different schemas will be used as a model. These schemas describe the included tables, with a fact table and the relations in between. We could see these as two cases, where one is simpler having a star schema structure, and the other is more advanced, having a snowflake schema structure. The advanced schema will enable the opportunity to create a cube with more information and more possibilities to group data into additional categories.

The prototypes should work for the general cases, not being hard coded for a particular schema relationship. Tables that should be considered included are those who has a foreign-key relationship from a parent table (ex. the fact table has a foreign key to a dimension table). Each dimension table (or sub-table) can have from one up to five level columns (L0,L1,L2,L3,L4). These describe data more or less summarised, where in this example, Level four is the most detailed and level zero the most summarised. If a table column has an additional DisplayName, this should be used for graphically display the content of a column. Every dimension will have at least two hierarchies: one where rows are just a plain list, and one grouped according to the dimension’s level structure.
3.1.1 Case 1: A star schema with five dimension tables

The first goal was to make so that the prototypes could be able to read and create cubes based upon data input of a star schema structure. A figure describing the star schema used during the implementation can be seen in the appendix, section A.1, where it had one fact table and five dimension tables.

3.1.2 Case 2: A Snowflake Schema

When the prototype could handle a star schema case, it got extended to also handle a snowflake schema case, where each dimension table could have additional sub-tables. The snowflake schema used for the implementation can be seen in the appendix, section A.2. It had one fact table and five dimension tables, where two of those dimension tables had two additional sub-tables connected to them.

3.2 Prototype Architectural Design

Both of the prototypes will initially use the same C# code for reading in the internal structure, which is later on needed for building the cube. This internal structure represents the involved database diagrams with relations, from the input database. The states that both prototypes need to go through is:

1. Run the program by giving the input server and database names, the fact table name to build from, and the name of the server and database to store the final cube in.
2. Call the Schema reader class to create an internal structure of necessary schemas and relations.
3. Prefetch all necessary schemas that have a connection to the given fact table (including the fact table itself).
4. Store this together with name references in an internal structure.
5. Return the structure to the main program.
6. Create a cube upon the internal structure, either using the method of prototype A or B.
Prototype A:

(A) The Program will create the data source, data source view, all dimensions, and the cube (with one partition) via the AMO code.

(B) Between every creation of an object mentioned above, the code will deploy and process each item into the output database and receive a response. The order of deployment should follow the order described above.

![Diagram of Prototype A](image)

**Figure 6:** Architectural Overview of prototype A

Prototype B:

Since BIML is a more indirect way of creating a cube, the structure of procedures will be like this instead:

(i) From the internal structure we can build up the BIML code.

(ii) This code will be built inside the Mist IDE, where the BIML Compiler will compile it into output files.

(iii) These output files will contain SSAS cube definitions, but also a Microsoft Visual Studio project file.

(iv) This file can be opened inside Microsoft Visual Studio, where the cube definitions will be read as well.

(v) This can then be deployed and processed onto the output database.
Figure 7: Architectural Overview of prototype B
3.3 Evaluation Criteria

This subsection describes what expectations one should have on a tool for generating SSAS cubes. They are described below together with an explanation on how it will be used later on in the evaluation chapter to evaluate AMO and BIML.

1. Completeness: It is important that the tool is complete enough so that the generated cubes look like the ones that would otherwise have been manually built.

   This evaluation is measured in two ways. First, a verification will be done so that each of the two prototypes can handle both schema cases stated in section 3.1. Secondly, we will compare the created cubes by each tool with a manually built cube using Trimma’s web-client. A couple of test reports will be constructed in their client where the internal values of the reports will be compared with the values inside reports made from a manually built cube. The values inside the reports should look the same between all three cube versions.

2. Usability: It is important that API AMO and BIML code is easy to read and understand, minimising the risk of creating bugs when coding in each code language, but also produce results more quickly. A survey will be made with the employees of Trimma to evaluate this, where they will read some particular code sections done both in AMO and BIML, and answer a few questions about each tool.

3. Documentation and Support: In cases when the understandability is not enough to perform a particular task in AMO or BIML, it is important that the tool is well documented, both in written texts and in web content such as blog forums, etc.

   For measuring this, we will count the number of forum threads about cubes for each technique on their main forum websites. Also, to measure the support, we will post a problem on each of the technique’s forums and send a mail to their support, measuring the time it takes to get an answer to the problem that helps getting the problem solved. The questioned problem was how one should do to create a relationship between dimension tables and sub-dimension tables for the data source view (which is the collection of tables and relationships from the input data warehouse).

4. Maintainability: The tool should be easy to maintain, be modularized and easy to extend.

   Heitlager, et-al. describes in their article for measuring Maintainability [20, p. 31-34] it can be measured in several different ways, where they, for example, mention the number of code lines (without comments and blank spaces), and Cyclomatic complexity per unit. The latter one was in their article done as a percentage risk calculation. We will use these two measure techniques, but the cyclomatic complexity’s “raw” number will be used instead, for comparison between the AMO and BIML prototype. A lower Cyclomatic Complexity number is desirable, which leads to easier maintainability and a lower risk of coding errors [21].
5. **Development speed:** It is important that the tool should be easy to understand and be expressive so that one can build much code quickly with it.

Specific implementation criteria will be created to measure this, describing when a feature for the prototype is done. When we have completed a feature, we will record how much time that was required to reach to that point. This measure will give an indication of how steep the learning curve is for each tool [22]. One should though be aware of that the measurements could be a bit biased since the development of the first prototype might help when developing the second prototype.

6. **Saved Developer Time:** How much time was approximately saved with using a tool for semi-auto-generating cubes than building them manually?

To get a measure of this, we will measure the time it took to create a program that generates a cube in AMO and BIML, versus the average time it takes for an employee at Trimma to manually build a cube in Microsoft Visual Studio. The employees at Trimma that participates in this measurement should have earlier experience of creating cubes manually. This will give a result that might say that it took \( x \) minutes to make a program for semi-automatic SSAS cube creation, and it takes \( n \) minutes to create a cube manually every time in average.

7. **License costs:** A small but still existing factor with the choice of technique is if there are any license costs to use the tool. If a license cost exists, how much is it?
4 Implementation

This chapter is about similarities and differences between AMO and BIML. First, general observations are presented, followed by the relevant, significant parts when building a cube. We will present these in the order that they were implemented when the two prototypes were built. Note though that the code samples only show the things that worked for both AMO and BIML. The inspiration for this comes from the implementation of the two prototypes, but also from Microsoft’s AMO articles [23] [24], and Weissman’s BIML SSAS article [25], which has been an inspiration source during the implementation as well.

4.1 General

The general similarities and differences between BIML and API AMO are:

4.1.1 Programming language

- **AMO:** is a library built on C# together with the .NET framework version 2.0 [15]. It is an object-oriented language with a hierarchy of classes and methods. You define an instance of a class, then calls it’s internal methods to add, delete, or change metadata inside it. Some classes can be instantiated at any time, and some need to be instantiated before other classes (e.g. a Dimension need to be instantiated before an attribute can be added to it) [15]. A code example for creating a dimension attribute is given below.

```csharp
DimensionAttribute attribMonth = dateDim.Attributes.Add("Month ID");
attribMonth.Type = AttributeType.Months;
attribMonth.KeyColumns.Add(....);
```

- **BIML:** is built on XML, using tags with metadata to define the cube structure. The order of which the tags appear could be important since these later on will be interpreted by the BIML compiler and converted to SSAS files. Rules exist so that some tags only exist within other tags. An example of creating a dimension attribute in BIML looks like this:

```xml
<Attribute Name="Month ID" AttributeType="Months">
  <KeyColumns> .. </KeyColumns>
</Attribute>
```
BIML have a feature of including C# or VB code inside BIML code [26]. This enables one to avoid repeating code sections. This code can be inserted in between the symbols:

<# ... #>

If we had several attributes in a dimension, the code could look like this:

```csharp
<# foreach (Attribute attrib in attribList){ #>
  <Attribute Name="<#=attrib.name()#>" AttributeType="Months">
    <KeyColumns> .. </KeyColumns>
  </Attribute>
<# } #>
```

### 4.1.2 Input Handling

- **AMO**: In C# (that is used in AMO) it is possible to build a program that can take input parameters when it starts to run. It is also possible to get data from other sources, using existing C# libraries. One example tested in the AMO prototype was Microsoft’s SQL Server Management Objects (SMO) [27], which was used for reading in an internal representation of the required database tables with relations.

- **BIML**: Since BIML can include C# and VB code, it makes it possible to include code libraries written in those languages. This is a good feature since you can reuse already written code. For example, this made it possible to reuse the SMO code for reading in the internal representation which we used in the AMO prototype. BIML can also take input from other sources, for example relational databases[28]. However, it seems like BIML is unable to take program start parameters. It is possible to send parameters between BIML code files, even read from separate files, but running the main BIML script with input parameters seems not to be supported (like the C#’s "Main(string[] args)"). Hence, in BIML one can include C# or VB code, but it does not seem to support making it the other way around, i.e. including BIML code inside existing C# or VB code.

### 4.1.3 Output Handling

- **AMO**: Communicates directly with the database, where the cube and its parts will be stored. It also has the feature to make the OLAP cube engine to process the object (dimension, cube), i.e. making the OLAP engine to extract data from the data warehouse to fill the object with data.

- **BIML**: is a more indirect method. You declare the cube structure (with internal properties) in BIML code. Then the BIML compiler in the Mist IDE builds SSAS output files from the code, including a visual studio project file. From this visual studio project file, you can inspect the result, deploy the cube onto a database and process it (i.e. filling the cube with values). This process of deploying the cube was done manually through out the thesis.
4.2 Creating a data source view

A data source view represents selected objects (such as tables with relations) from an input data source (in our case, the data warehouse). These are used when creating dimensions, cubes, and mining structures [29]. When creating a SSAS cube using either AMO or BIML, a data source with a data source view is needed. The ways that AMO and BIML do this is a bit different compared to each other.

- **AMO**: Before the creation of dimensions and cubes, the data source view with tables and relations in-between must exist in the output database where the cube, later on, will be stored. Microsoft has a basic tutorial for this on their website [23], which got upgraded in this thesis’ AMO prototype to work in a more general case. The order of implementation (in a somewhat pseudo-code form) of how it got implemented is presented below. Note that the visible code is in a brief form. Also, note that before the data source with view is created, the prototype has created an internal structure of tables, etc, needed for the cube.

```plaintext
DataSource <= Create dataSource(Name)
DataSource.ConnectionString <= Input Connection String
DataSource.Update
DataSourceView <= Create dataSourceView(Name)
DataSourceView.DataSourceID <= DataSource

//Add Dimension tables
inConn <= open connection to input database
foreach(InternalDimTable in internalDimTableList){
    Table <= get table ref. from input DB {inConn}, name = InternalDimTable
    DataSourceView <= Add Table to data source view
}

//Add fact table in the same manner as above

//Add Relations Between tables
foreach(InternalDimTable in internalDimTableList){
    fkeyList <= add all fkey-columns from parent table
    pkeyList <= add all pkey-columns from child table
    DataRelation <= new DataRelation(fromTable,fkeyList,toTable,pkeyList)
    DataSourceView <= add DataRelation
}
DataSourceView.Update
```

After this point it is possible to start generating dimensions and later on, cubes.
• **BIML:** Here, we declare the Data Source and Data Source View at the beginning of the BIML code, but relationships between the fact table and dimension tables get declared later on, first when the fact table is created. Each column in the fact table refers to the primary keys of earlier created tables. Below is an abbreviated BIML code, describing how the data source with data source view got set up in this thesis’ BIML prototype. Also, note how all tables (with internal dimension declarations) in BIML gets declared before the fact table.

```xml
<Biml xmlns="http://schemas.varigence.com/biml.xsd">
    //Read in internal structure of tables, relations, etc.
    <Connections>
        <AnalysisServicesConnection Name="output" Server="..." ConnectionString=".." />
        <OleDbConnection Name="input" ConnectionString="..." />
    </Connections>
    <Databases>
        <Database Name="myDW" ConnectionName="input" />
    </Databases>
    <Schemas>
        <Schema Name="dw" DatabaseName="myDW" />
    </Schemas>

    <Tables>
        //Create all tables here
        <#foreach(dimensionTable in InternalDimList) {#>
            <Table Name="dim_Name" SchemaName="myDW.dw">
                <Columns> .. </Columns>
                <Keys> .. </Keys>
                <AnalysisMetadata>
                    //Create dimension with metadata here
                </AnalysisMetadata>
            </Table>
        <# } #>

        //Add fact table.
        <Table Name="FactTableName" SchemaName="myDW.dw">
            <Columns>
                <#foreach(dimensionTable in InternalDimList) {#>
                    //It is here that the relations between fact table and dimension tables are set. Example:
                    <TableReference Name="KstID" TableName="myDW.dw.dim_Kst" />
                <# } #>
            </Columns>
            <AnalysisMetadata>
                //Add measures
            </AnalysisMetadata>
        </Table>
    </Tables>
    //create cube and create analysis services project
</Biml>
```
4.3 Creating a dimension

The generation of dimensions is needed for later on being able to build a cube upon these. During the development of each prototype, the goal was to make the code general so that it could handle the creation of multiple dimensions with varying content (attributes, hierarchies, etc.). Due to the different characteristics of BIML and AMO, the code has similarities but also differences. The code presented here is very brief. This to better point out the key similarities and differences. This subsection will be divided into three pieces to better be able to compare AMO with BIML.

4.3.1 Step 1: New empty dimension

The first step is to create the actual dimension, where it initially will be without any internal content. A more pseudo-code form of how this was done in AMO and BIML is presented below.

- **AMO:**

  ```csharp
  AddGeneralDimension(Database db, InternalDim oneDim)
  {          
    db.RemoveOldDim(oneDim.Name)           
    dim = new Dimension(oneDim.name)       
    db.Dimensions = dim                   
    db.ErrorConfiguration.KeyDuplicate = ReportAndStop
    ...
  }
  ```

- **BIML:**

  ```xml
  <Dimension Name="#=dim.name()#" ... >
    <ErrorConfiguration KeyDuplicate="ReportAndStop" ... >
    </ErrorConfiguration>
    <Attributes>
      ...
    </Attributes>
    ...
  </Dimension>
  ```

Here we can see that the similarities are that you call methods/constructors in AMO to set configurations for a data object, but in BIML you instead set configurations in the tag header. An example above is setting the dimension name.

Note that if one wants to start with an empty, clear database, the dimension needs to be dropped in the database when using AMO before the new dimension is created. BIML handles this by assuming that it should overwrite the dimension. Also, notice the small difference for when error configurations are set for the dimension, wherein AMO it is one method call for every configuration, but in BIML it is one parameter in the ErrorConfiguration-tag for every configuration.
4.3.2 Step 2: Add dimension key attribute

Once we have created the dimension, it is time to fill it with metadata. The first thing needed is to add an attribute for the key. Below it is shown how it can be done in AMO and BIML. As said before, this is a brief view, not showing all parts of the code.

- **AMO:**

```plaintext
srcTable <= db.dataSourceView.get(oneDim.name)
//add key attribute
dim.Attributes <= keyAttrib
keyAttrib.NameColumn <= DataItem(srcTable.name, srcTable.displayName, type)
for each (Pkey <= internalDimTable.primaryKeys){
    keyCol <= srcTable.column(pKey.name)
    keyAttrib.KeyColumns <= DataItem(srcTable.name, keyCol, type)
}
```

- **BIML:**

```xml
//Adding key attribute
<Attribute Usage="Key" Name="<#=keyAttribName#>">
    <KeyColumns>
        <#foreach (Pkey <= internalDimTable.primaryKeys){#>
            <KeyColumn ColumnName="<#=Pkey#>" />
        <#}#>
    </KeyColumns>
    <NameColumn ColumnName="<#=dim.displayName()#>">
</Attribute>
```

In both cases above, we create an attribute with the name keyAttribName. Like mentioned earlier, when you set configurations via methods in AMO, it is often set in BIML via parameters in the header tag. This can be seen above where the attribute’s usage is set to be key.

The key attribute has to be linked to the corresponding columns in the dimension’s source table. This is done in AMO by finding a reference to that column and adding it to the attribute, which is done for every column in the source table that belongs to the key attribute. In BIML, you add a keyColumn tag for every included column belonging to the attribute. Here the column name (i.e. Pkey in the example above) has to be the same as the name of the column in the source table. Also, a NameColumn is included in both examples above, which is used for graphically describing the attribute.
4.3.3 Step 3: Add dimension hierarchy level attributes

When the dimension exists with an internal key attribute, it is time to fill it with additional content. In this thesis’ prototypes, the following part was implemented to go through all predefined hierarchies and levels, creating two attributes for each level (one for describing how data should be sorted in that level, and one for containing the actual level data). These attributes get added as levels to a hierarchy, where relationships between them are set. A brief version for AMO and BIML is shown below (leaving out repeated parts with dots).

- **AMO:**

```csharp
//Let oneDim be the internal representation of a dimension table
//add attributes for all hierarchies and hierarchy levels
foreach(hierarchyName <= oneDim.Hierarchies){
    hierarchy <= new hierarchy(hierarchyName)
    dim.Hierarchies <= hierarchy
    foreach(levelName <= oneDim.Hierarchylevels){
        //add sort attribute the same way as described earlier.
        sortAttrib <= new Attribute(levelName + "_sort")
        dim.Attributes <= sortAttrib
        sortAttrib.Type <= FormattingOrder
        ...

        //add attribute for level, the same way as described earlier.
        lvlAttrib <= new Attribute(levelName)
        lvlAttrib.Usage <= Regular
        lvlAttrib.OrderBy <= AttributeKey
        lvlAttrib.OrderByAttributeID <= sortAttrib
        ...

        //add relationship between level attributes
        if(level != lowestLvl){
            lvlAttrib.Relationships <= Previous_attrib
        }
        //add relationship to sort attribute
        lvlAttrib.Relationships <= sortAttrib

        //adding level to hierarchy
        Level <= new Level(levelName)
        level.SourceAttribute <= lvlAttrib
        hierarchy.levels <= level

        Previous_attrib <= lvlAttrib
    }
}
}
dim.Update
```
• BIML:

```xml
<Attributes>
    <Attribute>
      // declaring key attribute here here...
    </Attribute>

    <#foreach(hierarchyName <= dim.Hierarchies){#>
        <#foreach(levelName <= dim.Hierarchylevels){#>
            <Attribute Name="<#=levelName + "_Sort"#>"
                AttributeType="FormattingOrder">
                ...
            </Attribute>

            <Attribute Name="<#=levelName#>"
                OrderBy="AttributeKey"
                OrderByAttributeName="<#=levelName + "_Sort"#>">
                ...
            </Attribute>
        </#foreach>
    </#foreach>
</Attributes>

<Relationships>
    <#foreach(hierarchyName <= dim.Hierarchies){#>
        <#foreach(levelName <= dim.Hierarchylevels){#>
            <#attrChild <= level - 1
                attrParent <= level #>
            <#if(level != lowestLvl){#>
                // Adding relationship between level attributes
                <Relationship Name="<#=attrParent#>"
                    ParentAttributeName="<#=attrChild#>"
                    ChildAttributeName="<#=attrParent#>"
                    Type="Rigid"/>
            </#if>

            // Adding relationship to sort attribute
            <Relationship Name="<#=levelName + "_Sort"#>"
                ParentAttributeName="<#=attrChild#>"
                ChildAttributeName="<#=levelName + "_Sort"#>"
                Type="Rigid"/>
        </#foreach>
    </#foreach>
</Relationships>

// Adding level attributes to hierarchy
<AttributeHierarchies>
    <#foreach(hierarchyName <= dim.Hierarchies){#>
        <Hierarchy Name="<#=hierarchyName#>">
            <Levels>
                <#foreach(levelName <= dim.Hierarchylevels){#>
                    <Level Name="<#=levelName#>"
                        AttributeName="<#=levelName#>"/>
                </#foreach>
            </Levels>
        </Hierarchy>
    </#foreach>
</AttributeHierarchies>
```
After this point, we have a dimension created in AMO or BIML.

As said before, there are similarities in the sense that when AMO uses methods to set configurations, BIML often uses parameters in the header tags. Here it can be seen for example when we set the attribute usage and sorting order for the level attribute. The sorting order in both AMO and BIML support an ordering defined by a separate sort attribute, where that attribute is created before the level attribute (as implemented above). This enables one to create a specific column describing the ordering of level values already in the data warehouse tables. Also, the way of connecting each attribute in relationships are similar, where each level attribute gets connected in descending order, and also gets connected to its sort attribute.

The major difference one can see is that there are more loops required in the BIML implementation compared to AMO. The creation of hierarchies, levels, and attributes could be placed inside the same loop for AMO, but in BIML this could not be done. This is because BIML declares its attributes, relationships, and attributeHierarchies individually, separated by tags (example, attribute gets declared inside the ”Attributes”-tag, then relationships are set in the ”Relationships”-tag).

### 4.4 Creating a cube

When we have declared the dimensions with internal attributes, hierarchies, and levels, we can build a cube upon these. The way that AMO and BIML do this is quite similar. How this got implemented in the two prototypes is shown on the next two pages, in a brief, more pseudo-like form.
**AMO:**

```plaintext
GenerateCube(db, dataSourceName, cubeName, cubeParts)
    cube <= new Cube(cubeName)
    db.Cubes <= cube
    cube.Source <= dataSourceName
    cube.StorageMode <= StorageMode.Molap

    //add dimensions to cube
    foreach(internalDim <= cubeParts.dimensions){
        cube.Dimensions <= db.Dimensions(internalDim.Name)
    }

    //Create measure group
    measureGroup <= new Measuregroup("Result")
    cube.MeasureGroups <= measureGroup
    measureGroup.StorageMode <= StorageMode.Molap

    //Add each measure to group
    foreach(measureName <= cubeParts.Measures){
        measure <= new Measure(measureName)
        measureGroup.Measures <= measure
        measure.Source <= DataItem(FactTable, measureName, type)
    }

    //Add mapping between cube and dimensions
    foreach(internalDim <= cubeParts.dimensions){
        cubeDimRef <= cube.Dimensions.get(internalDim.name)
        regMGDim <= new RegularMeasureGroupDimension(cubeDimRef)
        measureGroup.Dimensions <= regMGDim

        //First, add dimension key column
        measGroupAttrib <= new MeasureGroupAttribute(ref to dim. key column)
        regMGDim.Attributes <= measGroupAttrib

        //Then add surrogate key in fact table
        foreach (factTableFkey <= factTable.ForeignKeys){
            mgAttrib.KeyColumns <= DataItem(FactTable, factTableFkey, type)
        }
    }

    //Create partition objects
    part <= new Partition("Default");
    part.StorageMode <= StorageMode.Molap;
    part.Source <= tablebind to fact table

    //Update cube
    Cube.Update
}
```
Here there are some similar patterns of how the cube gets created, but still, differences exist. In AMO and BIML, we first create a cube and add the earlier created dimensions to it. In the example above for BIML, the measureGroup was created before adding the cubeDimensions, but it could as well have been done the other way around.

A measure group is also created in both AMO and BIML. The measure group contains the measures that existed in the fact table (in the prototype implementation it was amount and number). In both versions there are similarities. The measure group (with measures) gets created, and a declaration is done for what dimensions that are included in that measure group, and how the relation is between each included dimension and the fact table. Also, a partition is made for the cube, which is given the name "Default".

After this, we have created the Cube. However, in BIML you have left to create an analysis services project and add all dimensions and cube to it. This code sample is shown on the next page. If we build the code using the Mist IDE, the BIML compiler will create SSAS cube output files from this, together with a visual studio project file, which could be used to deploy the cube and process it (fill it with data).
BIML - Create Cube project:

```xml
<CubeProject Name="BimlCubeProj">
  <Connections>
    <Connection ConnectionName="outConnName"></Connection>
  </Connections>
  <Dimensions>
    <#foreach(DimensionPart oneDim in theCubeParts.getAllDimensions()){#>
      <#dimName = "BIML_DB.dw." + oneDim.getDimName() + "." + oneDim.getDimName().Substring(4);#>
      <Dimension DimensionName="<#=dimName#"></Dimension>
    <#}#>
  </Dimensions>
  <Cubes>
    <Cube CubeName="<#=cubeName#"></Cube>
  </Cubes>
</CubeProject>
```
5 Evaluation

The evaluation of the two prototypes is performed according to the criteria of Chapter 3.3.

5.1 Completeness

To be able to measure completeness in this thesis, we will evaluate the two built prototypes in two ways. First, we assess how well the two prototypes performed compared to the two cases stated in chapter 3.1 (i.e. a star schema, and a snowflake schema). Secondly, we will test how complete each cube is that is generated by each prototype, compared to a manually built cube.

5.1.1 Completeness compared to cases declared in chapter 4

During the development of the two prototypes, the first aim was that they should be able to handle case one, which was reading in one fact table with $n$ number of dimension tables connected to it. It should be able to generate dimensions and finally a cube upon these. When we achieved that case for each prototype, it got extended with the aim also to be able to include $m$ number of sub-tables to each dimension table, as described as case number two in chapter 3.1. How well these performed is described below in Table 1.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>API AMO:</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BIML:</td>
<td>Yes</td>
<td>Some uncertainty exist, according to the implementation test.</td>
</tr>
</tbody>
</table>

At the time of the implementation, it was discovered that it was possible to implement so that the AMO prototype could handle both cases. This has been tested with several different database schemas with different structures, and it behaves as expected compared to how it would behave for manually build cubes.
According to the implementation of the BIML prototype, it was successfully possible to handle the case of checkpoint one, with one fact table and \( n \) dimension tables. However, some problems appeared when trying to extend it to also include \( m \) sub-tables to dimension tables and make them hierarchies. The problem exists in adding all tables and their relations to the data source view. It was possible to insert all the tables in the data source view, but adding relations to sub-tables from the dimension tables were problematic. The BIML compiler gives no error when it is performed as described in the code snippet below, but when opening the generated SSAS files in Microsoft Visual Studio, the Data source view was broken.

The same way to how the relations were added to the fact table and the dimension table, was tested between the dimension tables and dimension sub-tables. The code tested looked briefly as follows, abstracting away loops, showing only the case where one sub-table exist to one dimension table:

```xml
//First the sub-table was added
<Table Name="SubTableName" SchemaName="myDW.dw">
  <Columns>
    ...
  </Columns>
  <Keys>
    <PrimaryKey Name="PK_keyID">
      ...
    </PrimaryKey>
  </Keys>
</Table>

//Then the dimension table was added
<Table Name="dimName" SchemaName="myDW.dw">
  <Columns>
    //Reference to sub-table created above
    <TableReference Name="<#=FKeyColumnInDimTable#>"
      TableName="<#=myDW.dw.SubTableName#>" />
    ...
  </Columns>
  <Keys>
    <PrimaryKey Name="<#=PK_DimKeyID#>">
      ...
    </PrimaryKey>
  </Keys>
  ...
</Table>
```

This way of creating a relation worked between a fact table and dimension tables, but not between dimension tables and sub-dimension tables. We did not find an answer to how this should be done. A thorough search on the internet, together with posting the problem on Varigence’s forum and emailing their support about this, was done to find an answer to how this could be solved.
Another BIML tag that was tested was the `MultipleColumnTableReference`, for referencing multiple rows in a key. It looks as follows:

```xml
<MultipleColumnTableReference Name="FKDimCol_1"
   ForeignColumnName="subTableCol_1"
   MultipleColumnTableReferenceGroupName="keyGroup1" />

<MultipleColumnTableReference Name="FKDimCol_2"
   ForeignColumnName="subTableCol_2"
   MultipleColumnTableReferenceGroupName="keyGroup1" />
```

But this did not give a good result either for our problem case.

### 5.1.2 Completeness compared to manually built cubes

To evaluate the completeness of cubes created by each prototype, compared to manually built cubes, Trimma’s Web-client INSIGHT will be used to create four reports that use different parts of the cubes. We will also use four key reference reports which have been set up from a manually built cube. Trimma has assured that these reports are correct and contain accurate numbers. The star schema structure utilised for this evaluation is a different one compared to the ones shown in chapter 3.1. For privacy purposes, this can not be shown in this report, but the general structure of tables looks as follows:

\[ \text{Figure 8: The schema used when creating the evaluation reports for each generated cube.} \]

The schema has five dimension tables, where one have a foreign key relationship that included two columns (a and b in the figure). Also, two of the dimensions had sub-tables with key relationships as shown in the Figure above.
For this evaluation, the same reports will be constructed as the key reference reports but based on the cubes made in the AMO, and BIML. These reports with its internal numbers will be compared with the numbers of the key reference reports. If the numbers inside all the reports are the same, then AMO and BIML can be concluded to be complete in this case. For minimising the possible error from evaluating the reports manually, the reports will be built in the INSIGHT client, downloaded as Excel files, and make Microsoft Excel check for differences between the Excel sheets.

The result from the completeness evaluation for each report is shown below in Table 2 below. Note that the fourth report requires that the cube contains sub-tables as well. Hence that report could not be tested for BIML.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Report 1</th>
<th>Report 2</th>
<th>Report 3</th>
<th>Report 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO:</td>
<td>Correct</td>
<td>Correct</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>BIML:</td>
<td>Correct</td>
<td>Unknown</td>
<td>Correct</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Here we can see that AMO could create all of the four reports and the internal numbers were correct compared to the manually built reports. Some uncertainty exists though with BIML. We knew since before that during the implementation part of this thesis, we were unable to make the BIML prototype to include sub-tables. Hence, that is why report four was not possible to create for BIML.

However, Report two was not possible to be built with the BIML prototype either. This is because that report used a dimension with a key relationship that included two columns (a and b in Figure 8). With the BIML prototype, we were not able to create the key relationship between the fact table and dimension tables for this case. The things that were tested to solve this were:

- Used the BIML tag:

  `<TableReference Name="<#=FKeyColumnInDimTable#>" TableName="<#=myDW.dw.SubTableName#>" />`

  This gave an error saying that the TableReference tag is for key relationships with only one column, and that multipleColumnTableReference should be used instead.

- Changed to use the MultipleColumnTableReference-tag everywhere where a foreign key had several columns to it. Otherwise, the TableReference tag was used. This lead to an error, saying that a column name exists at several places in the BIML code. It gave this since column a (see Figure 8) is a part of two separate key relationships and therefore exist as two tags in the BIML code.
• Each column that belongs to several key relationships was given a unique name. This compiled in the Mist IDE but gave too many columns in the fact table, and the created cube gave errors. We want to strive to have one column that can belong to several key relationships.

• Tried to make one MultipleColumnTableReference tag for each column in the fact table, with several internal tags stating each relationship. This failed since no possible tags were found that fits inside the MultipleColumnTableReference tag.

• Make all column relationships to be MultipleColumnTableReferences. This also failed, giving the same error as before that a column name is repeated in the code.

Although much effort was made to try to solve this for the thesis’ BIML prototype, including searching for help from external sources on the internet, no answer was found.

5.2 Usability

To be able to measure the readability and understandability of AMO and BIML code, a survey was performed on the employees of Trimma. The study had four code examples: two for AMO and two for BIML. The examples were made as general as possible not to be code specific to what was done in this thesis’ prototypes. Those performed the similar action, but in their code language. The actions were: adding a key attribute and adding a hierarchy with levels. After each code sample, a few questions existed with the aim of measuring the readability and understandability of each sample.

Among the staff that answered this survey, nine out of fourteen had previous experience of database cubes, dimensions, etc. This experience is from building cubes manually in Visual Studio and not via AMO or BIML. The numbers presented in the following diagrams includes all staff members that answered the questions. In the survey, the order of presentation in the first part was: first AMO, then BIML. In the second part, We switched so that the BIML code sample came first, before AMO.
Amount of persons participating in the survey: 14.

5.2.1 Understandability

Two questions existed to measure the understandability of AMO and BIML. The participants were asked to look at each code example, and answer if it was easy to understand, and also describe what the code sample was all about. The question and results of these are described below. The numbers inside the parenthesis describe how many survey participants that mentioned that they had prior knowledge of dimensions, cubes, etc. (this knowledge though is from creating them manually). A deeper description showing how the survey looked like can be seen in the appendix.

Was it easy to understand each code sample?

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Yes</th>
<th>No</th>
<th>Partly</th>
<th>No Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO 1</td>
<td>6 (5)</td>
<td>7 (4)</td>
<td>1 (0)</td>
<td>-</td>
</tr>
<tr>
<td>BIML 1</td>
<td>13 (8)</td>
<td>1 (1)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AMO 2</td>
<td>11 (8)</td>
<td>2 (1)</td>
<td>-</td>
<td>1 (0)</td>
</tr>
<tr>
<td>BIML 2</td>
<td>5 (3)</td>
<td>9 (6)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Describe the purpose of the code.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Correct Ans</th>
<th>Partly Correct Ans</th>
<th>Wrong Ans</th>
<th>Don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO 1</td>
<td>8 (6)</td>
<td>1 (1)</td>
<td>2 (1)</td>
<td>3 (1)</td>
</tr>
<tr>
<td>BIML 1</td>
<td>9 (7)</td>
<td>4 (1)</td>
<td>1 (1)</td>
<td>-</td>
</tr>
<tr>
<td>AMO 2</td>
<td>11 (7)</td>
<td>0</td>
<td>2 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>BIML 2</td>
<td>10 (6)</td>
<td>2 (2)</td>
<td>1 (1)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

By inspecting the two tables above, we can see that in the first case (number one above), BIML was easier to understand, and also more participants answered correctly or partly correctly when questioned about its purpose. More people did not understand the AMO code sample in this first case compared to BIML, and also answered wrong, or do not know, when asked about its purpose. However, in the second case, the survey participants responded that AMO was easier to understand and many thought that BIML was more complicated to read. In this second case, BIML had included C# code, which it did not have in the first case. Perhaps this was something that made people confused. Although many people thought BIML was harder to understand in the second case, many managed to answer correctly or partly correctly when asked what the AMO or BIML code’s purpose was.

The results here could have been affected a little bit by the order that the user reads each code sample (first AMO 1, then BIML 1, BIML 2, and finally AMO 2). The code example does the same thing in each of the two cases, thus, reading it a second time but in another language might make it easier. How much this has affected the result is assumed to be low, though. Overall it is hard to say which of AMO or BIML that is easier to understand from the survey result above. As it seems, it depends from case to case.
5.2.2 Readability

For measuring the readability of AMO and BIML, the participants were questioned to rank the readability of each code sample from 1-6, where one is very difficult, and six is very easy. The results are presented in the following table and diagram.

How easy is it to understand the code sample from a scale 1 to 6 (where 1=very difficult, 6 = very easy)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO 1</td>
<td>-</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>BIML 1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>AMO 2</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>BIML 2</td>
<td>-</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Here we can see that the participants thought that BIML was easier to read than API AMO in the first case, but it switched in the second instance. From considering BIML to be easy or very easy in the first case, it changed in the second instance when the BIML code had included C# code. Perhaps the participants thought that mixing two code languages like the BIML code did in the second case above, makes it harder to read.

5.2.3 Test Group Preferability

After each of the two test cases, the participants were asked to select which code language they would prefer to work with based on the code samples that were shown. The question with results for each test case is shown below.

Which code sample would you prefer to work with in each case?

<table>
<thead>
<tr>
<th>Test Case</th>
<th>AMO</th>
<th>BIML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>4 (3)</td>
<td>10 (6)</td>
</tr>
<tr>
<td>Case 2</td>
<td>13 (8)</td>
<td>1 (1)</td>
</tr>
</tbody>
</table>

In the first case, there were more who preferred the BIML sample. In the second case, it changed, and the participants rather preferred AMO. If these numbers are summarised, AMO gets 17 votes, while BIML gets 11. Hence, when asked which one they would rather work with, the participants preferred AMO over BIML.

5.3 Documentation and Support

It is important that the tool for OLAP cube creation is well documented, and also preferably has support available. If and when you get stuck coding, it is nice to be able to find help fast so that you can continue developing your program.
5.3.1 Documentation

The amount of available documentation on each of the two prototypes are described below.

**AMO**: Microsoft have created several documents that help you with building your AMO code. Their forum [30] is also a good source for finding help when you get stuck. At the date 2016-11-23, Microsoft’s Forum had just under 2 million answered posts overall, but around 26 000 answered posts in the field of SQL Server Analysis Services [31] (which AMO is a part of). When developing cubes using AMO, the Microsoft’s forums and articles might be the best source for getting help with your coding.

**BIML**: The amount of documentation for BIML is much more sparse, where Varigence’s forum [32] at the same date, had 1364 posts around BIML code. This amount of posts includes both SSAS cube creation, but also SSIS (e.g. ETL construction). Also a few posts on the website of Solisyon [33] was sometimes useful. Overall it feels like for BIML there is more documentation about SSIS than SSAS.

5.3.2 Support

When you have a specific question and can not get help via web forums, etc. It is good that support exists for the technique you have chosen. The idea of how to measure this arose when we got stuck during the development of the BIML prototype, more specifically of how to include sub-tables of dimension tables into a data source view. This question was tested for both AMO and BIML, where the time was measured until an answer was given. The result of this test is shown below in Table 3.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Web forum</th>
<th>Personal Support via Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO</td>
<td>Microsoft forum: 1 Hour, 23 Minutes [34]</td>
<td>Not tested, since it costs and answer was given by a user in their forum.</td>
</tr>
<tr>
<td>BIML</td>
<td>Varigence BIML forum: 3 weeks, and still no answer on the 30:th of November. [35]</td>
<td>Varigence Support: 3 weeks, and still no answer on the 30:th of November. (Also, A reminder email was sent after 1 Week).</td>
</tr>
</tbody>
</table>

Note however that we used the free trial version of BIML when the support request was sent, and we were not a paying customer. Hence, the result does not say if an answer could have been given if we were a paying customer. However, have in mind that posting a help post in Microsoft forums [31] is free as well, and an answer was given in a short amount of time. Also, In the mail to Varigence support, the problem was mentioned together with an explanation that it was going to be used in this master thesis.

Summarised, AMO gets the upper hand in this measurement, At least at the time when this test was performed (November 2016). One could also guess that perhaps it is also so that the user base is bigger for AMO than BIML.


5.4 Maintainability

For measuring maintainability on this thesis’ two prototypes, we will compare the lines of code for a couple of relevant sections, as well as how complex the code language is, using the Cyclomatic Complexity measure. Since it was not possible to fully implement all the desired features in the BIML prototype (which was described in the completeness section), the maintainability measurement will only be measuring the code that has the collective actions between BIML and AMO, which resulted in a cube. Hence, we will not count code that exists for a particular feature in the AMO prototype that was unable to be implemented in the BIML prototype.

5.5 Lines of code

The lines of code are measured for both of the two prototypes in four distinct parts in the cube generation process. The estimated numbers do not include white spaces, comments, or extra code lines for includes, or for easier readability. We also only count the things that both AMO and BIML could handle. Also, for BIML, the numbers are the actual BIML code with included C# code (i.e. not the expanded BimlScript), and for both prototypes, the code column width is set to maximum 80 signs and indentation set to width 4. The result is shown below.

In this measurement, the code for the BIML data source and view creation also involved counting the declarations of tables (but not dimensions). This code could perhaps have been 10-15 rows smaller if additional optimisation had been done to it, but due to lack of time, it was decided to stop and not optimise it further. We can see in this measurement that the BIML code requires fewer lines of code compared to API AMO. If the BIML code for creation of the data source and view could be optimised, losing 10-15 rows, it would be similar to the rows of code required in the AMO prototype. Have in mind that this evaluation was performed on the code that worked for both prototypes.
5.6 Cyclomatic Complexity

The Cyclomatic Complexity, \( v(G) \), is a technique for measuring complexity by counting the number of linearly independent paths through a program. It is defined by the following formula, stated by Thomas J. McCabe [36, 308-309].

\[
v(G) = e - n + p
\]  

(5.1)

where \( n \) = number of nodes, \( e \) = number of edges, and \( p \) = the amount of exit points. Nodes means when a decision has to be made (e.g. if-cases, allocations, etc.) and edges is the link in-between those nodes. The higher number of cyclomatic complexity, the more complex the code is and the higher risk there is that errors may exist inside the code [21]. This formula can, according to McCabe [36, 314-315], be simplified into:

\[
v(G) = \pi + 1
\]  

(5.2)

where \( \pi \) is the amount of predicates. Hence, To test our two prototypes’ Cyclomatic complexity, we could count the amount of predicates inside the code, i.e. If-cases, Loops, etc. and add one. If we do this, grouped into the same modules as when we measured the lines of code as in the previous section, we get the following data as shown in the diagram below. Also remember that to be able to test the two prototypes equally, we only count code that worked for both prototypes.

Here for BIML, when creating the data source view, all table declarations were counted as well. One reason why the Cyclomatic Complexity is higher here is that, when columns are declared for each table, there are several if-cases that sort out so that the column gets the correct datatype. This datatype may differ from the read in datatype that follows the SQL server standard and therefore needs to be re-declared according to BIML datatypes [37]. Summarising the complexity values for AMO and BIML gives an equal Cyclomatic Complexity number of 31, so it is hard to say which one that is more complex. However,
if we just measure the tasks independently, we see that BIML is less complex in two out of the four tasks. We can also see that one task has the same complexity, and AMO is only less complex at one out of four tasks. Summarised, the methods are equally complex.

5.7 Development speed

It is good that the technique for SSAS cube creation can be used to build much code in shorter amount of time. During the development of this thesis' two prototypes, time was measured in hours spent with developing each prototype. Before the implementation started, some key checkpoints were selected that each of the prototypes should be able to handle. When the prototypes could handle that checkpoint, the time required to get to there was noted. The result is shown below. Note that since both prototypes needed an internal structure of the input database tables and relations, the first checkpoint is the same for both prototypes, where it took 53 hours to create that internal structure. Also, this diagram is based on the time for when things were succeeded. Hence, it does not include the problems that were experienced with BIML (as described earlier in this chapter, in the completeness section). Also, the hours shown does not include the time needed to understand the basics concept about cubes, dimensions, data source view, etc.

From this diagram, we can see that the time required to build the prototypes by each method is quite similar. Some data for the BIML prototype does not exist, though, but to generate a cube on a simple star schema with only single-column key relationships and without sub-tables, require almost the same amount of time. Summarised, they perform equally regarding prototype development time.
5.8 Saved Developer Time

One aim of using a more semi-automatic way of generating cubes instead of building them manually is to save time and make the process more efficient. It also enables one to produce cubes more often, even for test purposes and small changes.

To measure the saved developer time by implementing a program in AMO or BIML instead of manually creating cubes was measured by letting an IT-consultant at Trimma build a Dimension. Unfortunately, it was not possible at the time when this thesis was performed, letting the IT-consultant create a whole cube manually since it would take up too much time for that employee at that moment. However, building only one dimension takes less time and was possible to perform. During the implementation of that dimension, the time was measured until it was finished. This time value gives an approximate value of how much time it would take to build \( n \) more dimensions, and then, later on, a cube. The dimension that was created was the dim_KST dimension, with one underlying sub-table.

Table 4 Manual development speed of dimension dim_KST and one sub-table

<table>
<thead>
<tr>
<th>Test person</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test person 1</td>
<td>7 Min. 45 s.</td>
</tr>
<tr>
<td>Test person 2</td>
<td>8 Min 20 s.</td>
</tr>
<tr>
<td>Average</td>
<td>8 Min 12.5 s.</td>
</tr>
</tbody>
</table>

In average it took 8 minutes and 12.5 seconds to create this dimension. If the test person had created five dimensions in total, it would have taken \( 492,5 \times 5 = 2462.5 \) seconds, i.e. circa 41 Minutes. After this, you have to create a cube based on those established dimensions, and also add cube partitions to it. Since there was no possibility when this thesis took place to measure the time to create a whole cube manually, an approximate value has been given by an employee at Trimma who has worked the most with creating cubes. The approximate value of creating a cube from five dimensions, with only one partition for the cube, not counting in creating predefined MDX or SSAS actions, takes about 10 minutes. All these times gives a result that:

- An approximate value of the time required to build a cube manually with five dimensions, and one cube partition is 51 minutes.

It is interesting to know how many times you should have to build a cube until it is worth spending time implementing a program that does this more automatically instead of doing it manually every time. By comparing the approximate value of the time spent to build a cube manually, with the time it took to create one of the prototypes for this thesis, we can get a rough estimate for this. Since it was possible with the AMO prototype to extend it one step further than BIML, to also include sub-tables, this is the time we will use.

- It took approximately 51 minutes, or 0.85 hours to create a cube manually, and 111 hours to build a program to do this more automatically. Hence, it will be worth spending the time developing a more automatic program if you plan to generate more than 131 cubes. However, if you also want to further automate the SSAS actions and MDX, more time could be saved.
5.9 License costs

A small but still interesting aspect when choosing between AMO and BIML is the licence and support cost. These costs are shown in the table below, measured on the date 2016-11-21. Note that the license costs shown is only for one license and that we only focus on the BI software needed for creating cubes, even though each IDE might include more features for implementing other things as well. Also, we only measure the cost for one month and one year.

For being able to build a cube using AMO, it is preferable that you use an IDE built for C#. One IDE available is Microsoft Visual Studio, where the free "Community" model should be sufficient together with the AMO library. A non-free Professional version also exist if extra features are desired [38].

For building cubes with BIML, you must have the Mist IDE with the built in BIML SSAS compiler [19].

<table>
<thead>
<tr>
<th>Tool</th>
<th>Trial</th>
<th>Monthly ($)</th>
<th>Yearly ($)</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO Visual Studio Community:</td>
<td>free [39].</td>
<td>Visual studio Community:</td>
<td>free [39].</td>
<td>Reporting/browsing a problem in their community database [42] and writing/reading in their forums [30] is free. Getting personal code-support by a Microsoft engineer costs though, how much depends on the problem you are having. However, MSDN subscribers get up to four free engineer calls each year [43].</td>
</tr>
<tr>
<td>Professional: 4 months [40]</td>
<td></td>
<td>Professional Cloud subscription: 45$ [41]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIML 14 days [44]</td>
<td>249$ [45]</td>
<td>3999$ [45]</td>
<td>Support and upgrades are included if a license is purchased [45]. Support seems also to be included during evaluation of the product (according to their support email auto-reponse). Writing on their help forum is free [46].</td>
<td></td>
</tr>
</tbody>
</table>

Since the Microsoft Visual Studio Community is sufficient and free to use, API AMO would be the preferable option in the license cost evaluation. Even if the professional standard subscription is bought, the yearly cost is cheaper than the BIML Mist IDE. The support might cost in API AMO if you require personal help, however, their code base and forum contain more than two million questions [30], so API AMO is still the preferable choice regarding costs since there is a big chance that a potential problem that might appear, could be declared in the forum.
## 5.10 Result Summary

<table>
<thead>
<tr>
<th>Checklist item</th>
<th>Highest Score</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>AMO</td>
<td>In our evaluation it was possible to show completeness for AMO, where it could handle both cases of schema structures and had correct internal numbers. This could not be totally shown for BIML.</td>
</tr>
<tr>
<td>Usability</td>
<td>Equal</td>
<td>Based on a survey made on Trimma’s employees, it is difficult to say which is more readable and understandable. It depends from case to case.</td>
</tr>
<tr>
<td>Preferability of Trimma’s employees</td>
<td>AMO</td>
<td>When asked during the survey which code language Trimma’s employees prefered.</td>
</tr>
<tr>
<td>Documentation and Support</td>
<td>AMO</td>
<td>A problem question was posted on each of the techniques’ forums (and a mail to the support of the BIML creator: Varigence). An answer was given within 2 hours for AMO. No answer was given for BIML when this measurement stopped after three weeks. Also, the available documentation is bigger for AMO compared to BIML.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>BIML</td>
<td>This was tested on the actions that both prototypes could handle. BIML required less rows of code to perform a specific action. However, the Cyclomatic Complexity was somewhat equal between them.</td>
</tr>
<tr>
<td>Saved Developer Time</td>
<td>-</td>
<td>It takes approximately 51 minutes to manually create a standard cube with five dimensions and one partition, without predefined MDX or Cube Actions. An automatic program does this in a few seconds. The implementation of a semi-automatic program takes about 111 hours to implement (AMO case). It will be worth the time if you build more than 131 cubes. However, this value is a rough approximate, see discussion for more information.</td>
</tr>
<tr>
<td>Checklist item</td>
<td>Highest Score</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>Development Speed</td>
<td>Equal</td>
<td>Only counting to the point that a cube exist (without dimension sub-tables), the prototype development time was somewhat equal. After that point, it was difficult to continue developing the BIML prototype to the next, extended, case. The documentation was very sparse and it was not understandable how one should do to reach case 2.</td>
</tr>
</tbody>
</table>

| License costs | AMO | The costs for using AMO is cheaper than BIML. AMO could sometimes even be totally free depending on desired Visual studio features. |
6 Discussion and Conclusion

6.1 Discussion

When the calculation was made for the saved developer time, the resulting values is a rough approximate. This since when both participants built the dimension, they both used a piece of help structure that helped with creating a base that they continued to build upon. Hence, it is a big risk that it takes a longer time to build a cube than just 51 minutes. During a discussion with one of the participants about how much time that would be required if that helping structure was not used, the answer was that around 4 hours would be needed to build a basic cube, for a person that has built cubes manually before. If that number is correct, then it will be worth creating a program that does this more automatically if you are going to create more than 28 cubes.

When measuring the license costs for BIML it was noticed that the cost for one monthly subscription license payed during a time period of 12 months is cheaper than paying for one yearly license. The reason why it is cheaper could not be found.

For the readability of each of the prototype code languages it is hard to decide which of AMO or BIML that is more readable. It might be simpler to say which one is better by picking the one that gets the highest summarised points. However, this gives AMO 108 points and BIML 110 points, so it is still difficult to select which one is preferable concerning readability. However, perhaps AMO has a little advantage since the people that answered the survey seem to think that BIML code gets harder to read when C# code is included inside the BIML code (which is preferable if one wants more slim and efficient BIML code).

Regarding the documentation for AMO and BIML, it was difficult to find answers to problems that arose during the implementation of the BIML prototype. For the AMO prototype, it was easier to get answers when help was needed. If BIML had more documentation around cube creation, then the results of this thesis could perhaps have been different since it might have been possible to make the BIML prototype to include a snowflake schema case.

It was possible to implement both cases into the AMO prototype, but it was not possible to implement the second in BIML. The problem lies in how one should build the relations between tables inside the data source view when there are table columns that belong to several key relationships. Many different ways of trying to solve this was performed, but these either lead to an error saying that a column name exists at several places in the BIML code, or that the output was defective. It feels like the problems that occurred with BIML should be possible to solve, but the lack of documentation gave that a solution was not found. If the documentation and support for BIML get better in the future, then BIML could be a possible option, but not as it is right now.
6.2 Conclusion

Our problem statement was declared as follows:

“Out of all available techniques for OLAP cube generation, can we determine which technique that is the best choice regarding completeness, good usability and high efficiency?”

According to the evaluations performed on the two prototypes built by each technique, AMO is the superior choice. This because BIML had an insufficient amount of documentation around cube creation, which might have had an influence on why it was not possible to conclude total completeness for BIML in this thesis’ evaluation. Completeness was, however, satisfied for AMO, at least according to the set completeness test framework. Thus, AMO performs best, better than the manual method currently employed by Trimma since it could do the same work in a more efficient manner. Also, a recommendation for Trimma’s future work is to use AMO before BIML at the current stage since it could not be secured whether BIML is complete enough.

6.3 Future work

This thesis focused only on creating the internal structure of a cube from scratch based on tables inside a database that has relations like a star/snowflake schema. More things could be done with this where the cubes, later on, could have predefined Multidimensional Expression (MDX) [47] and Cube Actions to it, which would enable additional features for the end user [48].

Implement a more automatic SQL Server Integration Services (SSIS) [3] ETL tool and evaluate different ways or techniques for doing this. During the implementation of the BIML prototype, it felt like it was more usable and better developed for that cause. Hence, BIML might be a good participant for this since it might perform better there compared to SSAS creation. Microsoft also has tools for this [3]. Other techniques might also exist.
References


A Appendix

A.1 Implementation case 1: Star Schema

Figure 9: A star schema showing a fact table (highlighted in blue) that includes five dimension tables. This was the first implementation goal. The dimensions here are having Swedish names, but the dimensions are Kst = Cost Center, Proj = Project, Resultatkonto = Results account, Mortp = Counterpart, and Time.
A.2 Implementation case 2: Snowflake Schema

Figure 10: A snowflake schema with one fact table (highlighted in blue) and five dimension tables, where Kst and Resultatkonto has two sub-tables each. This was the second implementation goal after that each prototype could handle the first implementation case (Star Schema)
A.3 User Survey

**User Survey**

Do you have any prior knowledge of fundamental concepts concerning OLAP cubes (e.g. Dimensions, etc.)?

Yes  No

**Code example 1:** Please read the code sample below, then answer the following questions.

```csharp
DataTable srcTab = db.DataSourceViews[0].Schema.Tables["dim_Time"]; DataColumn srcCol = srcTab.Columns["Date"]; DimensionAttribute attribDate = newDim.Attributes.Add("Date"); attribDate.Type = AttributeType.Date; attribDate.Usage = AttributeUsage.Key; attribDate.AttributeHierarchyVisible = false; attribDate.KeyColumns.Add(srcTab.TableName, srcCol.ColumnName, OleDbDbTypeConverter.GetRestrictedOleDbType(srcCol.DataType)); srcCol = srcTab.Columns["DateName"]; attribDate.NameColumn = new DataItem(srcTab.TableName, srcCol.ColumnName, OleDbDbTypeConverter.GetRestrictedOleDbType(srcCol.DataType), srcCol.MaxLength);```

Was this code sample easy to understand?

Yes  No

What is the purpose of the code? (i.e. what does it do?). Please answer with a few sentences. If the sample was incomprehensible, answer “don’t know”.

From a scale, 1 to 6, how easy was is to read the code sample? (1=very hard, 6=very easy)

1  2  3  4  5  6
**Code example 2:** Please read the code sample below, then answer the following questions.

```xml
<Dimension Name="..." DimensionType="...">
  <Attributes>
    <Attribute Usage="Key" Name="Date" AttributeType="Date"
      AttributeHierarchyVisible="false">
      <KeyColumns>
        <KeyColumn ColumnName="Date" />
      </KeyColumns>
      <NameColumn ColumnName="DateName" />
    </Attribute>
  </Attributes>
</Dimension>
```

Was this code sample easy to understand?

Yes        No

What is the purpose of the code? (i.e. what does it do?). Please answer with a few sentences. If the sample was incomprehensible, answer “don’t know”.

From a scale, 1 to 6, how easy was it to read the code sample? (1=very hard, 6=very easy)

1 2 3 4 5 6

Comparing the code language in code sample 1 and 2, which one would you prefer to work with?

Code sample 1        Code sample 2
**Code example 3:** Please read the code sample below, then answer the following questions.

```
<#foreach(hier in hierarchyList){#>
    <#i=1;#>
    <#List<String> lvlList = hierListLevels.getList(hier.Name()); #>
    <Hierarchy Name="<#=hier.name()#>" AllMemberName="All">
    <Levels>
        <#foreach(string level in lvlList){#>
        <# String lvlName = "Level " + i;#>
        <Level Name="<#=lvlName#>" AttributeName="<#=lvlName#>"
             HideMemberIf="NoName"/>
        <#}#>
        <#i++;#>
    </Levels>
    </Hierarchy>
<#}#>
```

Was this code sample easy to understand?

Yes  No

What is the purpose of the code? (i.e. what does it do?). Please answer with a few sentences. If the sample was incomprehensible, answer “don’t know”.

From a scale, 1 to 6, how easy was is to read the code sample? (1=very hard, 6=very easy)

```
1  2  3  4  5  6
```
**Code example 4:** Please read the code sample below, then answer the following questions.

```csharp
foreach (HierarchyPart oneHier in oneDim.getHierarchyList()){
    int i=1;
    Hierarchy oneHierarchy = newDim.Hierarchies.Add(oneHier.getName());
    oneHierarchy.AllMemberName = "All";
    foreach (String oneLevel in oneHier.getLevelList()){
        ...
        Level oneL = new Level("Level " + i);
        oneL.HideMemberIf = HideIfValue.NoName;
        oneL.SourceAttribute = oneLevelAttrib;
        oneHierarchy.Levels.Add(oneL);
        i++;
    }
}
```

Was this code sample easy to understand?

Yes  No

What is the purpose of the code? (i.e. what does it do?). Please answer with a few sentences. If the sample was incomprehensible, answer “don’t know”.

From a scale, 1 to 6, how easy was is to read the code sample? (1=very hard, 6=very easy)

1  2  3  4  5  6

Comparing the code language in code sample 3 and 4, which one would you prefer to work with?

Code sample 3  Code sample 4

Thank you for your participation!