Bachelor’s thesis

Information and Communication Technology

2020

Joel Juntunen

THE DEVELOPMENT OF AN ADAPTABLE DATA TRANSFER INTERFACE

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology

2020 | 18

Joel Juntunen

THE DEVELOPMENT OF AN ADAPTABLE DATA TRANSFER INTERFACE

The capacity of data warehousing has grown in due course with the technological development in the field. Simultaneously, the advancing diversity has created many challenges and opportunities for the companies in the field. Developers in data warehousing often end up face to face with seizing those opportunities for the benefit of the company. The best opportunities for serving customers come from integrating new data and work hours should be dedicated to it accordingly.

Unfortunately, due to the rigidness of database structures, uniformity between old and new data is not a given. Luckily for these developers, many data transfer tools are developed for the newest file formats before their widespread implementation. Nevertheless, the plurality of the formats is such that learning and using so many tools is no longer a simple task for a large data warehousing unit.

The thesis aimed to create an application for unifying data transfer protocols under a single interface. In this way developers could handle a plethora of different data transfer protocols without needing the knowledge of how to configure filetype specific tools. In other words, the application acts as an intermediary between the developer and different third-party data transfer protocols so that setting up configuration and optimising protocols can be carried out at the application development face rather than at the start of every new solution.

The necessity for this kind of application became apparent when the data warehousing team of the commissioning company needed to bring in new datatypes at a faster pace than it was efficient to build individual solutions for each source.

The end product of the thesis is an inhouse data transfer application that currently supports JSON, CSV, XLS, and XLSX file types and Azure Storage cloud service files. The first implemented target type is into an SQL database. The source code, which was written in C#, was designed so that adding new source and target types will be easy when the need arises. This makes the company data warehousing team much more flexible and adaptable to new needs as the data pool expands. Notably the ability to add new source types will enable smoother cooperation with partner organisations as offered data can be implemented into the data lake with less friction. The application also reduces the technical dept of the organisation, replacing work intensive protocols from still active old third-party tools.

KEYWORDS:

C#, FTP, data warehousing, data transfer, databases, SQL, software development, interfaces

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja Viestintätekniikan Koulutus

2020 | 18

Joel Juntunen

TIEDONSIIRRON TYÖKALUNRAJAPINNAN MUKAUTUVA KEHITYS

Tietokantojen koko on teknologian kehittyessä kasvanut suunnattomasti. Koon kasvaessa myös monimuotoisuus on lisääntynyt luoden haasteita ja mahdollisuuksia. Kehittäjänä tietokantojen parissa vastaan tulee kysymys yrityksen kehittymisestä. Tehtävää isossa yrityksessä riittää aina ylläpidon parissa, mutta suurimmat mahdollisuudet tarjota parempia palveluja asiakkaille nousevat uuden datan toimittamisesta. Kun dataa saadaan uusista lähteistä, sen yksimuotoisuutta ei voida taata käytössä olevan tietokannan rakenteen kanssa. Usein tiedonsiirtotyökalut uusille tiedostotyypeille ovat valmiiksi olemassa, mutta kun tietotyyppejä on paljon, ei niiden kaikkien käyttäminen välttämättä olekaan yksinkertainen tehtävä tietokantayksikölle.

Opinnäytetyössä kehitettiin tiedonsiirron protokollia yhdistävä sovellus, jonka pääasiallinen tarkoitus on yhtenäistää protokollat yhden käyttöliittymän taakse. Täten työmäärä, joka tarvitaan uuden tiedostotyypin tuomiseen tietokantaan, sijoittuu työkalun kehittämiseen, jonka jälkeen se on saatavilla kaikille kehittäjille.

Sovellus toimii siis rajapintana useille muiden palveluntarjoajien tiedonsiirtoprotokolille. Se luo automaattisesti kaikki tarvittavat konfiguraatiotiedostot ja toteuttaa tiedonsiirron.

Työn tarve nousi esille, kun uudentyyppistä lähdejärjestelmistä oli tarve tuoda dataa tietokantaan ripeämmällä tahdilla kuin kehittäjille oli tehokasta tehdä niille yksittäisiä ratkaisuja.

Opinnäytetyön lopputuloksena syntyi yhtiön sisäinen tiedonsiirtosovellus, joka tukee testatusti lähdetyyppeinä pilvipalvelu Azure Storage tiedostoja sekä json-, csv-, xls-, ja xlsx-tyyppisiä tiedostoja. Ensimmäiseksi kohdetyypiksi kehitettiin lataus SQL-tietokantaan. Sovelluksen lähdekoodi toteutettiin siten, että uusia lähde- ja kohdetyyppejä on yksinkertaista lisätä tarvittaessa. Tämä tekee yhtiön tietokantatiimistä paljon notkeamman ja kykenevämmän sopeutumaan uusiin tarpeisiin tietokannan laajentuessa. Varsinkin uusien lähdetiedostotyyppien lisääminen mahdollistaa sujuvamman yhteistyön uusien kumppanien kanssa. Sovellus vähentää myös yhtiön teknistä velkaa korvaten työläitä protokollia vanhoista vielä käytössä olevista kolmansien osapuolien työkaluista.

ASIASANAT:

C#, FTP, tietovarastot, tiedonsiirto, tekstitietokannat, SQL, ohjelmistosuunnittelu, käyttöliittymät

CONTENT

LIST OF ABBREVIATIONS

BOM	Byte Order Mark
CSV	Comma Separated File
DBMS	Database Management System
DWH	Data Warehouse
ELT	Extract, Load, and Transform
FTP	File Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MSSIS	Microsoft SQL Server Integration Services
SQL	Structured Query Language Source
TPT	Teradata Parallel Transporter
XLS	Excel Spreadsheet File
XLSX	Excel Microsoft Office Open XML Format Spreadsheet File

1 INTRODUCTION

There was a request to the data warehousing unit of the commissioning company to bring in data from a new partner. The main stepping stone was that the partner provided data in a format that was not already integrated into the existing Extract, Load, Transform (ELT) workflow. This was not the first time such a request had been made and from previous experience it was known that creating and upkeeping a separate solution for a single file format would create work with little return value and could easily end up in technical dept.

The thesis sets the goal of developing an adaptable data transfer interface to take care of the data formats that are outside the standard ELT workflow of the commissioning company. The scope of the thesis was set to cover some of the most common formats, and it is designed in a way that allows it to be easily expanded in the future. The sole load target is the database running on Teradata software. Additional load targets are planned to be added in the future but were left outside the scope of the thesis. Through this methodology, all requirements for new file types could be integrated under one software and developed in a structured way.

The benefits for this approach are many. Firstly, it reduces the amount of technical knowledge the team needs to maintain, as there are not multiple unconnected applications involved in updating the database. This reduces the amount of information in both configuring and maintaining said tools. Running all nonconventional data loads through a single software also makes it easier to schedule the loads in the production environment. As the software will be adapted for use within the Data Warehouse (DWH) unit, it will also be able to replace already existing load formats with work intensive workflows, reducing technical dept. Additionally, as all file loads of predetermined types will be run through the software, transfer protocols will need to be optimised only once.

Throughout this thesis, the development of the application will be explained, and the benefits will be described in detail.

2 THEORY AND BACKGROUND

2.1 Databases

Hugh Darwen defines databases as following: “A database is an organized, machine-readable collection of symbols, to be interpreted as a true account of some enterprise. A database is machine-updatable too, and so must also be a collection of variables” (Darwen 2010, 14).

In order to enable developers to interact with these machine-readable structures, information is displayed through a logical database using a Database Management System (DBMS) (Date 2013). These DBMS are usually provided and upkept by the service provider that supplies the database services in general.

This thesis revolves around creating an application for loading data to a database

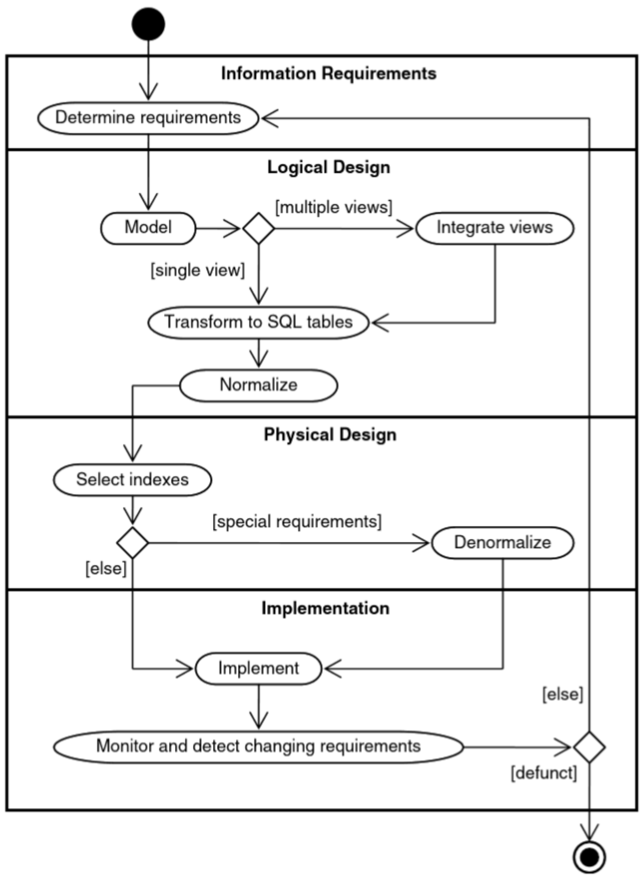


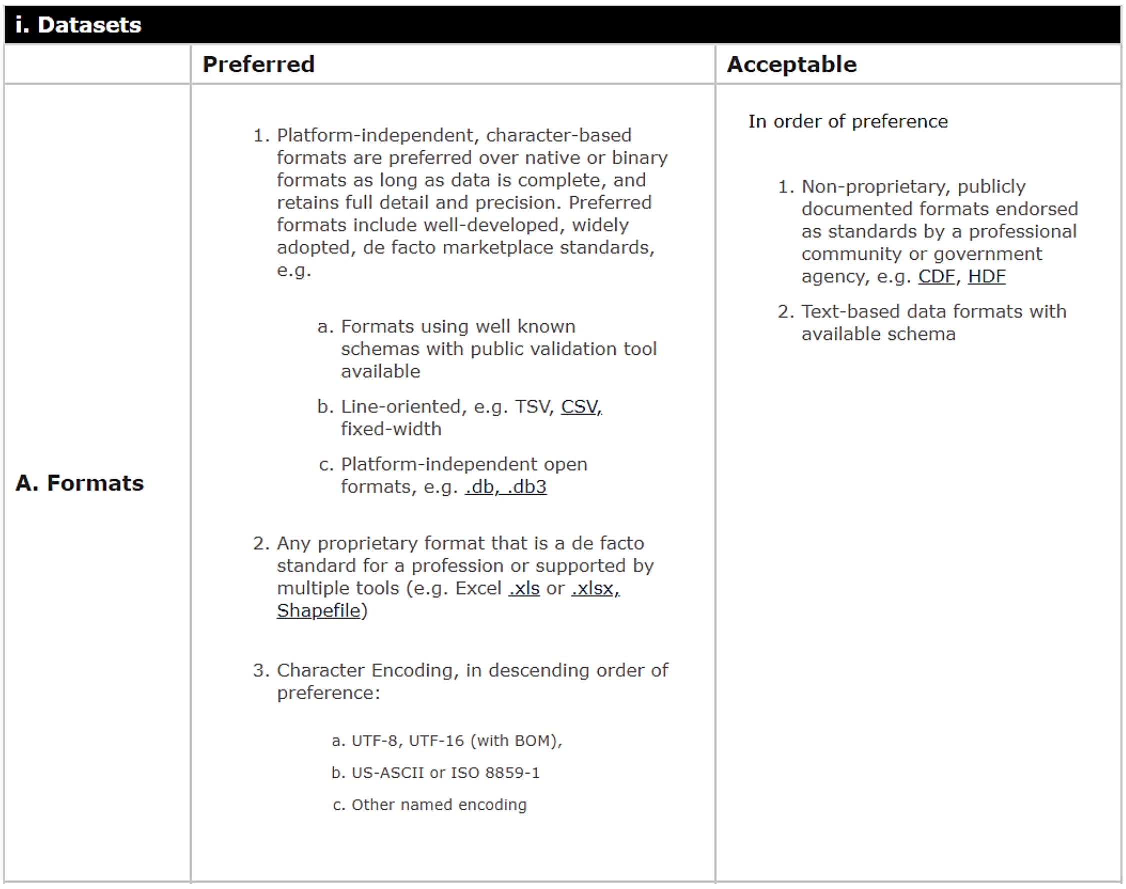
Figure 1. Typical life cycle of a database.

Figure 1 shows the typical life cycle of database (Teorey et al. 2011, 4). The application serves the loading of data for changing requirements.

2.2 Data transfer

As a database is redundant without data, DMBS often provide ways to load information from different sources. Offering multiple source types adds flexibility to the end users of the database.

There are multiple widely used file types suitable for datasets. Table 1 lists only the formats recommended by the U.S. Congress.



i. Datasets

	Preferred	Acceptable
A. Formats	1. Platform-independent, character-based formats are preferred over native or binary formats as long as data is complete, and retains full detail and precision. Preferred formats include well-developed, widely adopted, de facto marketplace standards, e.g. a. Formats using well known schemas with public validation tool available b. Line-oriented, e.g. TSV, CSV, fixed-width c. Platform-independent open formats, e.g. .db, .db3 2. Any proprietary format that is a de facto standard for a profession or supported by multiple tools (e.g. Excel .xls or .xlsx, Shapefile) 3. Character Encoding, in descending order of preference: a. UTF-8, UTF-16 (with BOM), b. US-ASCII or ISO 8859-1 c. Other named encoding	In order of preference 1. Non-proprietary, publicly documented formats endorsed as standards by a professional community or government agency, e.g. CDF, HDF 2. Text-based data formats with available schema

Table 1. Formats recommendation (Library of Congress 2020).

The variety of formats creates the need to know multiple data transfer tools as different service providers tend to supply their data in different formats.

Database service providers often offer a set of tools to import data into the database they provide. These tools often cover the most widely used formats like .csv and .xlsx and do a good job at optimising the transfers. At a glance the aforementioned tools are similar to the application developed in this thesis but differ greatly in adaptability and ease-of-use. The plethora of existing file types and sources in modern data warehousing has made it challenging for a service provider to enable every type of data transfer protocol in their native tools and conversions are often required. Furthermore, as those tools require a different set of configurations, uniformity of use can often be low for the user.

Unifying data transfers under a single interface reduces the need for a full understanding of multiple tools, saving developer time.

Similar tools are likely to have been developed throughout the industry either in format specific solutions or under the same premise. However, due to the need of customisations companies require due to differing DMBS and security practices, these tools are not universally available for every type of database. It is likely that other organisations have adapted similar inhouse applications.

2.3 An overview of data loading

Loading data to a DWH is a part of the widely documented Extraction, loading, and transformation (ELT) process. To summarise the process, data is extracted from an internal or external source, transformed to fit the structure of the target database, and loaded into a DWH. In most cases, loading data to a DWH is initially performed as a bulk load to create the first construction of the table. This is then followed by an incremental load to upkeep the dataset as a set of changes as new insertions, deletions, and updates reach the warehouse. (Vassiliadis 2009, 12-13.)

The application developed in the thesis focuses mainly on the loading part of the process but in the case of some sources also performs rudimentary extraction and transformation. The transformation process only slightly modifies the data, focusing on redacting noise that could be unreadable characters for example. In general, to optimise work, reducing the developing and maintenance cost of the ELT process is important (Awiti et al. 2020, 1).

Most commonly data loading is done as an integrated part of the ELT process. Some major examples of ELT tools are Talend Data Integration and Microsoft SQL Server Integration Services. (Souibgui et al. 2019, 677-680.)

2.4 Examples of Data Load Protocols

Teradata Parallel Transporter

Teradata Parallel Transporter is an object-oriented client application that provides scalable, high-speed, parallel data extraction, loading, and updating. The main limitation of TPT is that it only supports databases, data storage devices, and comma separated values (CSV). (Teradata, 2020)

The main benefit of TPT is that it is made to support parallel transmission. Parallel transmission can load several bits simultaneously, increasing the efficiency of data loading. This is visualised in figure 2.

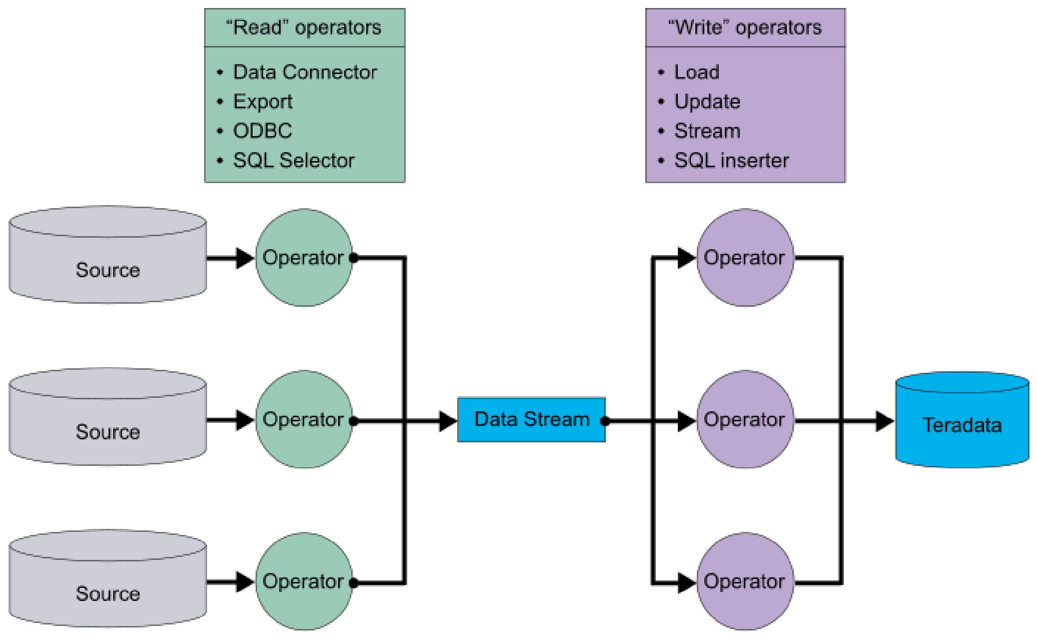


Figure 2. Teradata Parallel Transmission.

Microsoft SQL Server Integration Services

Microsoft SQL Server Integrations Services (MSSIS) is famous for holding the record for fastest ETL process ever and is developed by Microsoft corporation. Despite being a tool for automating the ELT process, MSSIS provides many additional functionalities like automating SQL Server Maintenance Plans. MSSIS has been commended for making it easy to build ELT solutions for inexperienced users and being generally easy to use with plenty of first and second-hand documentation. The biggest limitations of MSSIS are lack of support for non-windows operating systems and different integration styles. (Katragadda et al. 2015, 1-4.)

3 DEVELOPMENT WORK

3.1 Development environment

Programming language of the application was C# with the target framework .NET Core 3.1. This choice was made based on the existing support for a great number of third-party libraries for the planned data transfers.

The integrated development environment (IDE) used was Microsoft Visual Studio Pro.

3.2 Development process

3.2.1 Early stages – CSV Files

The development of the application started with transferring a .CSV source file to Teradata. Teradata Parallel Transporter (TPT) was chosen as the Data Transfer Protocol as it supports the Teradata's FastLoad functionality which is optimised for large files. The biggest challenge was automatically creating configuration files for TPT, which was done using the Text Template Transformation Toolkit (T4 Template) included in the .NET core. The application fetches column names and data types from target table and uses the retrieved metadata together with configuration files to create a .txt configuration file for the TPT transfer. As the nature of the application requires the usage of third-party tools, a large part of the workload went into implementing the usage of System.Diagnostics.ProcessStartInfo, which runs the TPT load, and building framework around it to handle input and errors.

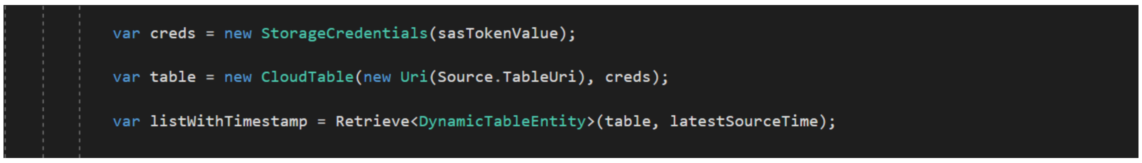
Nearing the testing phase, the need to handle file encodings became apparent as the test data included a large amount of Nordic special letters. The solution that was agreed upon was to generate a temporary runtime file into which the application would write the readable bits in the desired format. The runtime file would then be used as the source file for the data transfer and deleted after the execution. This was necessary not just for Nordic letters, but also for other special characters which the transfer protocol was unable to process. Detecting the encoding was originally done by reading the Byte Order Mark (BOM) of the source file. Nevertheless, later during the development it became apparent that there was a need to read files without BOM and an optional configuration setting was added to allow manually setting file encoding.

After the functionality was tested, an executable first version of the application was created and integrated into the scheduling environment of the company. One job was scheduled to run weekly.

3.2.2 Post-Deployment - Azure Data Storage

After the application was set up in production for the use of the organisation, the development work was aimed towards introducing a new source type for the database. The need to read Azure Data Storage had existed for a while but introducing a new tool for it had been deemed an unnecessary addition to the already prevalent list of tools the developers need to know. The need for such transfers is exactly why this unified data transfer tool was set into development.

In the case of Azure Data Factory, double set of functions was needed as no direct tool exists for loading from Data Factory to Teradata. The transfer of data is divided into two separate actions; pulling and pushing. Pulling data from Azure Data Storage has been made extremely simple via the CloudTable class provided in the Azure.Cosmos.Table NuGet package. This is show together with authentication in figure 3.



```
var creds = new StorageCredentials(sasTokenValue);

var table = new CloudTable(new Uri(Source.TableUri), creds);

var listWithTimestamp = Retrieve<DynamicTableEntity>(table, latestSourceTime);
```

Figure 3. Using Azure Data Storage libraries.

After pulling the data to a list, the application simply iterates through the extracted data entries and appends them to a Teradata Dataset, paying special care to the formatting of metadata.

For the transfer to Teradata to work, the presence of a timestamp for record time is mandatory. Therefore, a method was developed where the application looks for an inserted timestamp column in the target table and either resumes with records after the returned date or executes an initial load in case there are no records in the table. Finally, a Teradata provided library TdDataAdapter was used to load data into the database.

3.2.3 Further support – JSON and Excel files

The scope of the thesis was expanded to add support for .json, .xls, and .xlsx files. The development was approached through a limited scope, as adding a new data load protocol was deemed unnecessary for time being. The TPT loading was performing well so it was decided as the approach despite the lack of support for the data formats. The load was enabled by transforming the files to .CSV format using open source libraries. The transformation was a simple job for .JSON files, whereas Excel formats required more setting up with the more complicated structures around sheet numbers and equations.

Later in testing it became apparent that the library used for extracting data from the Excel files was not performing well in files that were more than half a gigabyte in size. The replacing extractor was set to be OLEDB Data Provider due to its documented reliability.

3.3 Flow of the application

The application is a .NET core console application that is run with arguments for execution id and configuration file path. For production purposes, the application is meant to run via an internal scheduling program, but it is provided in the form of an executable and can also be run manually in development phases.

The run is started with checking the validity of the arguments and both run specific and application wide configurations. In general, the tool runs as long as possible to provide the maximum amount of errors before being unable to continue. Once the local settings are confirmed to be correct, the application defines the interfaces to be used, performs the necessary file transformations, and establishes connection to source and target frameworks where applicable. Once the connection is established, the configuration-defined interface launches the corresponding transfer protocol. The interface class then returns the return output value of the protocol, resulting in the end status for the application. Figure 4 shows a simplified flow of the application.

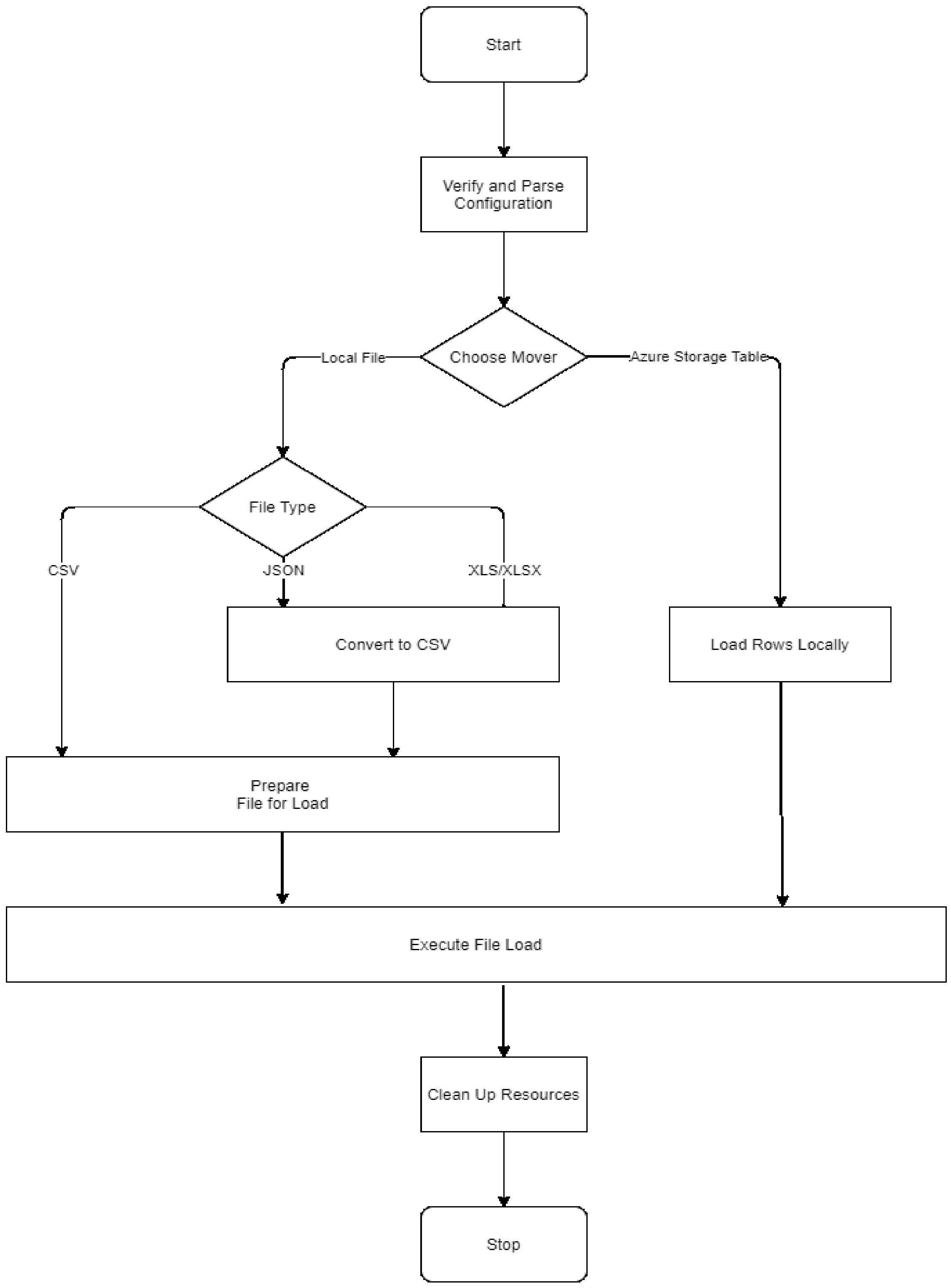
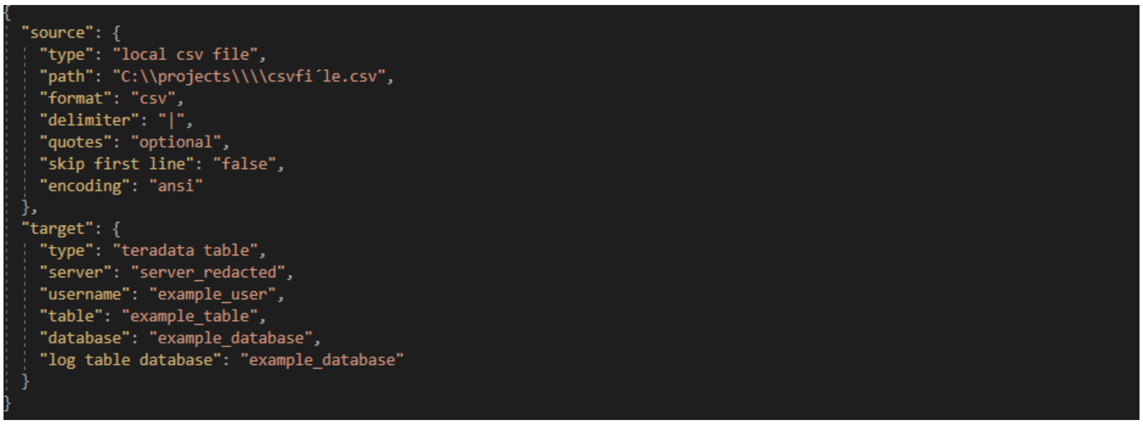


Figure 4. Simplified application flow of the thesis application.

3.3.1 Configuration

Due to ease of usage and reusability throughout the code, it was decided that the configuration of the application was to be done using local JSON files and the Newtonsoft.Json library. For security reasons, there was a need for two separate files; a general file for user identification and another for run specific source and target definitions. This way the run specific configuration can be saved in file sharing without the risk of accidentally sharing user credentials. Checks were implemented through the process to assure the validity of the configuration and to deliver more understandable error messages. Figure 5 shows an example of a load configuration file for the application.



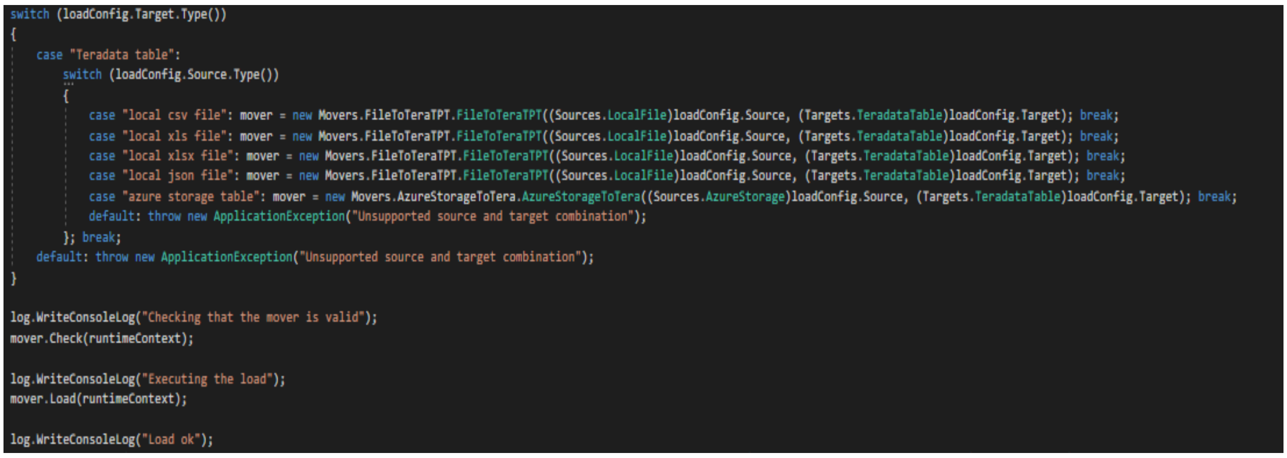
```
{
  "source": {
    "type": "local csv file",
    "path": "C:\\projects\\\\csvfi´le.csv",
    "format": "csv",
    "delimiter": "|",
    "quotes": "optional",
    "skip first line": "false",
    "encoding": "ansi"
  },
  "target": {
    "type": "teradata table",
    "server": "server_redacted",
    "username": "example_user",
    "table": "example_table",
    "database": "example_database",
    "log table database": "example_database"
  }
}
```

Figure 5. Example of a configuration file.

The variables retrieved from the configuration files are then transferred into a RuntimeContext class that is the general access layer for reusable runtime resources such as database connections.

3.3.2 Interfaces

After the initial functionality was implemented, the code was refactored to use interfaces. In C#, interface defines a contract that any implementing class or struct must implement the members of (Microsoft 2020). Defining the three interfaces of sources, targets, and movers, we were able to streamline the code execution and reduce processor load.



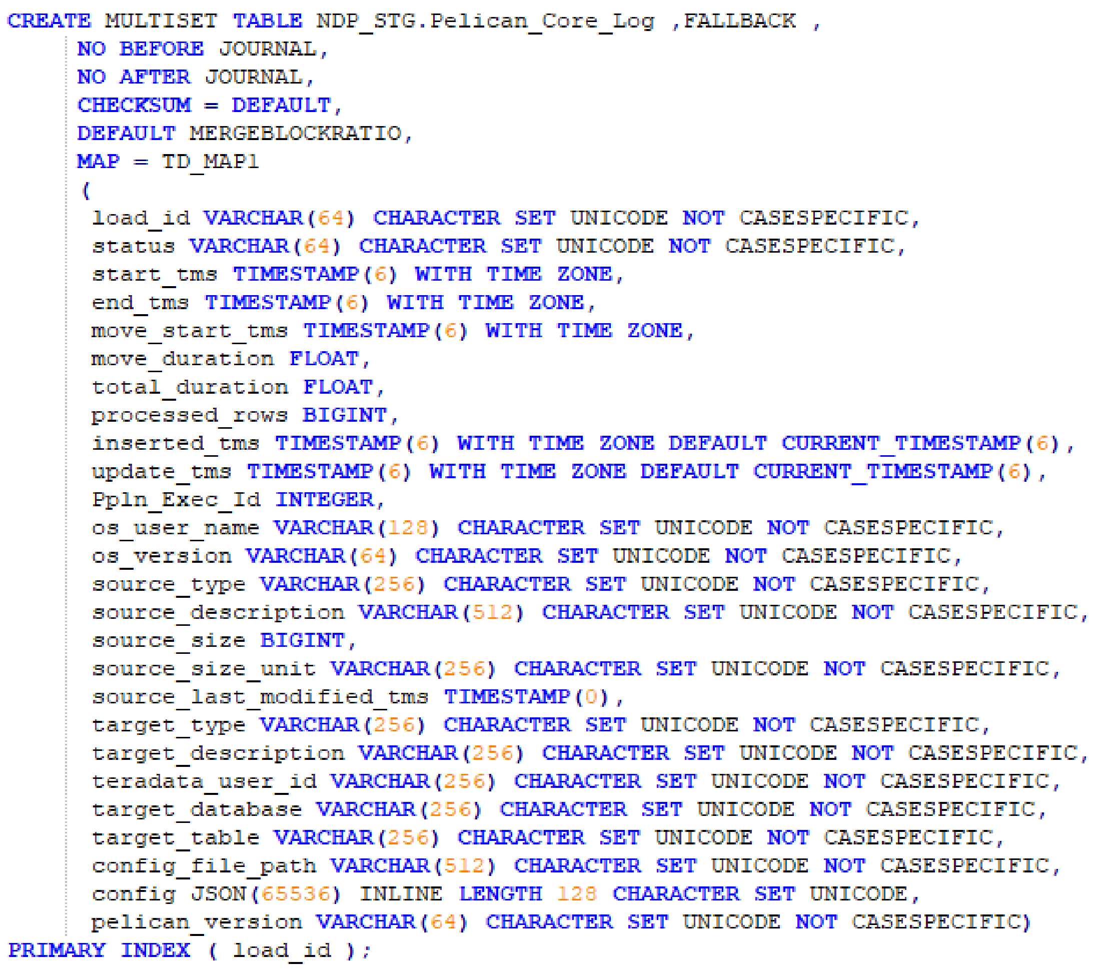
```
switch (loadConfig.Target.Type())
{
    case "Teradata table":
        switch (loadConfig.Source.Type())
        {
            case "local csv file": mover = new Movers.FileToTeraTPT.FileToTeraTPT((Sources.LocalFile)loadConfig.Source, (Targets.TeradataTable)loadConfig.Target); break;
            case "local xls file": mover = new Movers.FileToTeraTPT.FileToTeraTPT((Sources.LocalFile)loadConfig.Source, (Targets.TeradataTable)loadConfig.Target); break;
            case "local xlsx file": mover = new Movers.FileToTeraTPT.FileToTeraTPT((Sources.LocalFile)loadConfig.Source, (Targets.TeradataTable)loadConfig.Target); break;
            case "local json file": mover = new Movers.FileToTeraTPT.FileToTeraTPT((Sources.LocalFile)loadConfig.Source, (Targets.TeradataTable)loadConfig.Target); break;
            case "azure storage table": mover = new Movers.AzureStorageToTera.AzureStorageToTera((Sources.AzureStorage)loadConfig.Source, (Targets.TeradataTable)loadConfig.Target); break;
            default: throw new ApplicationException("Unsupported source and target combination");
        }; break;
    default: throw new ApplicationException("Unsupported source and target combination");
}

log.WriteConsoleLog("Checking that the mover is valid");
mover.Check(runtimeContext);

log.WriteConsoleLog("Executing the load");
mover.Load(runtimeContext);

log.WriteConsoleLog("Load ok");
```

Figure 6. Usage of interfaces.

Even without the use of interfaces, the implementation would require classes with similar functionalities. Using interfaces improves code readability as it eliminates the need for repeated switch clauses for actions taken throughout the source code. Figure 6 displays the ease of changing between different interfaces.

3.3.3 Logging

Additionally, to writing the flow of the application and potential error messages to the console window, the application was designed to write external, more precise logs to the company database. This decision was made based on the requirement to be able to run the application as a part of automated scheduling. The database log creates a row in the log table using an SQL injection after the configuration has successfully been read and updates it with further information as it becomes available throughout the progression of the execution. Figure 7 shows the log table structure.



```
CREATE MULTISET TABLE NDP_STG.Pelican_Core_Log ,FALLBACK ,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT,
     DEFAULT MERGEBLOCKRATIO,
     MAP = TD_MAP1
     (
      load_id VARCHAR(64) CHARACTER SET UNICODE NOT CASESPECIFIC,
      status VARCHAR(64) CHARACTER SET UNICODE NOT CASESPECIFIC,
      start_tms TIMESTAMP(6) WITH TIME ZONE,
      end_tms TIMESTAMP(6) WITH TIME ZONE,
      move_start_tms TIMESTAMP(6) WITH TIME ZONE,
      move_duration FLOAT,
      total_duration FLOAT,
      processed_rows BIGINT,
      inserted_tms TIMESTAMP(6) WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP(6),
      update_tms TIMESTAMP(6) WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP(6),
      Ppln_Exec_Id INTEGER,
      os_user_name VARCHAR(128) CHARACTER SET UNICODE NOT CASESPECIFIC,
      os_version VARCHAR(64) CHARACTER SET UNICODE NOT CASESPECIFIC,
      source_type VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      source_description VARCHAR(512) CHARACTER SET UNICODE NOT CASESPECIFIC,
      source_size BIGINT,
      source_size_unit VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      source_last_modified_tms TIMESTAMP(0),
      target_type VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      target_description VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      teradata_user_id VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      target_database VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      target_table VARCHAR(256) CHARACTER SET UNICODE NOT CASESPECIFIC,
      config_file_path VARCHAR(512) CHARACTER SET UNICODE NOT CASESPECIFIC,
      config JSON(65536) INLINE LENGTH 128 CHARACTER SET UNICODE,
      pelican_version VARCHAR(64) CHARACTER SET UNICODE NOT CASESPECIFIC)
PRIMARY INDEX ( load_id );
```

Figure 7. Log table structure.

3.4 List of Supported Load Procedures

The scope of the thesis limited the application to work with the file formats pictured in table 2.

Source	**Target**
Azure Storage Tables	Teradata SQL Server
.JSON files	
.CSV files	
.XLS files	
.XLSX files	

Table 2. Supported Source and Target Types.

3.5 Handling of User Credentials

As the connections to the transfer protocols need authentication, the application needs to access user secrets such as usernames, passwords, and connection strings. Since the run-specific configuration files are stored in production servers, they are at a risk of being exposed to other users. It was therefore decided that such secrets should be stored as key-value pairs under a ‘secrets’ titled section under the application configuration. Storing credentials outside the run configuration also brings a degree of distributability, as users do not necessarily need to alter the run configuration to run the load. Run specific configuration would then contain the key based on which the secret value is retrieved during runtime. This approach is not without security vulnerabilities such as the possibility of accidentally uploading your secrets to version control or production server. Therefore, the handling of secrets is planned to be moved to Azure Key Vault in a future version.

3.6 End result

The result of the thesis work finished a version of the application with the ability to support five source file types and one target type. The testing on large scale adaptation is ongoing at the time of the submission and there are several use cases that run the application on a daily basis. The developers in the data warehouse have been briefed on the functionality and have been invited to utilise its functionality. The feedback has been positive and interest for current and future capabilities has been clear. The nature of the application dictates that future versions be developed as the need to cover more sources and targets arises. The application development has been expanded from the author to also include the data warehouse tool development team.

4 FURTHER DEVELOPMENT PLAN

4.1 General roadmap

The timeline of the application is at its beginning and there are plenty of ideas lined up for its future. The DWH unit has created a team to upkeep all the inhouse software in the unit and the development work has been expanded to it.

As one of the focus points of the development was for the structure to be extendible for further source and target types, this will be an integral part of the development plan. Nevertheless, initially the application will need to be spread to the DWH unit more widely. There are six solutions running on a weekly schedule using the application and the goal is to have this number higher to catch possible issues and bugs before the scope of the application gets too big to fix them. Furthermore, it will be useful for the developers to be familiar with using the application before implementing more functionality to not confuse them with too many options at the start. More users also means a better understanding of what the needs of the organisation for the application are.

4.2 Initial plan for expanding support

Although the initial focus will be on fixing any issues arising in further distribution of the application, some ambitions for expanding support have already risen. Most importantly fixed-width files are likely to be the first added file format support. The possibility of supporting SQL Server View and Tables is also under discussion.

Adding support for target types will prove more work compared source types since in an optimal situation every source and target type should work together. Currently there is only one supported target type so adding a second one would in theory require supporting it for all five sources. Nevertheless, being able to generate .CSV files as a target is a planned feature.

4.3 Features

Currently, an issue has been encountered where in some cases of unsuccessful execution, TPT Fastload will lock the target database table. Currently this requires manual release for the table and functionality to automatically resume failed jobs would reduce manual work.

Secondly, more flexibility in loading protocols would be useful in the long run. TPT Fastload has some limitations, namely only being able to load into an empty table, which make it too rigid for all file loads. Loading into already populated tables will make the application more flexible and TPT Mini-Batch Loading is being investigated for this purpose. However, some concerns regarding efficiency remain for the load mode.

5 RESULTS

The result of the thesis is an inhouse file loading application that has gone through proof of concept and functionally supports the loading of five different source types to a Teradata database. The application acts as an interface for multiple third-party data load tools, optimising procedures and simplifying development work on the data warehouse. This simplification arises from a major reduction on the need for developers in the DWH unit to know how to operate multiple file load tools. Additionally, building further support for new source formats will be focused on the application, providing a framework and reusable functions.

The architecture of the source code, written in C#, was designed in a way that enables expansion for more source and target types in the future as the needs of the data warehousing unit grow. Further work on these and other features is ongoing, signalling the relevance of the application.

At the time of writing, multiple solutions are already running on a weekly schedule using the application and it is in the process of being adopted more widely within the commissioning organisation. Numerous solutions are currently being developed using the application. Additionally, development has started on more inhouse tools that depend on the application for functionality.

The application has enabled multiple source types to be loaded into the database that were previously unavailable. Migration of workflows for certain data formats will also be moved to the application due to a greater ease of use over the current tools. Future similar migrations are also possible, enabling the reduction of technical dept via development of the application. Through enabling data load operations for common data formats, the application has also provided the means for smoother cooperation with other organisations for solutions that are already under development.

REFERENCES

Awiti, J., Vaisman, A.A. & Zimányi, E. 2020, "Design and implementation of ETL processes using BPMN and relational algebra", Data & Knowledge Engineering, vol. 129, pp. 101837.

Date, C. 2013, Relational Theory for Computer Professionals.

Hugh Darwen 2010, An Introduction to Relational Database Theory.

Katragadda, R., Tirumala, S. & Nandigam, D. 2015, "ETL tools for data warehousing : an empirical study of open source Talend Studio versus Microsoft SSIS ".

Library Of Congress , Recommended Formats Statement. Available: https://www.loc.gov/preservation/resources/rfs/data.html#databases [Accessed 2020, 9.12.].

Microsoft 2020, , Interface (C# Reference) . Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface [Accessed 2020, 12.12.].

Souibgui, M., Atigui, F., Zammali, S., Cherfi, S. & Yahia, S.B. 2019, "Data quality in ETL process: A preliminary study", Procedia Computer Science, vol. 159, pp. 676-687.

Teorey, T.J., Lightstone, S.S., Nadeau, T. & Jagadish, H.V. 2011, Database Modeling and Design : Logical Design, Elsevier Science & Technology, San Francisco.

Vassiliadis, P. 2009, "A Survey of Extract–Transform–Load Technology".