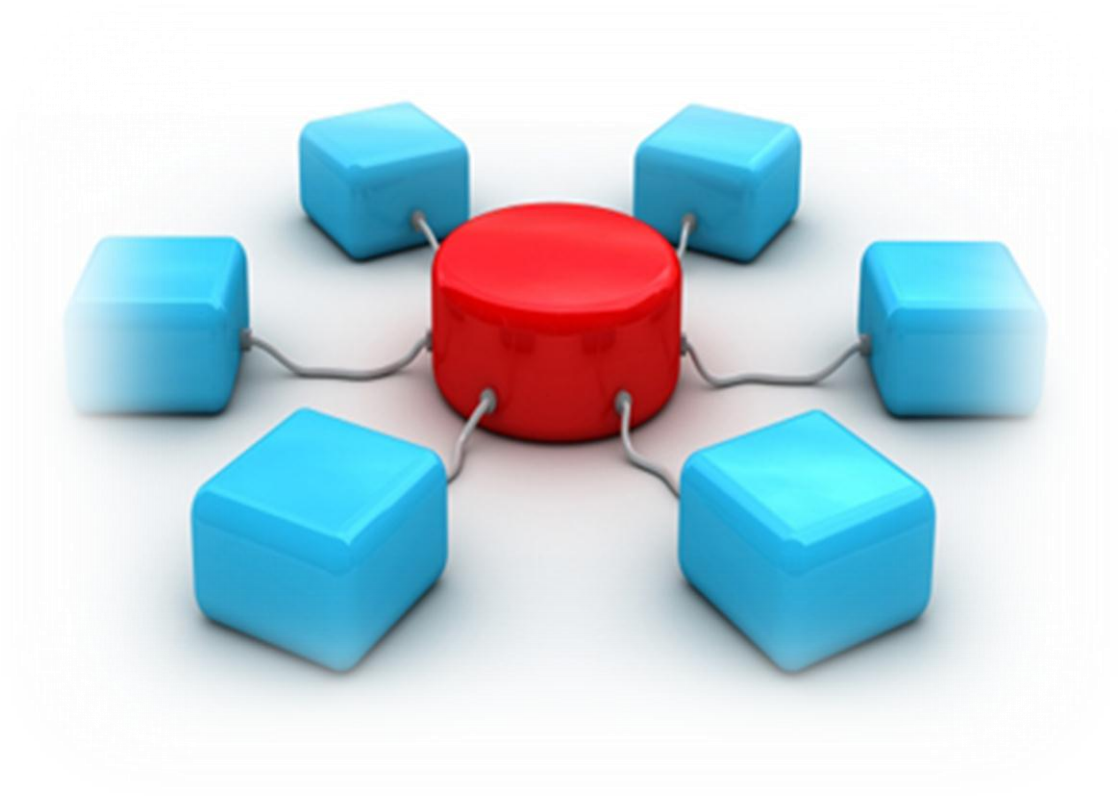
Bachelor of Technology

Information Technology Project

Data Integration



Shawn D'souza
sdso010@aucklanduni.ac.nz
University of Auckland
October 2012

Abstract

Data integration is a pervasive challenge faced in applications that need to query across multiple autonomous and heterogeneous data sources. My project for BTech this year is to design an interface that will allow users of the application to query and import data from various data sources. In this report I will describe my project, I will explore possible solutions to solve data integration issue and explain why my proposed solution is apt for this project.

Acknowledgements

I would thank Dr S. Manoharan for giving the opportunity to go ahead with this project. Aslo I would like to thank my Industrial mentor Frederik Dinkelaker for guidance and mentoring in developing the solution. I would also like to thank my academic supervisor, Dr. Xinfeng Ye for supporting with the research of my project. I would also like to thank the development team at torque for their support throughout the development process.

Contents

List of Figures

Introduction

Integration of application and business processes is a top priority for many enterprises today. Very few business applications are being developed or deployed without a major focus on integration, essentially making integratability a defining quality of enterprise applications (Hophe, 2002). For my project I will a add feature to the existing application to integrate data from other source into the application. In this report I will talk about what technique companies can use to integrate data from multiple sources. How information from multiple data sources can be queried. And what is the most efficient way of doing it.

Project Description

Company Information

The company I have chosen to work with for my BTech project is Torque IT Solutions. It is a start-up company that began in September 2011. Torque I.T.S is a finance company that primarily focuses on enabling car dealerships make the most profit from their business transactions, by providing them with the technical tools required to monitor and audit sales transactions in the company. The company has many software applications under development, and will be working on one of the applications, namely DPMS (Dealer performance management system). The application allows the Car dealer managers to manage the sales made within in the dealership, by allowing the staff to input transactional data into the system.

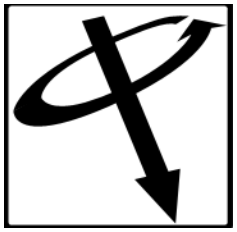


Figure 1: Torque ITS Logo

Problem Background

The Dealer Performance Management System is an application designed to allow mangers of car dealerships to identify their profit potential. This is done by allowing the business manager of the car dealership to monitor the car sales and services. This is done by the staff at dealership (may it be sales personal or finance mangers) to input their sales details into the application. The car dealership companies have their own point of sale and vehicle databases in which they store the information pertaining to the sales transactions and vehicle details. DPMS allows the business manager of a car dealership to monitor those sales made by the sale persons, finance managers, and aftersales staff. Before the business manager can make use of DPMS, he or the staff first need to enter the data into the system. This re-entry of data from the POS system is a major draw-back of DPMS. To resolve this issue, the application needs to be able to extract/ import the information from other information systems such as POS. For my project I will need design an application that will allow the user of DPMS to integrate the data from their Point of Sales system. Also users of the system might also want to include data from there on site vehicle database, or query vehicle information from the vehicle transport database. The application needs to be flexible and uniform so that data from any number for data sources can be queried and data can be imported with minimal code change.

Project Motivation

The purpose of the project is to reuse the existing data from external systems, as to reduce the time for data entry and eliminate human error. Allowing the option of user import their existing data into the system will make the application compatible with existing greatly increase the appeal of the system and increase the sales more car dealerships to buy the DMPS product. Having the form fields auto-populated from the data bases will eliminate any human error that could occur.

Project Goal

The goal of the project is to provide a uniform query interface to a multitude of data sources, thereby freeing the casual user from having to locate data sources, interact with each one in isolation and manually combine the results.

The project

I have been given the possible ways the data could be extracted. The data that needs to be imported into the system can come from several data sources and formats. It is up to me how I design the interface to allow for the flexibility to import any data source in a consistent format. However the data sources can be split into major categories based on the frequency of data updates. The first type of data source contains data that changes frequently (i.e. every hour); this data is stored on the point of sale system. The DPMS application needs to be able to query these changes as soon as they have been made. The second category is data whose value if changed will not affect the overall outcome the application. This data is not mission critical and doesn't need to be updated constantly

Project Specifications

An application need to be developed that will allow Users choose which databases, and applications to search the data from. The User of the system must be able to query all these data sources and obtain the results in a consistent format so that the data can be imported into the DMPS application.

Specifications

- Users must be given the choice for which external data source they want to pull data from
- Users must be able to import parts of a log entry (or specific columns of data), from any data source they desire.
- The web service will provide you with a number of methods you can call. The search parameters offered will be known.
- Each data source will offer different parameters
- The search parameter will be run over multiple data sources; some of the sources will not have the same parameters for the search.
- List of alternatives should appear in case of multiple results (i.e. two existing quotes in POS for same vehicle).

Project Research

Data Integration universal challenge faced in applications that needs to query across multiple autonomous and heterogeneous data sources. Today's reality is that a large percentage of a data warehouse's total cost of ownership (TCO) is related to post development integration costs—that is, the on-going costs of loading source data into the data warehouse and distributing data from the data warehouse to downstream data stores. In the IT industry we are often presented with the problem of having data sets ready to be used, and yet being unable to use them. This happens due to several facts: the coded information does not satisfy some of the actual needs or the resource format is not appropriate. These situations may lead to incompatibilities between the data used by two applications that perform the same kind of task, preventing the cross reusability of those data sets or even their combination to form a richer set of data.

For my project research I have explored possible solutions to the problem described. I will look what data integration means, what challenges are faced by businesses when they attempt to integrate data with external systems. I will briefly go through common data integration strategies and patterns used currently in the IT sector. Finally I will compare and contrast these patterns and validity to my project.

Before I dive into research on how data can efficiently extracted and transported to an application, I think it would be worth noting the other option which we can also come across if we were to own or have access over the counter part application. One reason to extract data and transport an external application is so that some data analysis can be carried out before the data is stored.

Live Data Reporting Strategy

The growing need for real-time or near real-time reporting outside of the line of business database; organizations are increasingly running some data integration processes more frequently—some close to real time.

With a Live Data strategy, the production POS data is reported on directly rather than reporting from a target Data Warehouse such as DPMS. Reporting and Business Intelligence software is used with Report Templates similar to those described in the Data Warehouse Strategy to deliver information to management when reporting using a Live Data strategy.

Implementing a Live Data strategy for reporting and business intelligence is the best approach for companies who want to empower people outside IT to develop and distribute reports to management without the technical skills, resources and budget required to implement and maintain a Data Warehouse.

Advantages:

- Less costly than a Data import strategy
- Less complicated than a Data import strategy

- "IT Lite" with much less reliance on IT resources than a Data import strategy
- Reports run against live production data rather than a Data Warehouse so you know all data returned in reports is guaranteed to be the most recent data.
- Corporate production data sources in all formats (DB2, SQL Server, Oracle, Access, Excel, etc.) can be reported against directly rather than transmitted to a Data Warehouse a required first step of configuring all the transformations to get the data into a Data Warehouse
- Report development can be performed by people outside IT
- Reports, dashboards, and queries can contain drilldowns and hyperlinks to information in any module or data sources outside DPMS.

Disadvantages:

- If you purge your production tables often you will have to copy them first if you want to report historical information with a Live Data strategy
- Report processing is shared with transactional processing on your production POS database

Data warehouse strategy

With a Data Warehouse strategy, data is extracted, transformed, and loaded into a reporting database configured as a Data Warehouse including the Static Metadata.

The five main elements of a Data Warehouse strategy are:

1. Your production Point of Sale (POS) database
2. A second database used as your target Data Warehouse
3. ETL software used to Extract Transform and Load data from POS to DPMS
4. Reporting and business intelligence software used to deliver information from your Data Warehouse to managers throughout your organization
5. Report Templates used to address Dynamic Metadata, calculations and functions not integrated in a Data Warehouse

Advantages

- Reports run against the Data Warehouse rather than your production database so your production database can be dedicated to transactional processing rather than reporting
- If you have multiple instances of DPMS or multiple POS's you can easily merge data from these multiple data sources into one target data store for consolidated reporting
- Reporting can be faster than Live Data systems
- Static Metadata is provided in the Data Warehouse

Disadvantages:

- Building or buying pre-built Data Warehouses is more expensive than a Live Data strategy
- "IT intensive" with heavy reliance on IT support
- Resource intensive to manage, maintain, and provide additional content on an on-going basis

- Dynamic Metadata is not addressed in the Data Warehouse so it will need to be addressed in another format like a report template or created in reports you develop
- Populating the Data Warehouse with the data necessary to address all possible questions requires that you identify up front every possible scenario that needs to be addressed so you are sure the information is stored in the Data Warehouse
- The frequency of data being refreshed in the Data Warehouse may impact reporting. For instance, while in your month-end close process you post a journal entry and want to run a financial statement immediately thereafter, you need to ensure that the entry was refreshed in the Data Warehouse
- Requires additional database software to store data and ETL software to populate your Data Warehouse
- Reporting and business intelligence software needs to be implemented on top of the Data Warehouse
- When data is extracted from your production database it competes with transactional processing
- Because of this complexity the failure rate for Data Warehouse implementations is over 50%. (Gartner)

As you can see the pros of live reporting outweigh the cons, since with this project I don't have the option of implementing the preferred live reporting approach, I have will implement the data warehouse strategy. A key component in most data warehouse project is data integration. In fact, 75% of the initial Data Warehouse project commonly consists of data integration. (Source: Foundations of SQL Server 2005 Business Intelligence, Lynn Langit)

Approaches to Integration

In this section, we apply an architectural perspective to give an overview of the different ways to address the integration problem. The presented classification is based on [Dittrich and Jonscher, 1999] and distinguishes integration approaches according to the level of abstraction where integration is performed.

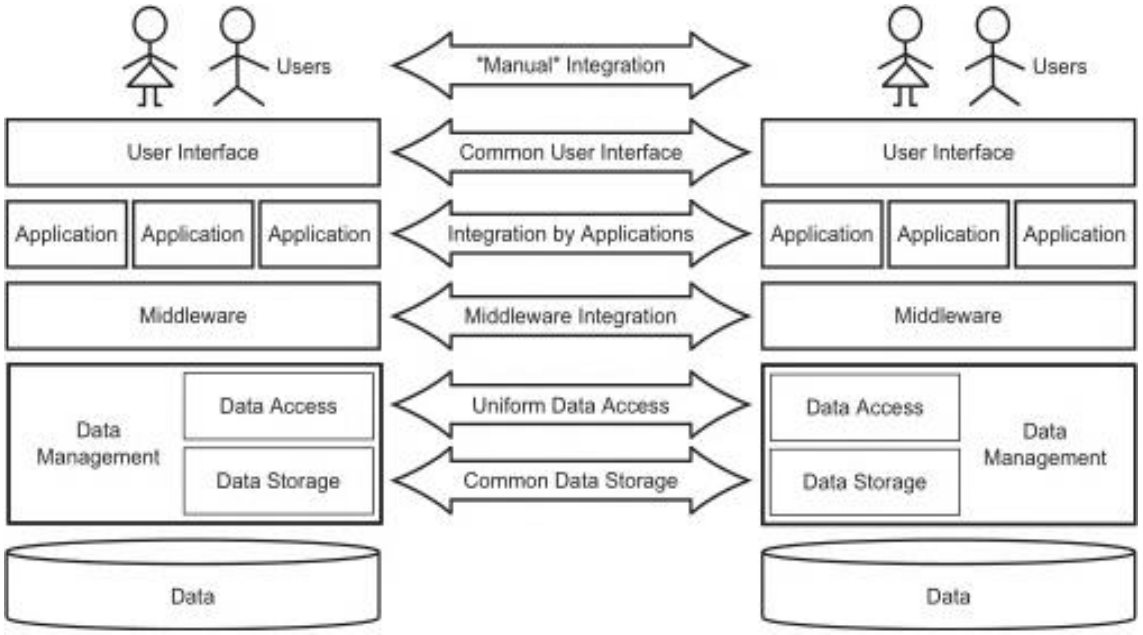


Figure 2: General Integration Approaches on Different Architectural Levels

Manual Integration

Here, users directly interact with all relevant information systems and manually integrate selected data. That is, users have to deal with different user interfaces and query languages. Additionally, users need to have detailed knowledge on location, logical data representation, and data semantics.

Common User Interface

In this case, the user is supplied with a common user interface (e.g., a web browser) that provides a uniform look and feel. Data from relevant information systems is still separately presented so that homogenization and integration of data yet has to be done by the users (for instance, as in search engines).

Integration by Applications

This approach uses integration applications that access various data sources and return integrated results to the user. This solution is practical for a small number of component systems. However, applications become increasingly fat as the number of system interfaces and data formats to homogenize and integrate grows.

Integration by Middleware

Middleware provides reusable functionality that is generally used to solve dedicated aspects of the integration problem, e.g., as done by SQL-middleware. While applications are relieved from implementing common integration functionality, integration efforts are still needed in applications. Additionally, different middleware tools usually have to be combined to build integrated systems.

Uniform Data Access

In this case, a logical integration of data is accomplished at the data access level. Global applications are provided with a unified global view of physically distributed data, though only virtual data is available on this level. However, global provision of physically integrated data can be time-consuming since data access, homogenization, and integration have to be done at runtime.

Common Data Storage

Here, physical data integration is performed by transferring data to a new data storage; local sources can either be retired or remain operational. In general, physical data integration provides fast data access. However, if local data sources are retired, applications that access them have to be migrated to the new data storage as well. In case local data sources remain operational, periodical refreshing of the common data storage needs to be considered.

In practice, concrete integration solutions are realized based on the presented six general integration approaches. Important examples include:

Mediated query systems represent a uniform data access solution by providing a single point for read-only querying access to various data sources. A mediator [Wiederhold, 1992] that contains a global query processor is employed to send subqueries to local data sources; returned local query results are then combined.

Portals as another form of uniform data access are personalized doorways to the internet or intranet where each user is provided with information tailored to his information needs. Usually, web mining is applied to determine user-profiles by click-stream analysis; that way, information the user might be interested in can be retrieved and presented.

Data warehouses realize a common data storage approach to integration. Data from several operational sources (on-line transaction processing systems, OLTP) are extracted, transformed, and loaded (ETL) into a data warehouse. Then, analysis, such as online analytical processing (OLAP), can be performed on cubes of integrated and aggregated data.

Operational data stores are a second example of common data storage. Here, a "warehouse with fresh data" is built by immediately propagating updates in local data sources to the data store. Thus, up-to-date integrated data is available for decision support. Unlike in data warehouses, data is neither cleansed nor aggregated nor are data histories supported.

Federated database systems (FDBMS) achieve a uniform data access solution by logically integrating data from underlying local DBMS. Federated database systems are fully-fledged DBMS; that is, they implement their own data model, support global queries, global transactions, and global access control. Usually, the five-level reference architecture by [Sheth and Larson, 1990] is employed for building FDBMS.

Workflow management systems (WFMS) allow to implement business processes where each single step is executed by a different application or user. Generally, WFMS support modelling,

execution, and maintenance of processes that are comprised of interactions between applications and human users. WFMS represent an integration-by-application approach.

Integration by web services performs integration through software components (i.e., web services) that support machine-to-machine interaction over a network by XML-based messages that are conveyed by internet protocols. Depending on their offered integration functionality, web services either represent a uniform data access approach or a common data access interface for later manual or application-based integration.

Peer-to-peer (P2P) integration is a decentralized approach to integration between distributed, autonomous peers where data can be mutually shared and integrated. P2P integration constitutes, depending on the provided integration functionality, either a uniform data access approach or a data access interface for subsequent manual or application-based integration.

Data integration is responsible for moving, cleansing and transforming set-based data—often very large data sets—from source(s) into the Production data area and then into the Consumption data area as shown in Figure 3.

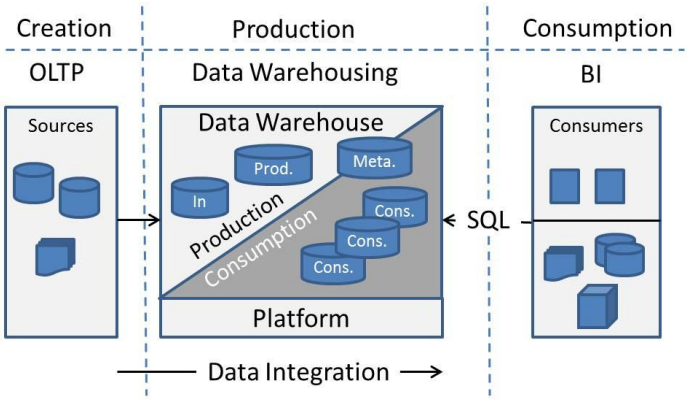


Figure 3: The role of data integration in a data warehouse project

Requirements for the data integration

The requirements for the data integration component include:

- **Trust** – Business consumers must be able to trust the results obtained from the data warehouse.
- **One version of the truth** – Consolidating heterogeneous sources into an integrated view supports business consumers' need for an enterprise-level view of data.
- **Current and historical views of data** – The ability to provide both a historical view of data as well as a recent view supports key business consumer activities such as trend analysis and predictive analysis.
- **Availability** – Data integration processes must not interfere with business consumers' ability to get results from the data warehouse.

Challenges for the data integration

The challenges for the data integration team in support of these requirements include:

- **Data quality** – The data integration team must promote data quality to a first-class citizen.
- **Transparency and auditability** – Even high-quality results will be questioned by business consumers. Providing complete transparency into how the data results were produced will be necessary to relieve business consumers' concerns around data quality.
- **Tracking history** – The ability to correctly report results at a particular period in time is an on-going challenge, particularly when there are adjustments to historical data.
- **Reducing processing times** – Efficiently processing very large volumes of data within ever shortening processing windows is an on-going challenge for the data integration team.

Data Integration Patterns

The industry has several well-known data integration patterns to meet these requirements and solve these challenges, and it's important for data warehouse practitioners to use the correct pattern for their implementation.

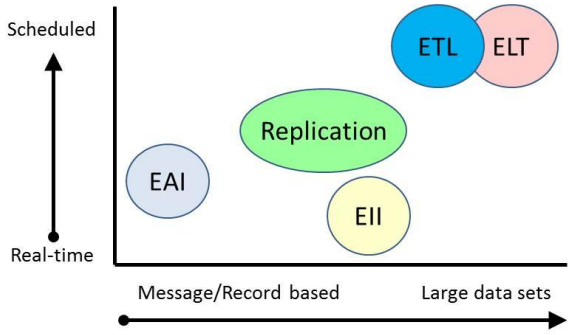


Figure 4: Positions the different integration options that are available.

The two axes in Figure 3 represent the main characteristics for classifying an integration pattern:

- **Timing –** Data integration can be a real-time operation or can occur on a scheduled basis.
- **Volumes –** Data integration can process one record at a time or data sets.

Integration patterns

The primary integration patterns are:

- **Enterprise Information Integration (EII) –** This pattern loosely couples multiple data stores by creating a semantic layer above the data stores and using industry-standard APIs such as ODBC, OLE-DB, and JDBC to access the data in real time.

- **Enterprise Application Integration (EAI)** – This pattern supports business processes and workflows that span multiple application systems. It typically works on a message-/event-based model and is not data-centric (i.e., it is parameter-based and does not pass more than one "record" at a time).
- **Extract, Transform, and Load (ETL)** – This pattern extracts data from sources, transforms the data in memory and then loads it into a destination.
- **Extract, Load, and Transform (ELT)** – This pattern first extracts data from sources and loads it into a relational database. The transformation is then performed within the relational database and not in memory.
- **Replication** – This is a relational database feature that detects changed records in a source and pushes the changed records to a destination or destinations. The destination is typically a mirror of the source, meaning that the data is not transformed on the way from source to destination.

Overnight Data Integration patterns

Data integration, which frequently deals with very large data sets, has traditionally been scheduled to run on a nightly basis during off hours. In this scenario, the following has held true for the different patterns:

- EII is not commonly used in data warehouses because of performance issues. The size and data volumes of data warehouses prohibit the real-time federation of diverse data stores, which is the technique employed by the EII pattern.
- EAI is not used in data warehouses because the volume of the data sets results in poor performance for message-/event-based applications.
- ETL is the most widely used integration pattern for data warehouses today.
- ELT is seen mostly in legacy data warehouse implementations and in very large data warehouse implementations where the data volumes exceed the memory required by the ETL pattern.
- Replication, used to extract data from sources, is used in conjunction with an ETL or ELT pattern for some data warehouse implementations.
    - The decision to use replication can be based on a variety of factors, including the lack of a last changed column or when direct access to source data is not allowed.

Which Pattern Should I Use?

Typically, a data warehouse should use either ETL or ELT to meet its data integration needs. The costs of maintaining replication, especially when re-synchronizing the replication process is required, makes it a less attractive alternative for extracting data from sources. However, hybrid approaches such as ETL/ELT combined with source system net-change detection capabilities may be required for near real-time data.

Data Integration Paradigms (ETL and ELT)

ETL products populate one or more destinations with data obtained from one or more sources. The simplest pattern is where one source loads one destination, as illustrated in Figure 4.

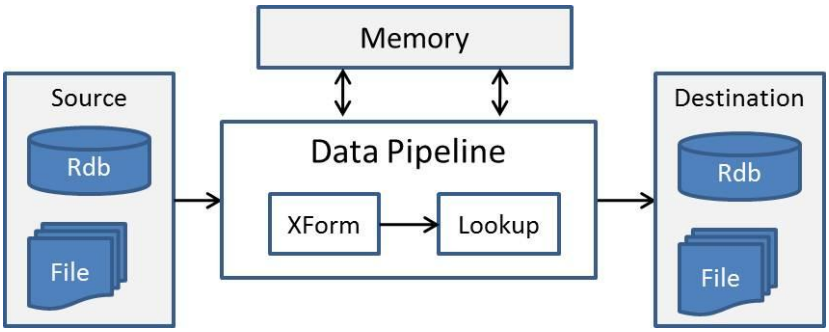


Figure 5: Simple ETL data flow

Processing steps in ETL data flow

The processing steps are as follows:

1. The ETL tool retrieves data sets from the source, using SQL for relational sources or another interface for file sources.
2. The data set enters the data pipeline, which applies transformations to the data one record at a time. Intermediate data results are stored in memory.
3. The transformed data is then persisted into the destination.

Advantages for ETL data flow

Advantages to this process are that:

- Procedural programming constructs support complex transformations.
- Storing intermediate results in memory is faster than persisting to disk.
- Inserts are efficiently processed using bulk-insert techniques.

Disadvantages for ETL data flow

However, the disadvantages include the following:

- Very large data sets could overwhelm the memory available to the data pipeline.
- Updates are more efficient using set-based processing—meaning using one SQL UPDATE statement for all records, not one UPDATE per each record.

Figure 5 shows an example of an SQL Server Integration Services (SSIS) data flow that performs transformation processes (joining, grouping, calculating metrics, and so on) in the pipeline. This data flow has the advantage of leveraging the memory resources of memory is limited or the data set needs to entirely fit in memory, the processes will the server and can perform many of the transformation tasks in parallel. However, when slow down.

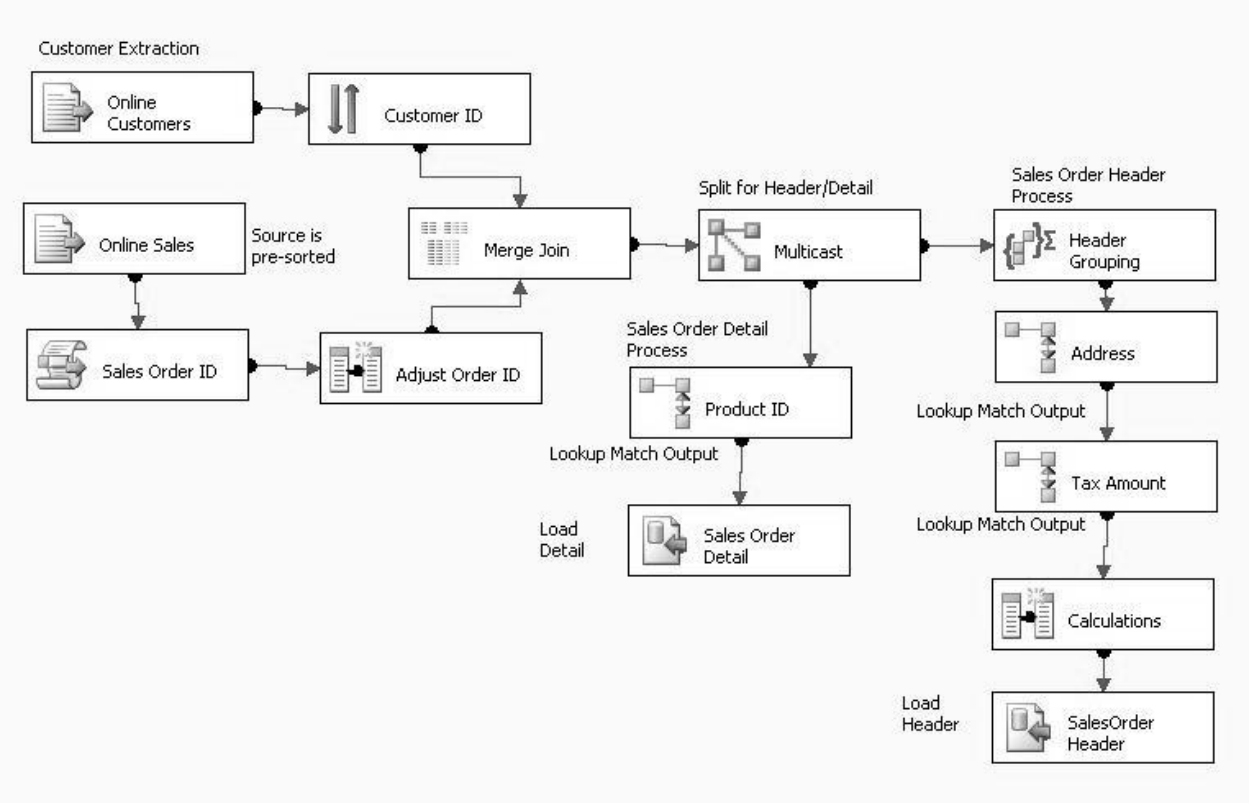


Figure 6: SSIS data flow example

Extract, Load, and Transform—also moves data from sources to destinations. ELT relies on the relational engine for its transformations. Figure 6 shows a simple example of ELT processing.

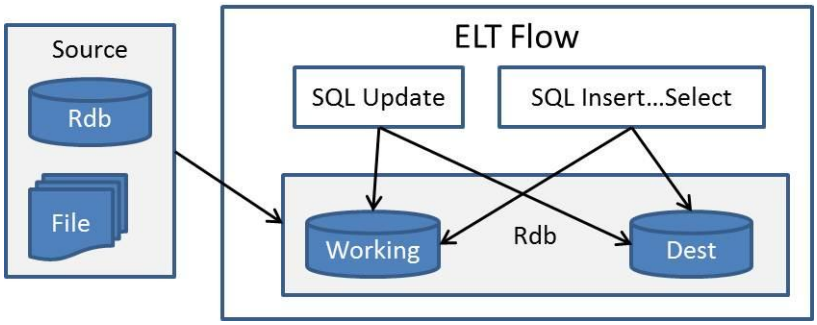


Figure 7: Simple ELT data flow

Processing steps in the ELT data flow

The processing steps in the ELT data flow are as follows:

1. Source data is loaded either directly into the destination or into an intermediate working table when more complex processing is required. Note that transformations can be implemented within the source SQL Select statement.
2. Transformations are optionally applied using the SQL Update command. More complex transformations may require multiple Updates for one table.
3. Transformations and Lookups are implemented within the SQL Insert…Select statement that loads the destination from the working area.
4. Updates for complex transformations and consolidations are then applied to the destination.

Advantages of ELT Data Flow

The advantages of this process include the following:

- The power of the relational database system can be utilized for very large data sets. Although, note that this processing will impact other activity within the relational database.
- SQL is a very mature language that translates into a greater pool of developers than ETL tools would.

Disadvantages of ELT Data Flow

However, you need to consider these disadvantages:

- As just noted, ELT places a greater load on the relational database system.
- You will also see more disk activity because all intermediate results are stored within a table, not memory.
- Implementing transformations and consolidations using one or more SQL Updates is more inefficient than the ETL equivalents, which make only one pass through the data and apply the changes to the destination using a single SQL statement rather than multiple ones.
- Complex transformations can exceed the capabilities of the SQL Insert and Updates statements because transformations occur at the record level not the data set level. When this occurs, SQL cursors are used to iterate over the data set, which results in decreased performance and hard-to-maintain SQL code.
- For a given transformation, the processes applied are often serialized in nature and add to the overall processing time.

Figure 7 shows the SSIS control flow used in more of an ELT-type operation. You can identify ELT-type operations by their multiple linear tasks, which perform either Execute SQL Tasks or straight data loads using a few working tables.

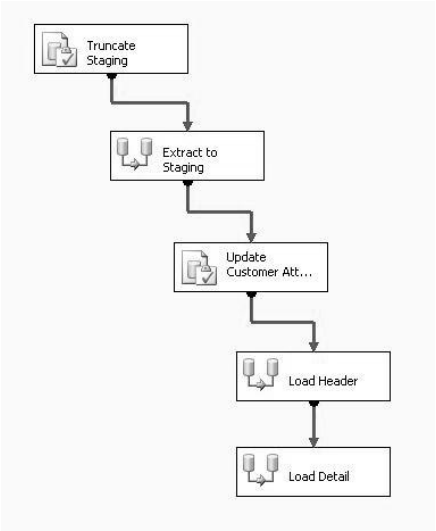


Figure 8: SSIS control flow ELT process

Which Should I Use for My Implementation?

The decision about whether to use an ETL or ELT pattern for a SQL Server data integration solution is based on the following considerations.

Use ETL when…

- Working with flat files and non-relational sources. ETL tools have readers which can access non-relational sources like flat files and XML files. ELT tools leverage the SQL language which requires that the data be first loaded into a relational database.
- This is a new data integration project or the current first-generation implementation is hard to manage and maintain. The visual workflows for tasks and data flows make the process easier to understand by non-developers.
- The transformations are complex. ETL's ability to apply complex transformations and business rules far exceeds the abilities of one set-based SQL statement. Many legacy ELT solutions have become unmanageable over time because of cursor-based logic and multiple Update operations used to implement complex transformations.

Use ELT when…

- The data volumes being processed are very large. Huge data sets may exhaust the available memory for an ETL approach. Remember that the ETL data pipeline uses in-memory buffers to hold intermediate data results.
- The source and destination data is on the same server and the transformations are very simple. A SQL-centric ELT solution is a reasonable choice when the current database development team is not trained on SSIS. But keep in mind that complex transformations can easily translate into poorly performing, unmaintainable data integration code.

Development Plan

Extracting the Data

There are two ways the data can be extracted from external systems. The car dealerships can offer a web service to query their point of sale of dealer management database. The second

option is that the dealerships, car manufactures, and other vehicle databases daily offer a database dump of the database. The option is preferred but however is not always possible for dealerships to create a web services to access their system due to the additional cost of development.

1. Ad-hoc Web Service

When a sales person in the car dealership makes a sale the transaction is stored in the POS. These kinds of changes need to be incorporated in the DPMS as soon as it happens. A solution to this is pulling the data from the external system via AD Hoc web service. In this approach the underlying P.O.S (Point of Sales) system will have a web service which returns data. For example when the user enter a registration number or VIN number the system will automatically trigger a web-service that will send a request to the external POS system, This query will then run on the systems database thus providing the user latest information. Which in turn will respond with corresponding vehicle detail, this data can then is used to populate the rest of the fields.

2. Using Data Dump

The second means of gathering data is through an overnight process where the external system in our case Auto link, which a data interface tool used by dealer to upload mass vehicle information to databases, this is same system currently being used by Trade me. The plan is use the data dump created by the vehicle database, Auto link to update our database. This means that all the latest vehicle details can be imported to into our system automatically without any user input; this eliminates any human error or data inaccuracy. In this case when the user enters an identifier the vehicle details will be looked up within the local data store and populate the remaining fields.

Figure 8 shows how the data from the different data sources will be extracted into the application.

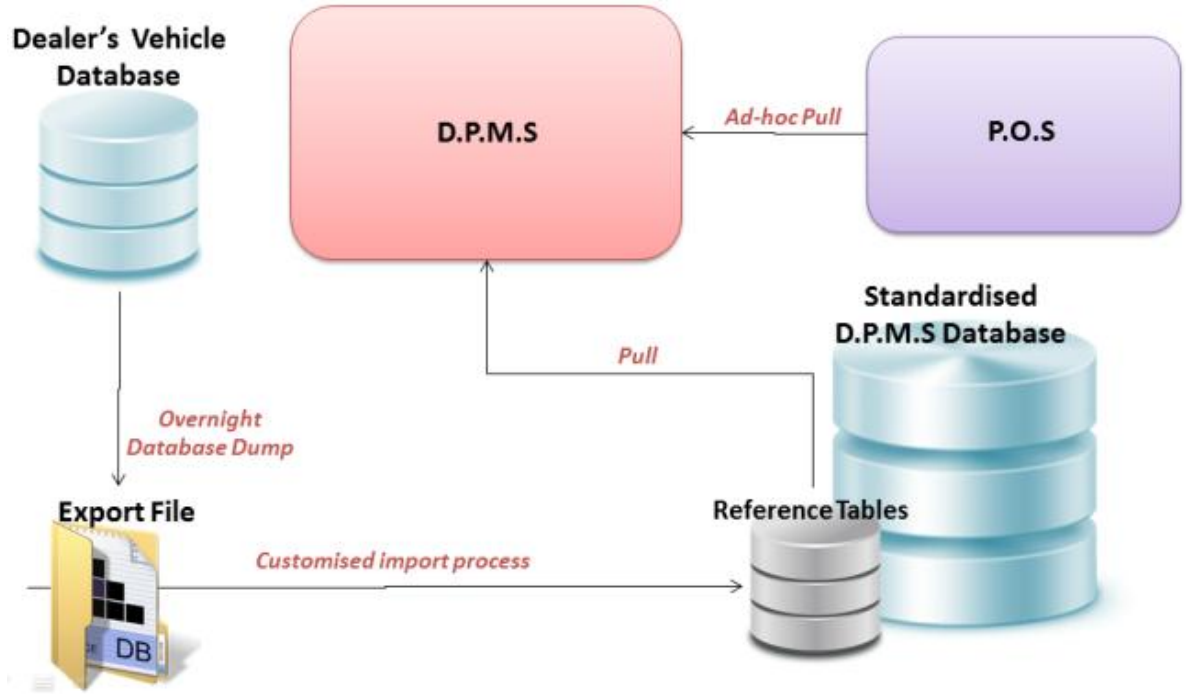


Figure 9: DPMS Data Integration Architecture

Import Interface Design

Design & Implement the Visual Interface

For the importing the data, I created a design of how the interface should look. The interface will allow the user to specify whether they want to import or manually enter the, Depending on the user configuration the system will use one of the two ways described above will used to retrieve the vehicle details.

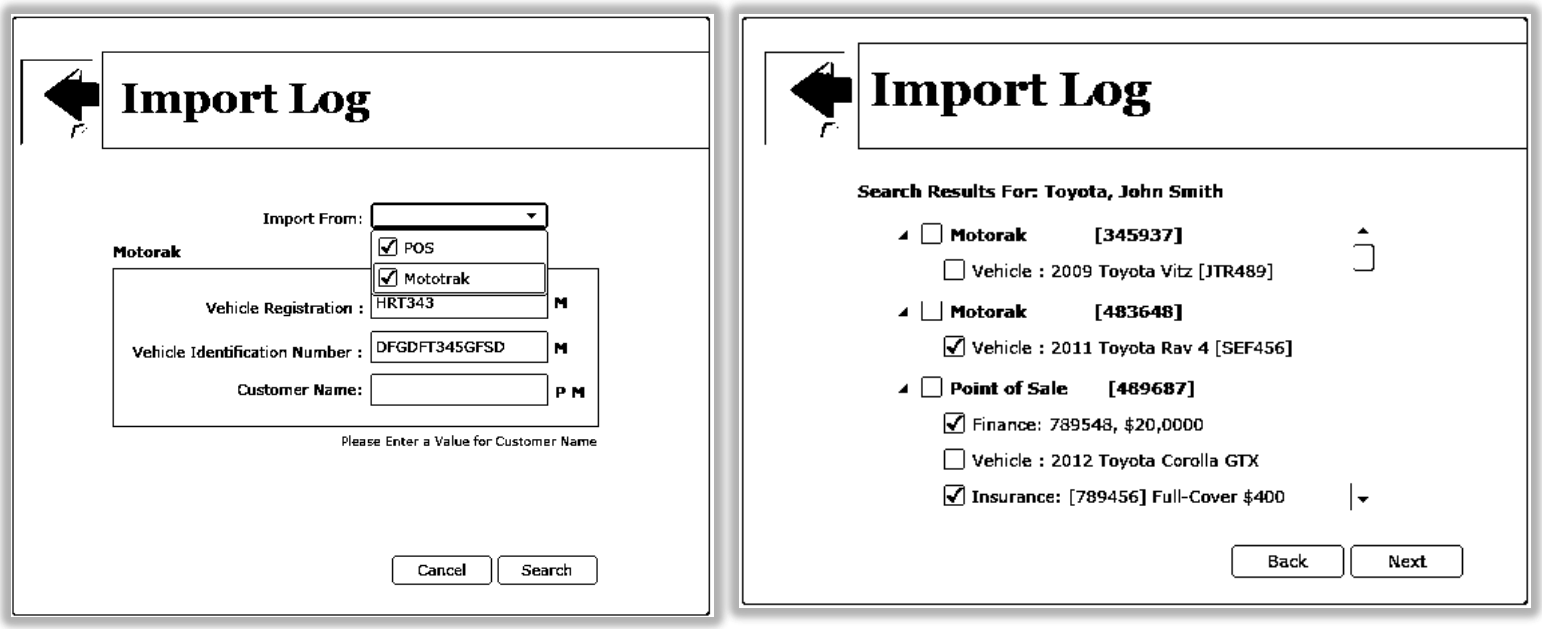


Figure 10: Import Interface: First two mock screens

Use Case 1: Creating and Editing Sales / Finance Entries

The user will create / Edit Entry they will be prompted with a popup, giving them the option whether to import data; the user will have the option to click cancel. If they choose to import (and most cases this should be the case), the user will go through a small wizard which will allow them to pick the entries they want and from where to pick the data from.

The user will pick the data sources they want to search from; they may choose either to search entries from e.g. Auto Link or POS or both.

Depending on what data sources they choose the identifiers' fields will be displayed*. The identifiers entered will be passed to POS web Service & Auto Link and cumulative results will be returned in a grid.

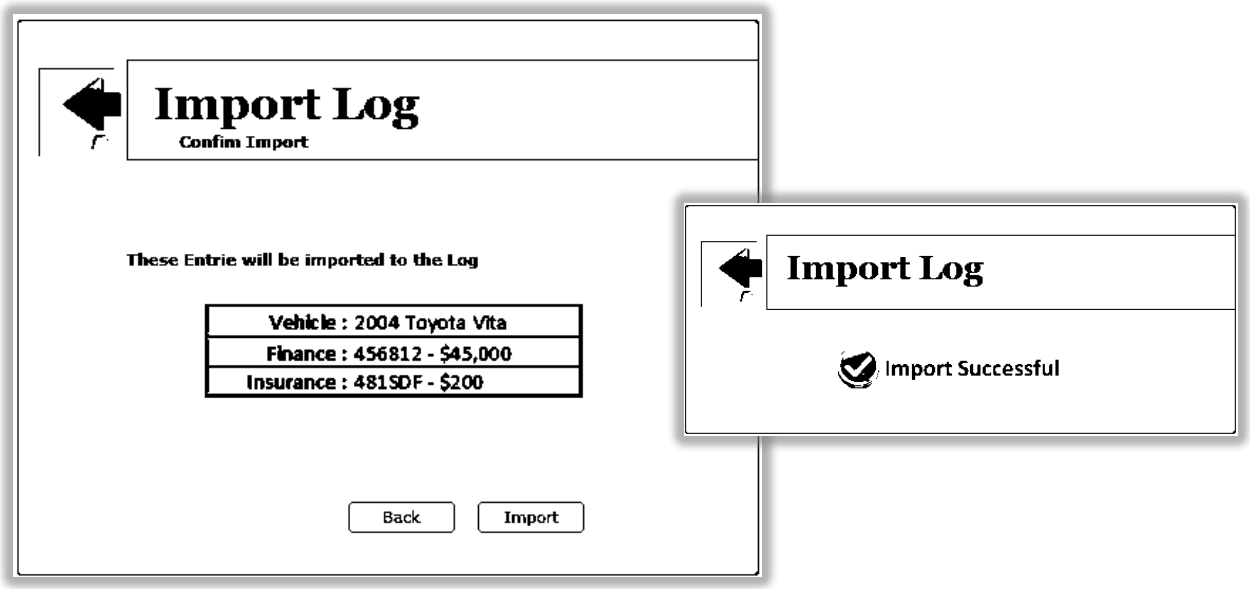


Figure 11: Import Interface design screens

1. From the tree the user will select one entry, the user has the option to select the whole entry or select sub entries that relate to the entry.(maybe even individual fields as a future improvement)
    a. The grid shows the data sources that returned matches to the query. Each data source has sub-entries that match the query. The user can only select one sub-entry, from the grid. For example, the user may only select vehicle information coming from Auto Link or the Point of Sales system. The user will not be allowed to select a vehicle entry coming from Point of Sales if a vehicle entry from Auto Link is already selected. Once the user is satisfied with their data import selection, they must click 'Import'.
2. In this screen they will get a final chance to go back and make any changes, before all the data is imported into the log entry Form.
3. If the import is successful, the users will be shown the screen on the right. The Screen on the right will appear as popup that disappears automatically after a second or so.
4. The selected entries will be imported into our current interface (i.e. fields on user screen will be populated according to imported data) where it can be further edited and saved. Only those input fields which are not disabled/read-only can be imported. Other values are ignored.

Technologies

Before I can start building a solution to this problem, it was vital that I understood the existing programming model used in the application. The early days of this project involved weeks of learning and research into the working of a web application. It was important that to understand the existing design patterns and structures in-place before I could start programming. In this section I will briefly go over some of the key design patterns, frameworks and plugins that I will encounter during my development.

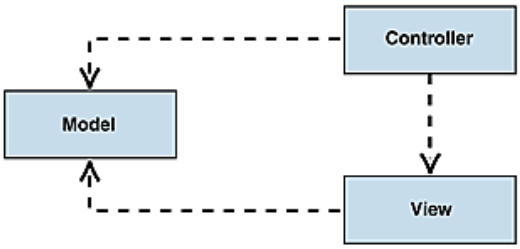
MVC

DPMS is a web application build using Microsoft ASP.Net Framework. The application itself is based around the MVC Programming model. Microsoft's MVC architecture separates the modelling of the domain, the presentation, and actions based on user input.

Model: The model manages the behaviour and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

View: The view manages the display of information.

Controller: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.



MVVM

Developing the user interface of a professional software application is not easy. It can be a murky blend of data, interaction design, visual design, connectivity, security, internationalization, validation, and a bit of luck. When designing the user interface, I had to keep in mind the underlying system was not exposed. Researching online I found various design patterns that claim to be the solution to maintain the code within your program. In the end I decided to follow the Model View View-Model (MVVM) design pattern as it natural add-on to the MVC framework. The main purpose of using this design pattern is to add structure to the code, to maintain the distinction between business objects and graphical user interface code.

The view model is a 'model of the view' meaning that it encapsulates the presentation logic and data for the view. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be rendered. View models are responsible for coordinating the view's interaction with any model

classes that are required. In DPMS there is a one-to many-relationship between the view model and the model classes. The view model is used to convert or manipulate model data so that it can be easily consumed by the view. Basically the view is a class that contains properties which can directly be displayed in view. The following is code the code snippet of the import view model.

```
public class ImportViewModel
{

    [Required]
    public IList<int> SelectedDataSourceIds { get; set; }

    /// <summary>
    /// The external Data sources available for the dealer
    /// </summary>
    public IList<ExternalLogImportSource> DataSources { get; set; }

    public IList<ExternalLogImportSource> SelectedDataSources { get; set; }

    public IEnumerable<ImportSearchParameter> SearchParameters { get; set; }

    public IEnumerable<ResultsViewModel> Results { get; set; }
```

ORM

The foundations of a data driven web application is database. Nhibernate is great Object-relation mapper (ORM) is designed to virtually wrap around the relational database. If you look at the name (ORM), it basically translates into: mapping relation tables to an object-orientated domain model. In DPMS each of the database tables are mapped (one-to-one) to code files using a mapping file. Using Nhibernate allows you to work with the database data as objects and program in the object-orientated fashion, which is familiar to almost all programmers.

All relational databases use data types for each of the fields, for example: int, small int, blob, char etc. Sometimes you have to convert the data types on the fly to properly add a record to the database. A good ORM such as NHibernate takes care of these details for you.

Traditional techniques of exchange between an object-oriented language and a relational database required you to declare a connection to the database ever time to want to access the database also you would need to be mindful to close the database session. In contrast Nhibernate abstracts away from all of this rote code and enables you to write actual business logic code faster and easier that before.

Using an ORM such Nhibernate create a consistent code all of the queries written will be in C#, therefore there is no SQL code to add complexity. This makes it easier to write and debug the program, especially if more than one programmer is on the job.

Security is always an issue in developing web applications, using an ORM shields the application from SQL injection attacks since the framework will be filtering the data.

Repositories

The Idea of repositories was new to me before I started work on the project. In enterprise application such as DPMS the repository pattern is a common construct to avoid duplication of data access logic throughout our application. The sole purpose of the repository is to hide the details of accessing the data. We can easily query the repository for data objects, without having to know how to provide things like a connection string. The repository behaves like a freely available in-memory data collection to which we can add, delete and update objects.

The Repository pattern adds a separation layer between the data and domain layers of an application. It also makes the data access parts of an application better testable.

The example below shows typical repository class using the Nhibernate ORM:

```
public class ImportRepository : NHibernateGenericRepository<ExternalLogImportSource>
{
    public ImportRepository() : base(null) { }

    public ImportRepository(ISession session)
        : base(session)...

    public IList<ExternalLogImportSource> GetExternalImportSourceForDealerId(int dealerId)
    {
        return session.Query<ExternalDataSourceDealerMap>().Where(m => m.Dealer.Id == dealerId)
            .Select(d => d.ExternalLogImportSource).ToList();
    }

    public IList<ExternalLogImportSource> GetExternalImportSourceForSystemClientId(int id)...

    public IList<SystemTenantImportSourceMap> GetSystemTenantImportSourceMapsForSystemClientId(int id)...
```

Figure 12: Snippet of Import Repository

The implementation of the repository is pretty straightforward. The class inherits from `NHibernateGenricRepository`, and the object type parameter is queried upon using lambda queries. The generic repository contains all the methods that can apply to any object such as `Insert, Delete, GetById and GetAll.`

JQuery

JQuery is a JavaScript framework, which purpose is to make it much easier to use JavaScript on your website. You could also describe jQuery as an abstraction layer, since it takes a lot of the functionality that you would have to write many lines of JavaScript to accomplish and wraps it into functions that you can call with a single line of code. JQuery does not replace JavaScript, and while it does offer some syntactical shortcuts, the code you write when you use jQuery is still JavaScript code.

JQuery tries to simplify a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation, so that you may do these things without knowing a lot about JavaScript.

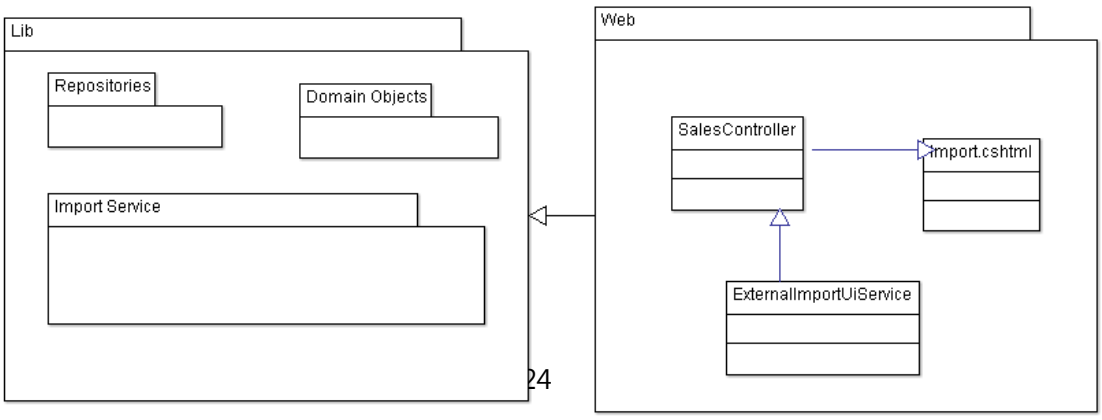
There are a bunch of other JavaScript frameworks out there, but as of right now, jQuery seems to be the most popular and also the most extendable, proved by the fact that you can find jQuery plugins for almost any task out there.

Implementation

To implement this external data import functionality feature there are a number of details I had to take under consideration. This section is a documentation of the thought process that led me the final solution. I will start by describe the steps I have taken to build the 3-tier application, starting from the data layer moving to the business logic and finally the presentation layer.

Project Structure

In my research and discovery phase I learn how MVC application is structured, and how different design patterns work together. Analysis of the existing DPMS project structure gave me clues as to how my solution will be laid out. The following diagram describes the overall structure of my implementation.



From this diagram we see the barebones of the architecture with still a lot of the class hidden for confidentiality. We can see that the web project depends on the libraries in the Lib project. The lib project will hold all the core application business logic. While the web project will be responsible for display the data to the browser. The import service library shown in the diagram contains the import interface as well as the concrete class implementations. The library also holds the supporting class that allow the domain object to be converted to view models.

Domain Model

To integrate external data into DPMS we need to convert the 'raw' data returned from the web service to domain object objects. The following is a snippet of existing DPMS Domain Structure:

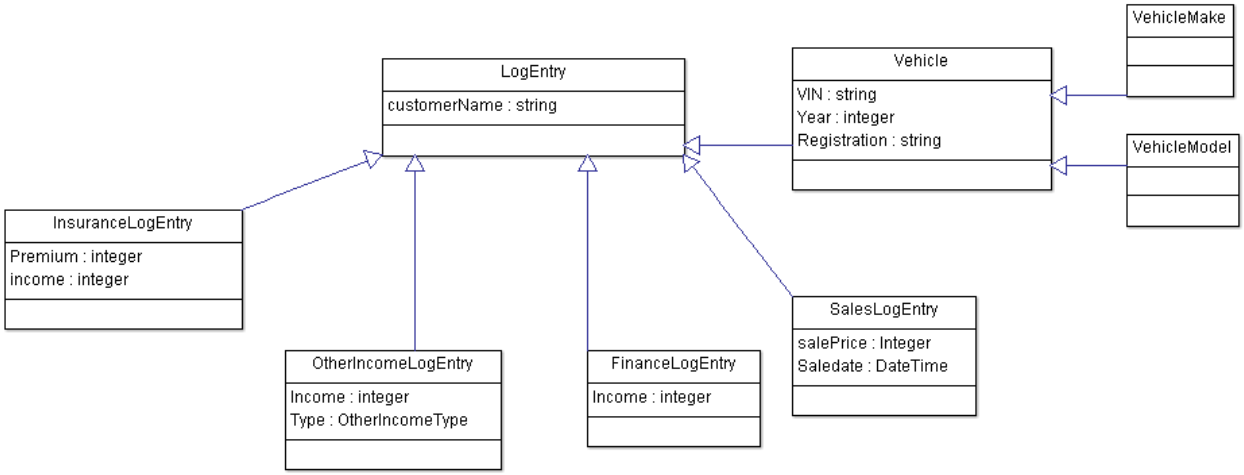


Figure 13: Skeleton LogEntry Domain Model

One of the requirements of the interface was to allow sub sections of a Log entry to be imported individually. From the figure 13 we can see that a 'LogEntry' consists of several other domain objects. To enable the select of sub section a view model structure was required.

External Import Data Source

To store the data about each external import data source, the following domain model structure was used to store information about the external data sources.

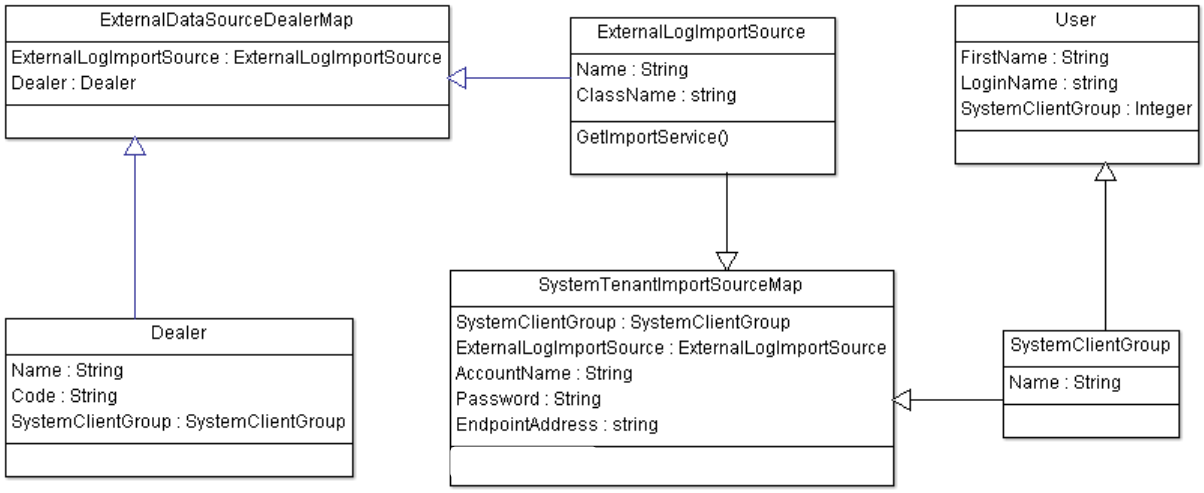


Figure 14: External Import source table structure

The diagram shows two table associations `ExternDataSourceDealerMap` and `SystemTenantImportSourceMap`. These objects provide a relation between `ExternalLogImportSource` and the Dealer and used when permission checking is done.

The next step is to do implement the interface that will allow multiple data sources to be associated to the system tenant. A common interface for retrieving all associated data sources for a user is required. The following class diagram outlines how each external Import source can be easily added.

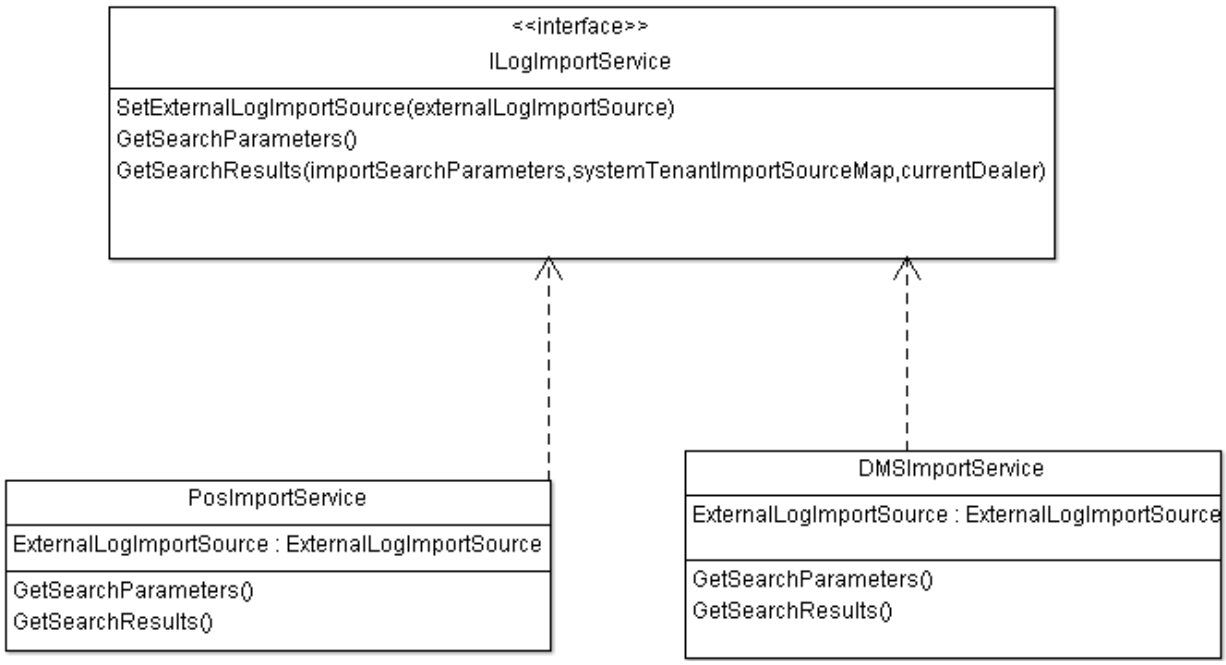


Figure 15: Common Interface to access the Import sources

The above `ILogImportService` interface is used to access methods of each of the external import sources.

Each external import source that uses a web service to pull data, requires a way to parse the data into local domain object. In my current version of the application the `PosImportService` class receives the data returned from the web service as a response. The response returns a quote data object. In the GetSearchResults method, the dataset goes through a forloop in which each of the objects is converted to object in the DPMS domain model. The following code shows how the data in each result returned is converted.

```
VehicleMake vehicleMake = vehicleMakeRepo.GetVehicleMakeByName(quoteData.makeName);

// if vehicleMake is not null, then copy it over
if (vehicleMake != null)
{
    vehicle.VehicleMake = vehicleMake;
    VehicleModel vehicleModel = vehicleModelRepo.GetVehicleModelByName(currentDealer.DealerGroup.Id, vehicleMake.Id, quoteData.modelName);
    vehicle.VehicleModel = vehicleModel;
}
vehicle.SubModel = string.Format("{0} {1}", quoteData.modelName, quoteData.subModel).Trim();
result.LogEntry.Vehicle = vehicle;
```

From the code above we see that how the 'raw' data string of vehicle make is converted into a `VehicleMake` object. To create a `VehicleMake` object the local database is queried via the repository (`VehicleMakeRepo`). In a similar way other raw data such as sale price, interest rate, customer name, etc can be converted into SalesLogEntry, FinanceLogEntry, and Customer domain objects. Once the response from the web service is converted to a domain object it is returned to the `ImportExternalLogEntryUiService` class where it converted to a view for display.

UI service class

The `ImportExternalLogEntryUiService class` in an extension of the controller, when a GET request to load the import window is made the method call is handled by the controller. When the controller is called a method in the UI service class is called where all the 'heavy lifting' is done.

To load the view of the import interface, permission checking is done to verify whether a user has any import sources configured. For each of the configured import sources the search parameter offered are collated and returned.

After the user fills the search form, the data is posted back to the server. The following snippet of code shows how the interface is used to combine the search results from each data source.

From the code snippet we see that the search parameters are passed into the '`GetSearchResults`' method of the interface, where the contrecte implementation is carried out in each of the import source classes.

```
public IList<ImportSearchParameter> GetParameters(IList<ExternalLogImportSource> datasources)
{
    List<ImportSearchParameter> finalSearchParameterList = new List<ImportSearchParameter>();
    foreach (ExternalLogImportSource externalDataSouce in datasources)
    {
        IList<ImportSearchParameter> externalImportParameterList = new List<ImportSearchParameter>();
        ILogImportService logImportService = externalDataSouce.GetImportService();
        externalImportParameterList = logImportService.GetSearchParameters(); //  could store the parameters in the session for use later

        //isp: import search parameter from each external data source
        //fp:  import search parameter in the final search parameter
        externalImportParameterList.ToList().ForEach((isp) =>
        {
            if (finalSearchParameterList.Where(p => p.Name == isp.Name && p.Type == isp.Type).Count() > 0)
                finalSearchParameterList.ForEach((fp) => { if (fp.Name == isp.Name)fp.ExternalImportSource.Add(externalDataSouce); });
            else
                finalSearchParameterList.Add(isp);
        });
    }
    return finalSearchParameterList;
}
```

In the `ImportExternalLogEntryUiService` class, the method 'returns a list of `LogEntryViewModels` based on the selected view models.

Displaying the Data

The user interface was created in the Razor view engine. The razor view engine allowed me to quickly integrate the server code with html mark-up, while writing the minimum amount of code.

```
var values = $("#import_form").serialize();
values += "&salesLogEntryTypeId=" + salesLogEntryType;

$.ajax({
    type: 'POST',
    url: '@Url.Action("ImportSearch", "SalesLog")',
    data: values,
    success: function (data, textStatus) {
        var window = $('#import_window').data('tWindow');
        window.content(data);
    },
    beforeSend: function (jqXHR, textStatus) {
        $('.import-window-content').html('<img class="importWindowLoading" src="@Url.CdnContent("~/Content/Images/loading_big.gif")" />');
    },
    error: function (jqXHR, textStatus, errorThrown) {
        alert(jqXHR, textStatus, errorThrown); //todo
    }
});
```

Input Validation

In a web application a user can practically post kind of input to the server. Not validating input can lead to system crashes, malicious data manipulation, and even database corruption.

A requirement of the interface was when a user imports a log entry, the user may select one of each sub entry to import. The following java script code demonstrates how I implemented this client side validation.

Outcome

Overall, I think this project has been a successful one. I have not been able to complete the import interface but I have also managed to complete using a robust and scalable solution.

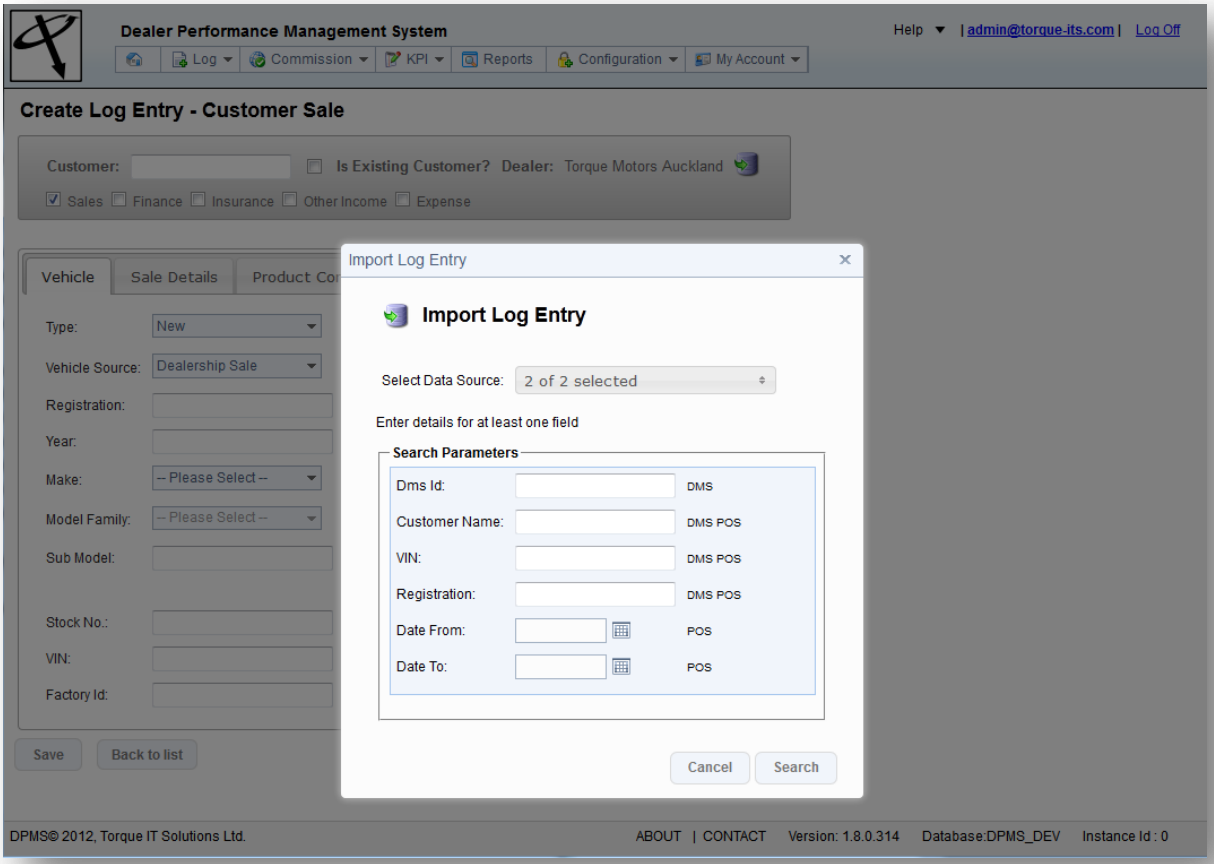


Figure 16: Import search form

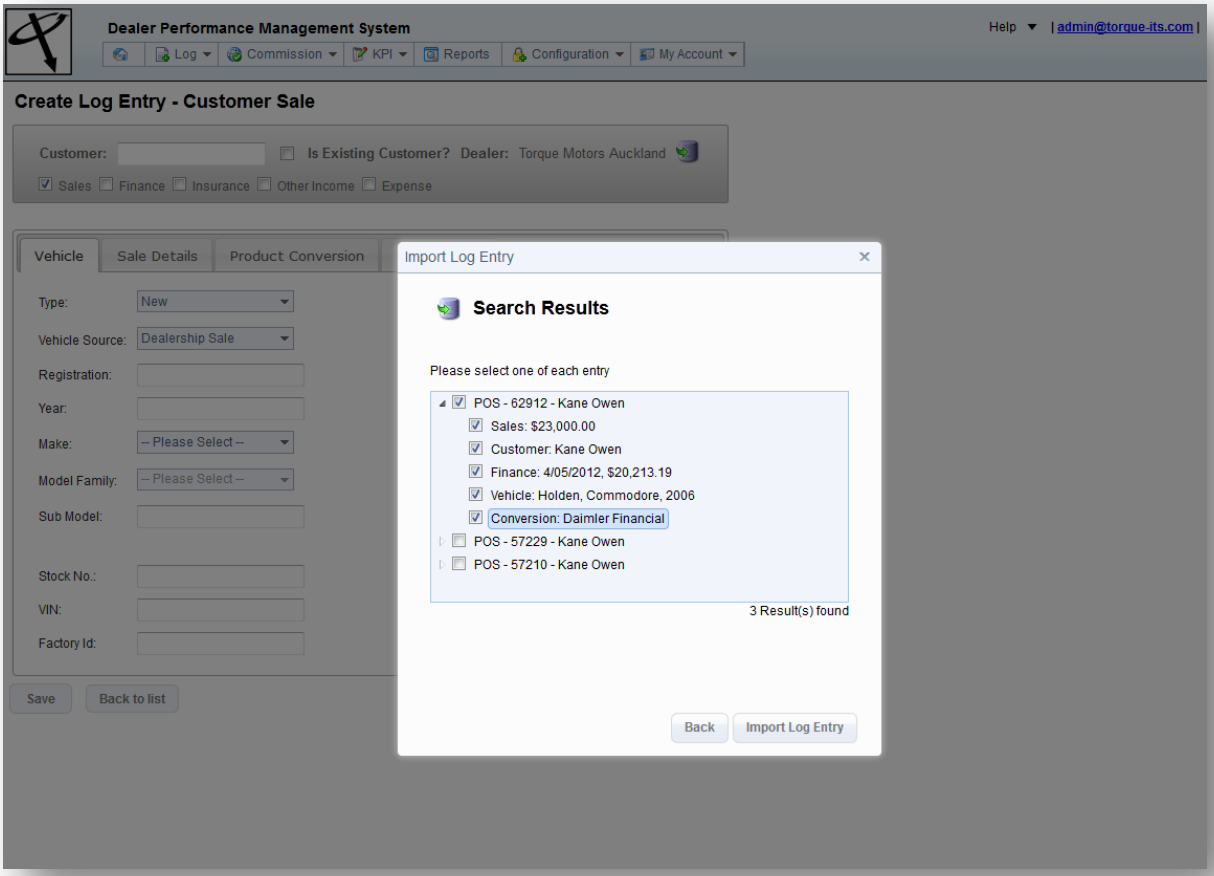


Figure 17: Import search results screen

Future Work

By completing my project, I have laid the foundations to Import data from other I.T. systems. Whether data is going to be imported via a web service or queried by reference tables, the `IExternalImportInterface` provides as central way to access all the data sources. To enhance the import functionality there are few improvements that can be made.

Currently, the vehicle data that is received from the web service gives a string which then search in the DPMS data by exactly matching the vehicle name or vehicle to model. If the data is stored in a different for example Merc instead for Mercedes, then the system would fail to return this vehicle. Although this is expected behaviour, as it meets the base requirements of the task. In future it would be nice to be able to identify the vehicle even though there minor spelling mistakes exists or data is expressed differently. A solution to this issue could be resolved by

setting up an intermediary table which holds a mapping between the commonly misspelled or different formats of expressing the same vehicle data. The mapping would point to the actual vehicle record stored in the database and return the appropriate vehicle object.

In the future, another 'nice to have' would to allow the user to import data only for specific fields. This can easily be implemented in the user interface by adding another level to the treeview structure. Aslo another view model class would be required to hold the data foreach of the select fields.

Achievements

This project given has given great insight the field of software development. I have not only gained practical real world experience but have also gained valuable technical knowledge. Through working with the company, I got an insight into the software development life cycle, learnt how to document and communicated my idea with peers.

Working with a company has taught me about discipline and planning.

Documentation

An important skill I learnt during my time working with the company was the need for and importance of documentation. Documentation not only allowed me to organise my thought but also aided my communication with my industrial mentor. Documenting my implementation as I have done in this report had helped me stick to the plan. The importance for documentation goes far beyond my personal benefits. This documentation contains information on the inner workings of implemented solution. This information makes it possible to reproduce the software or adapt it to maintenance. Although my documentation contributions had been small in the grand scheme of things, this kind of documentation will be built on and can worth millions to a company in the future.

Conclusion

Overall, my project has been a success. I have managed to complete my project and have gained a great insight about the challenges face by IT companies with regards to data integration. I've learn techniques to solve one of the most costly problems face by IT companies. Working with the development team has taught me the importance of communication to exchange ideas and refine the understanding of the problem space. One of the most important technical skills I learnt is the idea of design patterns. Before working with the company I was not familiar with patterns such as MVC, repositories, service locators etc. Working in the field has exposed me to how developers solve the problems in the real world.

Work Done

Date	Work
Week 1	Chose the project I will be working on this year
Week 2	This week I met with the company, we discussed about the company background, and a brief overview of the project
Week 3	Discussed the requirements of the project, and more details of the system were discussed
Week 4	The week I had my introductory presentation for BTech, which I discussed my project
Week 5	I worked the specifications document that outlined the scope and finalised the requirements of the project
Week 6	This week I was working the proposal the import interface and research some possible solutions for integration of data
Week 7	Been working on the proposal and more research was done with regards to MVC.NET
Week 8	This week I have been working on use cases
Week 9	I have started giving thought the UML diagrams required for the project
Week 10	Been working on the User interface designs for the import interface
Week 11	Started Implementing the Class diagrams & began work on the interface for the web service
Week 12	Working on Developing the interface and classes
Week 13	Conducted research on approaches and strategies to data Integration.
Week 14	Start working with NHibernate and creating domain model objects. Began to get familiar with MVC framework and Jquery
Week 15	Created the data tables & Domain Classes with the corresponding Nhibernate mapping files. Start work on the repository classes and methods
Week 16	Started work on the Import controller and the popup window. Wrote code to display a simple form
Week 17	Wrote the business rules to associate the configures import sources to the users, and limit access to the import interface based on user permissions
Week 18	Started implementing the *ILogImportService* class and worked on retrieving data from the local reference tables.
Week 19	Got the Wsdl for the POS web service. Connected to the web service and started the converting the returned to fit DPMS domain model.
Week 20	Worked on algorithm that will allow the domain object to be converted to view models uniformly
Week 21	Worked on the displaying the data on the view using Telerik UI controls
Week 22	Using Jquery wrote scripts to allow the user to select the entries that need to be imported.
Week 23	Worked the client side validation of user selection and form input.
Week 24	Conducted testing and User interface clean-up.
Week 25	Continued fixed bugs found in testing; also added finishing touches to the project.
Week 26	Conducted research on the scalability of the Web services and data Integration for large data volumes.
Week 27	Started working on Final presentation and Project report

Bibliography

(n.d.).

A Flexible Model for Data Integration. (n.d.). Retrieved from http://msdn.microsoft.com/en-us/library/bb245674.aspx

Enterprise Information Integration: A New Definition. (n.d.). Retrieved from http://www.information-management.com/news/1009669-1.html

Hophe, G. (2002). Enterprise Integration Patterns.

Ribeiro, R. (n.d.). How to Integrate data from different sources.

Understanding Enterprise Application Integration - The Benefits of ESB for EAI. (n.d.). Retrieved from http://www.mulesoft.org/enterprise-application-integration-eai-and-esb

Yin, R. (n.d.). An Effecient Data Service Layer.