Resolute Natural Resources Co. – Data Transfer Tool

Submitted by: Brandon Kraus, Ivo Yueh, Teddy Yueh

EXECUTIVE SUMMARY

Resolute Natural Resources Company is a natural gas and oil company that specializes in extracting resources from wells that other companies have already dismissed from being economically viable. Their accounting data for expenses on and off each site are stored in a series of databases. The primary goals of this summer field session project are to copy the data into a data warehouse and generate detailed reports via a graphical user interface.
The user interface should also allow users to send reports to the appropriate printer or file.

There will be two main databases that will provide the data, one from an accounting database and the other from a well activity database. However, the application should be modular enough to allow for simple database additions or be used for similar purposes in the future. The language for this project will be C# on the .NET 2.0 framework. Connectivity to the data will be via ODBC to Microsoft SQL Server 2005.

TABLE OF CONTENTS

1 REQUIREMENTS....1
1.1 Functional....1
1.2 Non – Functional....1
2 ANALYSIS (Use Cases)....2
2.1 User opens program....2
2.2 User initiates data transfer....2
2.3 User chooses to view report....3
2.4 User chooses to print report....3
2.5 User chooses to save report to file....4
3 Design....5
3.1 Modules....5
3.1.1 Database Connectivity Module....5
3.1.2 Data Transfer Module....6
3.1.3 Printing Module....7
3.1.4 Report Module....9
3.1.5 Saving Module....9
3.2 Class Outlines....10
3.3 UML Diagram....15
3.4 Database Schema....16
4 Testing....18

TABLE OF FIGURES

Figure 1 – Illustrates the ConnectionManager and ConnectionDetails classes with database connection............................5
Figure 2 – Basic flow of a data transfer...7
Figure 3 – Basic diagram of the print engine in custom printing mode..8
Figure 4 – Overall class UML diagram...15

1 REQUIREMENTS

1.1 Functional

1. Results of data transfers must be reportable and printable
2. The program must transfer data from multiple databases to a data warehouse (Decision Support database)
3. Data can be transferred at anytime by existing users
4. Transfers must be audited in both the data source and the data warehouse with a timestamp and responsible user.
5. The user must be prompted to commit each data transfer
6. The interface must display the current date and user
7. Program must authenticate user
8. The interface must inform user of progress and status via progress bar
9. ODBC's must be established when program initializes
10. Appropriate error messages must be displayed for connectivity problems

1.2 Non – Functional

1. Database transfers must be handled through a GUI
2. Data will be retrieved/ stored via OBBC in a Microsoft SQL Server 2005
3. The program must be written in C# .NET
4. The program must be a stand alone executable
5. Database transfers must be written as a stored procedure
6. The program must be delivered with full documentation on a CD / DVD
7. Multiple data transfers are allowed

2 ANALYSIS (Use Cases)

The following are the use cases that will be tested to determine whether the program meets its requirements.

2.1 User opens program

Description:	Initializes all necessary database connections and retrieves the necessary information to display to the user.
Precondition:	None
Postcondition:	Application opens

Primary Flow:

1. Program searches for user in active directory
2. a. Program finds user
3. a. Program displays current data and user
4. a. Program establishes ODBC to all involved databases

Alternate Flow:

1. b. Program fails to find user
2. b. Program closes

Alternate Flow:

1. c. Program fails to establish ODBC to one or more databases
2. c. Program displays appropriate error message

2.2 User initiates data transfer

Description:	The user uses the program to copy data from one database to a target database, which in our case is the DS database.
Precondition:	User has selected transaction period
Postcondition:	Respective data is transferred from data source(s) to data warehouse

Primary Flow:

1. User chooses to initiate data transfer
2. Program requests confirmation
3. a. User confirms data transfer
4. a. Program displays transfer progress
5. a. Program uploads data from respective source to data warehouse
6. a. Database records timestamp and user
7. Program notifies user of results

Alternate Flow:

3. b. User cancels data transfer
4. b. Program aborts data transfer

2.3 User chooses to view report

Description: Once a transfer has been completed, the user is able to view what was transferred by selecting a specific transaction. A history of transactions will be selectable to view.

Precondition: Data transfers have successfully completed
Postcondition: User views report

1. User chooses to view results of data transfer on screen (report) with a specified time period
2. Program generates report
3. Program displays report on screen

2.4 User chooses to print report

Description: Once a user has opened a report to view, the user can then print the report to a networked printer.

Precondition: Report has been generated
Postcondition: Report sent to printer

Primary Flow:
1. User chooses to print report
2. Program displays print properties
3. a. User confirms properties
4. a. Program sends report to printer

Alternate Flow:
3. b. User changes properties
4. b. User confirms properties
5 Program sends report to printer

2.5 User chooses to save report to file

Description:	Once a user has opened a report to view, the user can save the report as a file.
Precondition:	Report has been generated
Postcondition:	Report saved to file

Primary Flow:

1. User chooses to save report
2. Program displays save dialog
3. a. User confirms save
4. a. User selects path
5. a. User provides filename
6. a. Program saves file
7. a. Program notifies user of save status
8. a. Save dialog closes

Alternate Flow:

3. b. User cancels
4. b. Save dialog closes

Alternate Flow:

User can cancel at any time up to step 6 and the save dialog will close

3 Design

3.1 Modules

3.1.1 Database Connectivity Module

Classes: ConnectionDetails, ConnectionManager

In order to easily allow the client to establish and organize any number of database connections, this module utilizes hash lists to manage connection details and respective connections. The connections list maintains individual connection instances that know how to establish themselves. See Figure 1.

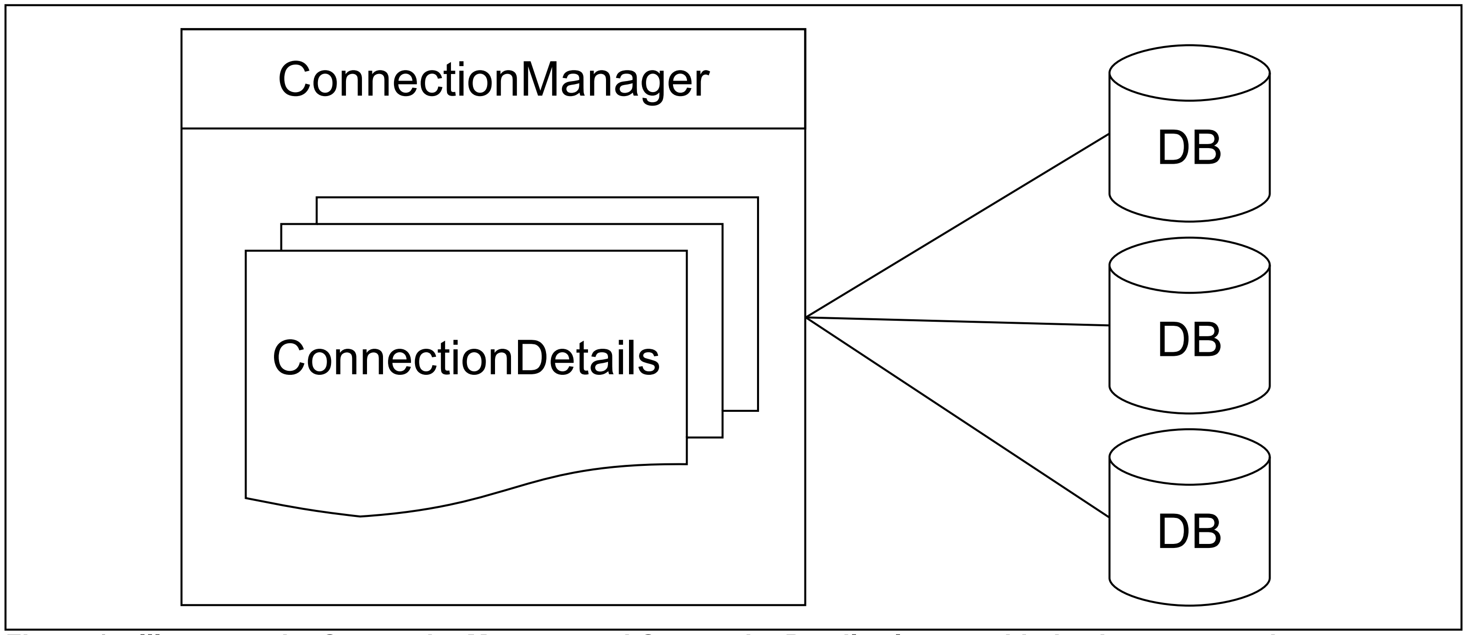

Figure 1 – Illustrates the ConnectionManager and ConnectionDetails classes with database connection

3.1.2 Data Transfer Module

Classes: AsyncSqlHelper, DataRepository, DataTransfer, FormProgress, StoredProcedures

This module handles all database activity. It is essentially a factory that creates interfaces to databases via stored procedures. Any SQL code implemented in the application will be from the data transfer module. The data transfer module will not only handle the transfer of data from one database to another. It is also responsible for the database activity that automates data retrieval when the application is first opened. Also handled in this module is the backend for the management tool that will be addressed in the *Action Plan/Scope* portion of this report.

***Deprecated; Windows SQL Server Express edition does not allow the dynamic insertion of table names in stored procedures. In light of this, the functionality of this class, as described above, has been currently dismissed.

This module has been redesigned in response to the lack of dynamic support of Windows SQL Server Express edition. It still maintains and manages database activity, but not through stored procedures held on a database.

The module now uses a collection of prepared statements held in StoredProcedures. The DataTransfer class uses the AsyncSqlHelper class to retrieve the data to be transferred on a separate thread so the user interface does not freeze during data retrieval. See Figure 2 below.

A progress bar accurately indicates the transfer progress based on the ratio of records transferred and total records to be transferred. Once the data has been retrieved, the progress bar increments 20% of its maximum size. The remaining 80% increment as data is actually transferred to the target database.

The module is generic in that databases details which feed the transfer methods are derived from a data repository class, DataRepository.

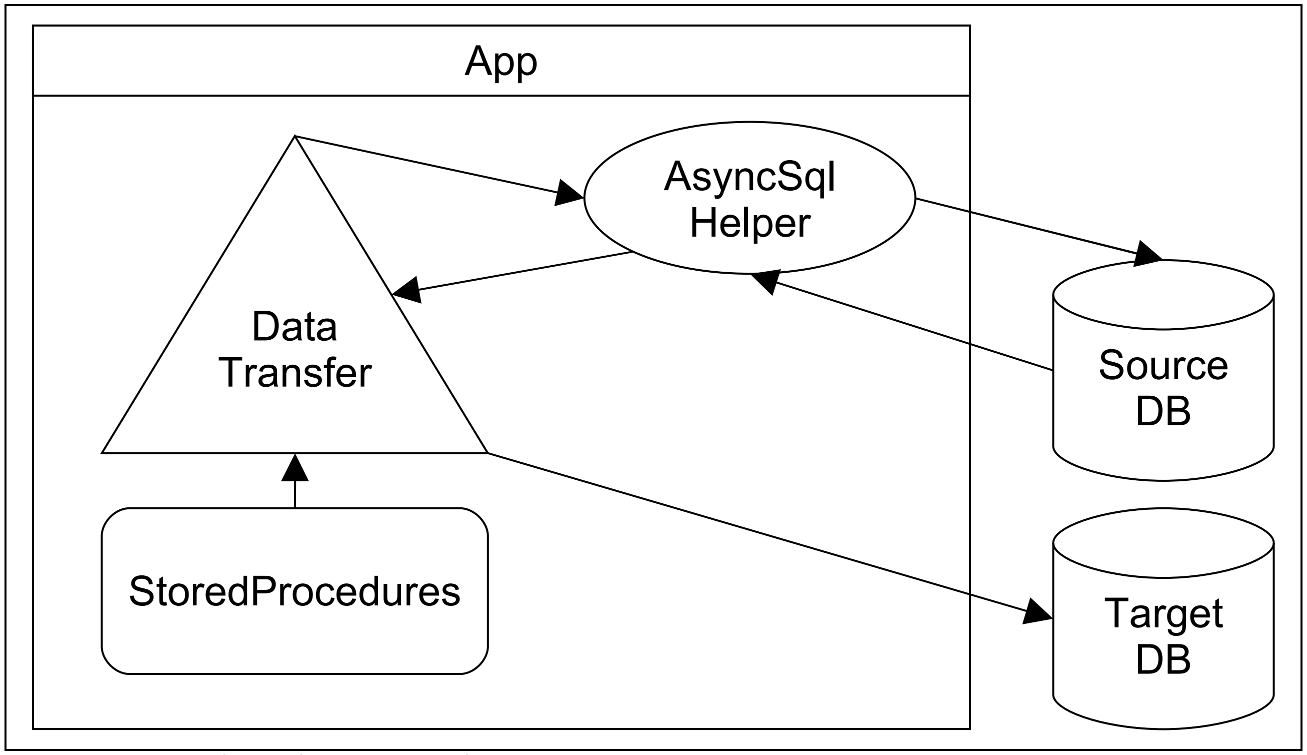

Figure 2 – Basic flow of a data transfer

3.1.3 Printing Module

Classes: IPrintable, IPrintPrimitive, PrintElement, PrintEngine
PrintPrimitiveRule, PrintPrimitiveText

The printing engine offers two methods of printing: control and custom. The control method is where the control seen on screen is printed. The control on screen is drawn to a bitmap in main memory and sent to the printer.

The custom method is driven by the idea that each object it is responsible for printing knows how to print or draw itself to the printer. Interface usage offers opportunities for future extendibility of the engine. The IPrintable interface enforces the assumption that each printable element knows how to print itself. Further, each printable element is composed of primitive objects such as lines. When an object is being printed, the PrintEngine allocates a PrintElement to the object to fill with IPrintPrimitives for object composition. See Figure 3 for a basic diagram of the custom print engine.

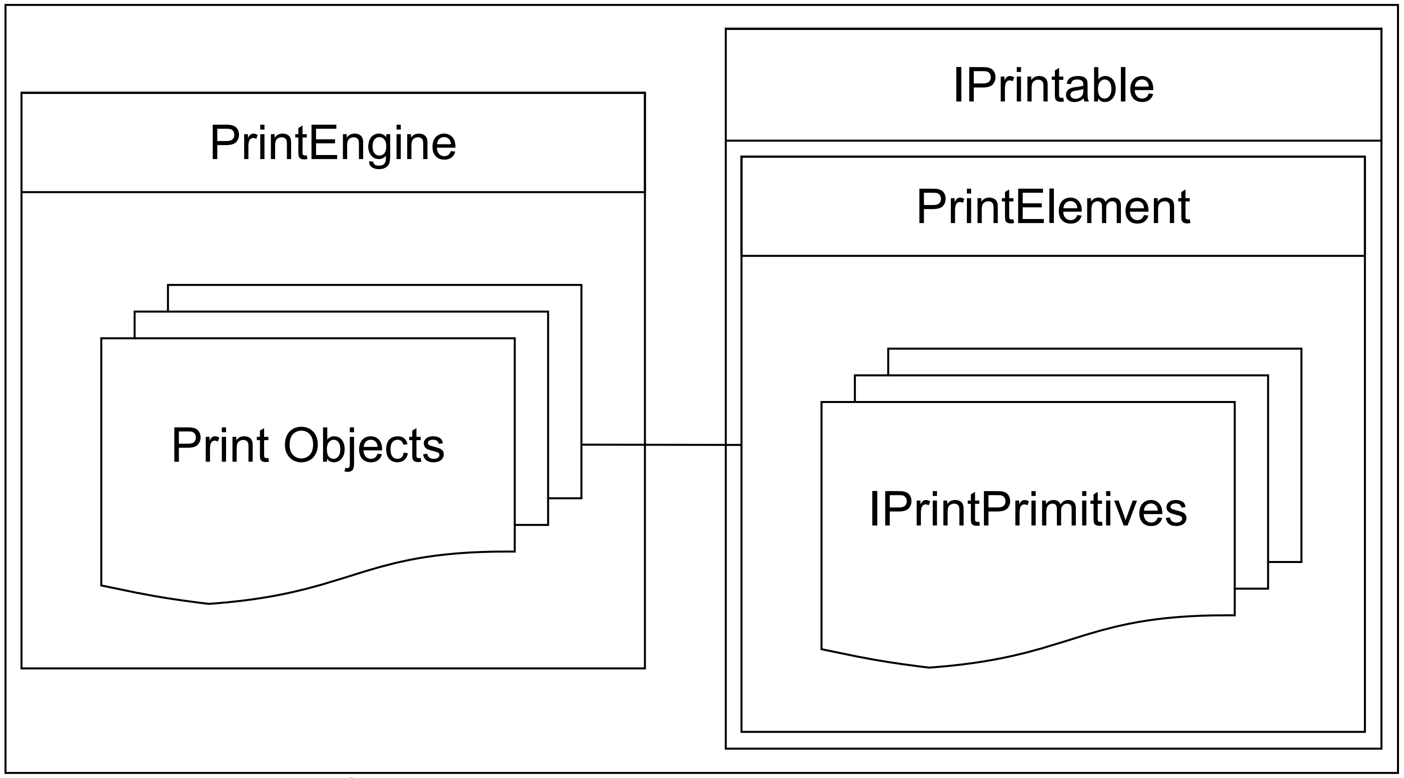

Figure 3 – Basic diagram of the print engine in custom printing mode.

3.1.4 Report Module

Classes: ReportDetails, ReportViewer

The report module is designed to be able to report information about any data transfer that took place, by using a separate table to record transfer information, which points to a foreign key in the primary database. The report class simply queries this secondary database to retrieve the transfer information, as well as the primary database to retrieve the information on the data that was actually transferred. It then stores this information to be reported either to the screen (GUI), printed to a file, or printed to an actual printer.

3.1.5 Saving Module

Classes: IHTMLSave, ReportSaver

The saving module is similar to the printing engine in that each object must know what format it wants to save itself in, and how it wants to print itself. For example, the Resolute transfer report is to be saved in HTML format, so it has to know how to print the tags, as well as the correct information in the correct format to stay consistent with the printed and viewed reports. First, a list of reports is added into the report saver class, and then when the user sends the signal to save it with a given filename, each object implementing the saving functionality will return an array of strings that represent the elements of the report.

3.2 Class Outlines

The various classes and their data member and method details are displayed below.

<u>AsyncSqlHelper</u>

Data Members

-delegate IDataReader ExecuteStoredProdecureReaderDelegate

Methods

-void AssignDataSourceDelegate
-void AssignDataSource
-void BackgroundWorker_LoadView
-IDataReader ExecuteReaderDelegate
-IDataReader ExecuteReader
-IAsyncResult BeginExecuteReader
-IDataReader EndExecuteReader
-IDataReader ExecuteStoredProdecureReader
-IAsyncResult BeginExecuteStoredProcedure
-IDataReader EndExecuteStoredProcedureReader
-void FillDataTableDelegate
-void FillDataTable
-IAsyncResult BeginFillTable
-void EndFillDataTable

ConnectionDetails
Data Members
-int timeout
-string database
-string password
-string server
-string table
-string user
Methods
-string connectionString

ConnectionManager
Data Members
-ConnectionDetails details
-HashList<ConnectionDetails> connectionDetails
-HashList<SqlConnection> connections
-SqlCommand command
Methods
-EstablishConnection(string connection)
-EstablishConnections
-getConnectionDetails

DataHelper
Data Members
-None
Methods
-DataTable GetSelectedTable

DataRepository

Data Members
-string primaryConnection
-string searchConnection
-string primaryStoredProcedure
-string searchStoredProcedure
-string transferDataStoredProcedure
-string TransferTable
-string ID
-string DatabaseName
-string User
-string BeginDate
-string EndDate
-string TransferStart
-string TransferEnd
Methods
-SqlCommand PrimaryCommand
-SqlCommand SearchCommand
-createStoredProcedure

DataTransfer
Data Members

-EventHandler DataIterated
-EventHandler TransferComplete
-EventHandler SourceDataReceived
-DataTable sourceDataTable
-DateTime beginDate
-DateTime endDate
-DateTime transferStart
-DateTime transferEnd
-int currentRecord
-int numProcessIterates
-int transferID
-int totalRecords
-SqlConnection getConnection
-SqlConnection insertConnection
-SqlConnection storedProcedures
-string sourceDatabaseName
-string sourceTableName
-string targetTableName
-string transferTableName

Methods

-void OnDataIterated
-void OnTransferComplete
-void OnSourceDataReceived
-int NumberOfProcessIterates
-void Transfer
-void CompleteTransfer
-void EndFillDataCallback

<u>Form1</u>

Data Members

-Button btnReports
-Button btnTransfer
-ComboBox cmbDatabases
-ConnectionManager connectionManager
-Label lblBeginDate
-Label lblDate
-Label lblEndDate
-Label lblWelcome
-DateTimePicker dtpBeginDate
-DateTimePicker dtpEndDate
-ReportViewer reportViewer

Methods

-void Transfer
-void ViewReports

<u>HashList</u>

Data Members

-List<string> keys

Methods

-List<string> Keys
-int KeyIndex
-T this
-void Add
-void Clear
-bool Contains
-int IndexOf
-void Remove

<u>IHTMLSave</u>

Data Members

-None

Methods

- -string[] write

IPrintPrimitive
Data Members
- -None

Methods
- -CalculateHeight
- -Draw

IPrintable
Data Members
- -None

Methods
- -Print

IPrintPrimitive
Data Members
- -None

Methods
- -CalculateHeight
- -Draw

PrintElement
Data Members
- -ArrayList printPrimitives
- -IPrintable printObject

Methods
- -void AddBlankLine
- -void AddHeader
- -void AddHorizontalRule
- -void AddText
- -float CalculateHeight
- -void Draw

- -void Print

PrintPrimitiveRule
Data Members
- -int height

Methods
- -float CalculateHeight
- -void Draw

PrintPrimitiveText
Data Members
- -Font font
- -string text
- -StringFormat stringFormat

Methods
- -float CalculateHeight
- -void Draw

PrintEngine
Data Members
- -List<object> printObjects

Methods
- -OnBeginPrint

ReportDetails
Data Members
- -int id
- -string dbName
- -string user
- -DateTime beginDate

-DateTime endDate
-DateTime transferEnd
-DateTime transferStart

Methods

-void Print
-void String[] Write

<u>ReportSaver</u>

Data Members

-ArrayList reportList

Methods

-void Add
-void ClearReports
-void Write

<u>ReportViewer</u>

Data Members

-Button btnSearch
-ComboBox cmbDatabases
-DataGridView dgvResults
-DataTimePicker dtpBeginDate
-DateTimePicker dtpEndDate
-DateTimePicker dtpTransferStart
-DateTimePicker dtpTransferEnd
-Label lblDatabases
-Label lblBeginDate
-Label lblEndDate
-Label lblID
-Label lblTransferEnd
-Label lblTransferStart
-Label lblNoResults
-Textbox txtID

Methods

-void Search
-void PrepareReport
-void Print
-void Save

3.3 UML Diagram

Figure 4 below describes how the different classes in our program interact.

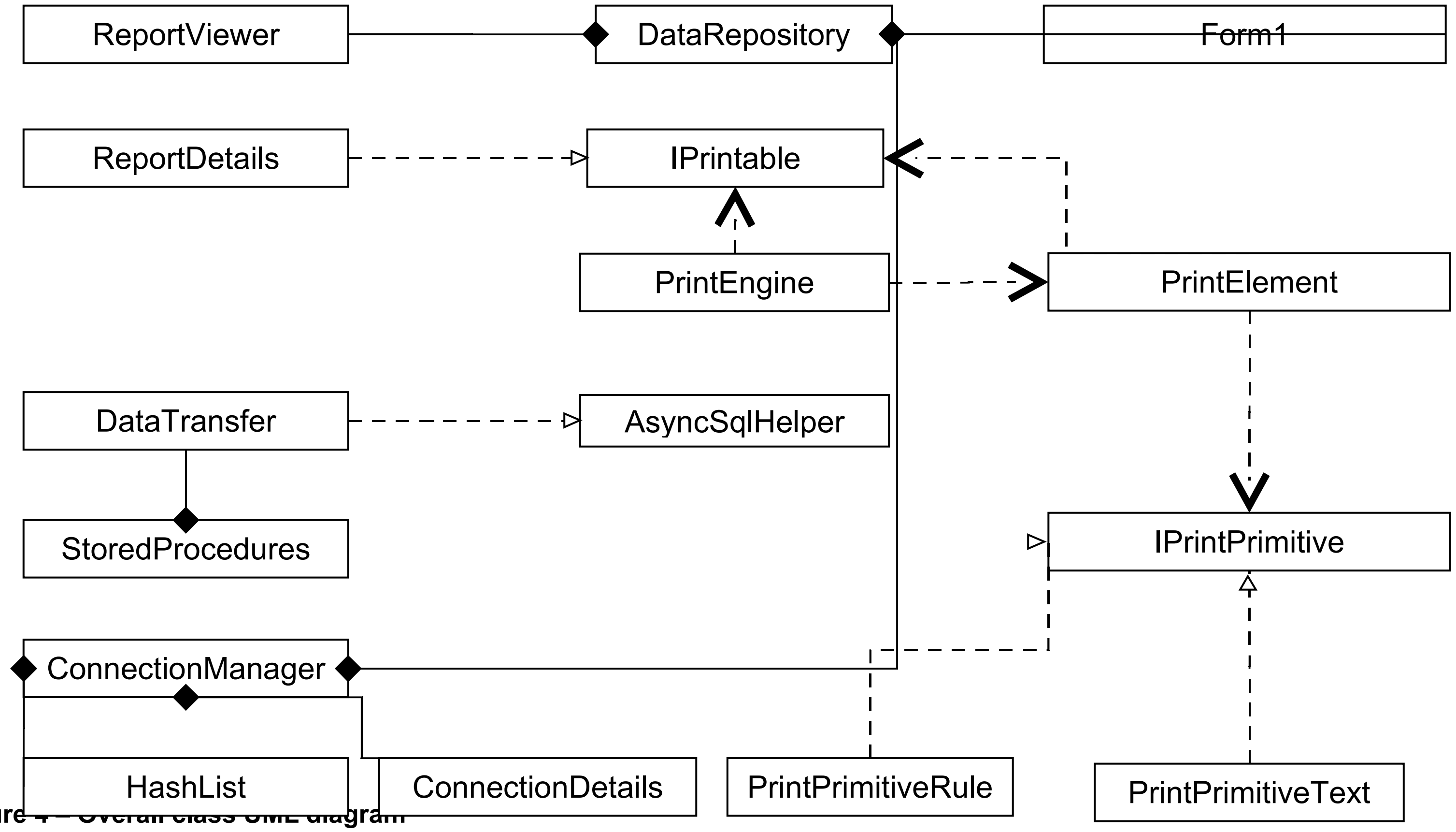

Figure 4 – Overall class UML diagram

3.4 Database Schema

The following is a schema of the back end of the program. The program pulls data from certain tables to determine permissions and the users and potential data sources, which are tables in other databases, involved during runtime.

<u>tblUsers</u>

UserName	The username used to login to the Windows XP operating system stored in Active Directory.
FName	First name of the user stored in Active Directory.
LName	Last name of the user stored in Active Directory.

<u>Connections</u>

Server	Server holding the data source database.
DatabaseName	Name of the data source database.
TableName	Table to be transferred in data source.
UserName	Username to log in to the data source.
Password	Password to log in to the data source.
Timeout	Number of seconds of inactivity before timeout from data source.
Type	Type of connection; ‘sqlexpress’, etc.

<u>tblStoredProcedures*</u>

SP_ID	Primary key
SP_Name	Name of stored procedure.

<u>tblSPParameters*</u>

SP_ID	Foreign key from tblStoredProcedures.
P_Name	Name of parameter to be sent into stored procedure.

tblPermissions

username	Foreign key from tUser.
Database	Foreign key from tDatabase.

<u>Transactions</u>

id	Primary key; auto-incrementing integer.
tpBegin	Begin date in the time period of the transfer.
tpEnd	End date in the time period of the transfer.
tsStart	Start time of the time stamp of the transfer.
tsEnd	End time of the time stamp of the transfer.
UserName	User responsible for the transfer; foreign key from tUser.
Data	

ds_<Data source table name>**

tid	Foreign key from Transactions.
<Columns from data source table>	

*tblStoredProcedures and tblSPParameters is the answer to dynamic stored procedures in the program. Since Windows SQL Server Express version does not allow dynamic insertion of table names in the code, a separate stored procedures class was created to be pseudo stored procedures. Acting in the same manner, they exist in the program rather than on the target database itself. These tables are queried in order to create dynamic SQL that is embedded into the program's functionality.

**Each transferred data source table will have the same name as in the original database but with a "ds_" prefix to denote "Decision Support", the project's target database.

4 Testing

The following are the test case scenarios implemented for the project, and the results of each of those tests.

Test Case	Expected Result	Actual Result	Pass or Fail
A: User opens application	Application Opens, user name and current date is displayed as well as the data sources the user can transfer from.	Application Opens, user name and current date is displayed as well as the data sources the user can transfer from.	Pass
B: User initiates data transfer	Transfer progress bar appears in a new window, updating dynamically and alerting user of transfer completion.	Transfer progress bar appears in a new window, updating dynamically and alerting user of transfer completion.	Pass
C: User chooses to view report	All selected reports are displayed to the screen.	All selected reports are displayed to the screen.	Pass
D: User chooses to print report	A print dialog opens for the user to select printing options, and when the user confirms the options, the selected reports are printed.	A print dialog opens for the user to select printing options, and when the user confirms the options, the selected reports are printed.	Pass
E: User chooses to save report to file	A save dialog opens for the user to select saving options, and when the user confirms the options, the selected reports are saved.	A save dialog opens for the user to select saving options, and when the user confirms the options, the selected reports are saved.	Pass