A Tool for Test Data Generation

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

By

Lakshmi Vaimpalli

In Partial Fulfillment of the

Requirements of the Degree of

Master of Software Engineering

March, 2010

A Tool for Test Data Generation

By Lakshmi Vaimpalli

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kasi Periyasamy Examination Committee Chairperson	Date
Dr. Dave Riley Examination Committee Member	Date
Dr. Mao Zheng Examination Committee Member	Date

ABSTRACT

LAKSHMI, VAIMPALLI, R., “A Tool for Test Data Generation”, Master of Software Engineering, March 2010, Advisors: Dr. Kasi Periyasamy, Dr. David Riley.

Software engineers struggle to find proper test data to test software products within time-to-market period. Under-tested software will not only cost a lot of time and effort for customers but also incur huge expenses for the company. Two major causes for not generating proper test data are lack of skills and insufficient time to prepare. The aim of this project is to develop a tool called Test Data Generation Tool (TDGT) that assists software testers in generating proper test data with minimal efforts. It provides mechanisms to read company-specific data sources as well as generic data sources such as flat files, EXCEL spreadsheets, XML documents and SQL Server databases. In addition TDGT helps software testers generate test data quickly so that they can spend more time finding bugs in the software rather than preparing test data. TDGT provides a rich user interface to design and execute the data generation task, and hence even a new tester can use this tool with minimal effort. TDGT can also be used to transfer data from one data source to other data sources. Such move is quite common to satisfy day-to-day business needs of testing department in a software company. Another usage of this tool is to view Unicode data for all the supported data sources mentioned above. This project also provides primitive metrics to help testers understand the process of test data generation.

ACKNOWLEDGEMENTS

I would like to express sincere thanks to my project advisors Dr. Kasi Periyasamy and Dr. Dave Riley for their valuable comments, guidance and splendid advices. Dr. Kasi spent many hours whenever I met him and provided constructive feedback on the project progress. I would also like to thank project sponsor, Mr. Dave Dobson, who initiated this project and provided a lot of support, time, resources, and comments during iterative demos with his expertise in application domain. I would also like to thank my employer, Business Objects an SAP company for the opportunity and encouragement to pursue this degree. In addition, I wish to thank my manager Dan Bills and co-workers Metin Tuzmen, Mark Spiess, Jeff Woody, and Pete McDonald for their comments and advices during various stages of this project and in particular Metin Tuzmen who spent many hours to test this product and provided valuable feedback on UI. Finally, I want to thank my wife Anitha Yeddula for her patience and support during the tenure of this degree. Without her support, I couldn't have made this happen.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

.NET Framework

The Microsoft .NET framework is a software component included in Microsoft Windows operating system. It provides pre-coded solutions to common software development and manages execution of programs written specifically for the framework. The .NET framework is intended to be used by most new applications created for the Windows platform.

ADO.NET

A set of software components that can be used by programmers to access data and data services. It is a part of the base class library that is included with the Microsoft .NET framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also be used to access data in non-relational sources.

API

An Application Program Interface is a collection of methods targeted to a computer operating system or to an application program by which a programmer can make requests of the operating system or the application.

Code page

Code page is another name for character encoding. It consists of a table of values that describe a character set for particular language. It maps logical character codes to single or multiple-byte character representations. It is used by an operating system to correctly display and print a language.

Configuration

Configuration is set of options combined in a XML document to process specified task by reading data, processing data based on the rules specified in the XML document and output the processed data.

DataTable

Represents one table of data in memory. The DataTable is a central object in the ADO.NET library. Other objects that use the DataTable include DataSet and DataView.

DataSet

Represents cache of data in memory. DataSet is a major component of the ADO.NET architecture. It consists of a collection of DataTable objects that one can relate to each other

with DataRelation objects. DataSet can also persist and reload its contents as XML, and its schema in XML notation.

DTD
DTD stands for Document Type Definition which is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, and HTML). DTD is kind of XML schema.

Flat Files
Flat files are data files that contain records with no structured relationships. Additional knowledge such as the file format and other properties are required to interpret these files. Each line in a flat file holds one record, with fields separated by delimiters such as commas or tabs.

HTML
Hyper Text Markup Language is a markup language designed for creating web pages and other information to view on a web browser.

ICU
International Components for Unicode is an open source software library package and is widely used set of C/C++ and Java libraries providing Unicode and globalization support for software applications.

Referential truth data
A static data set that is designed to exercise and test a subset of functionality within a SAP software product. The data in the referential truth data set does not change over time, and is designed by testers to force all code paths within the product to be executed.

Simscore algorithm
Similarity score or simscore is based on the number of differences between the values returned as percentage score that reflects the similarity of the two strings. The calculation is based on a number of factors including common data entry and spelling errors, codepage conversion errors, and omissions.

SQL
Structured Query Language is a development language created for manipulation of data in relational databases.

SQL Server
Microsoft SQL Server is a relational database management system (RDBMS).

SQL Scripts
These are textual scripts written in SQL language for creating tables in SQL Server databases.

SRS
Software Requirements Specification is a document format supplied by the IEEE 830-1998 Standard for specifying the requirements of a software system.

STL Port
It is an open source implementation of multiplatform ANSI C++ standard library implementation.

TDGT
"Test Data Generation Tool" is the name of the product described in this report. Initial version of this product is named as TDGT 1.0.

Trolltech QT
It is a cross-platform application and UI framework. It includes a cross-platform class library and integrated development tools [10].

XML
Extensible Markup Language is a W3C recommendation for creating special-purpose markup language and is widely used to exchange data and to store configuration data along with many other uses.

XML Data Record
It is a data record in XML format in which each field is encompassed in XML element and all such elements are placed in a parent element called Record.

XSLT
XSL Transformations is an XML markup language used for transforming XML documents.

1. Background Information

Software testing is a fundamental component of software quality assurance. The greater visibility of software systems and the cost associated with software failure are motivating factors for thorough testing. It is common for a software organization to spend considerable amount of effort on testing. Maintaining high quality standards of a software product requires exhaustive testing at various stages in the product life cycle. Such testing requires an extensive collection of appropriate test data. Preparing proper test data is therefore a core part of a software development project.

Designing test data requires a lot of time and expertise irrespective of the testing method used. If an organization doesn't have a systematic approach for building test data, there are chances of missing some important test cases and test scenarios. Therefore, it becomes the tester's responsibility to ensure a maximal coverage of test data being used for testing. While it is not possible to prove the exhaustiveness of test data coverage, testers must attempt to generate full coverage of test data from the test cases. After all, the end result of testing is to convince that the product satisfies the stated requirements in all possible scenarios.

1.1. Problem Statement

Products developed by SAP at La Crosse are known for improving the enterprise customer data through cleansing (Address Cleanse) and matching (Match) process by providing data parsing, cleansing, standardization, matching, and consolidation capabilities. Address Cleanse verifies that the relationships between city, state, and ZIP code. If an address contains only city, and state, then Address Cleanse usually can add ZIP code, and vice versa. Address Cleanse also standardizes the address line by correcting misspelled street name, filling in missing information, and stripping out unnecessary punctuation marks.

Many products that are developed at SAP attempt to match customer data against a set of referential truth data (master data). The Match process follows complex rules and is often "fuzzy" in nature, matching and returning data that is close to the original input data. The Match product is responsible for performing matching based on the business rules specified by the user and return the matching records by means of comparing name, address, and other customer data. The project described in this manuscript does not consider Address Cleansing.

The following Figure 1 provides better picture of the application domain problem and how this project will solve it. The lower portion of the diagram in red is the actual Match Product in application domain required testing. Referential truth (input) data fed into the Transformations component which is the core project logic and produces the output data sets which are real test cases to test the Match Product. In this diagram referential truth data contains one string "*BOULEVARD DELA*" which is input to the project and is transformed into three variations "*BOULEVARD D L", "BLVD DE LA", and "BOULEVARD*". These three variations are actual test cases to Match Product.

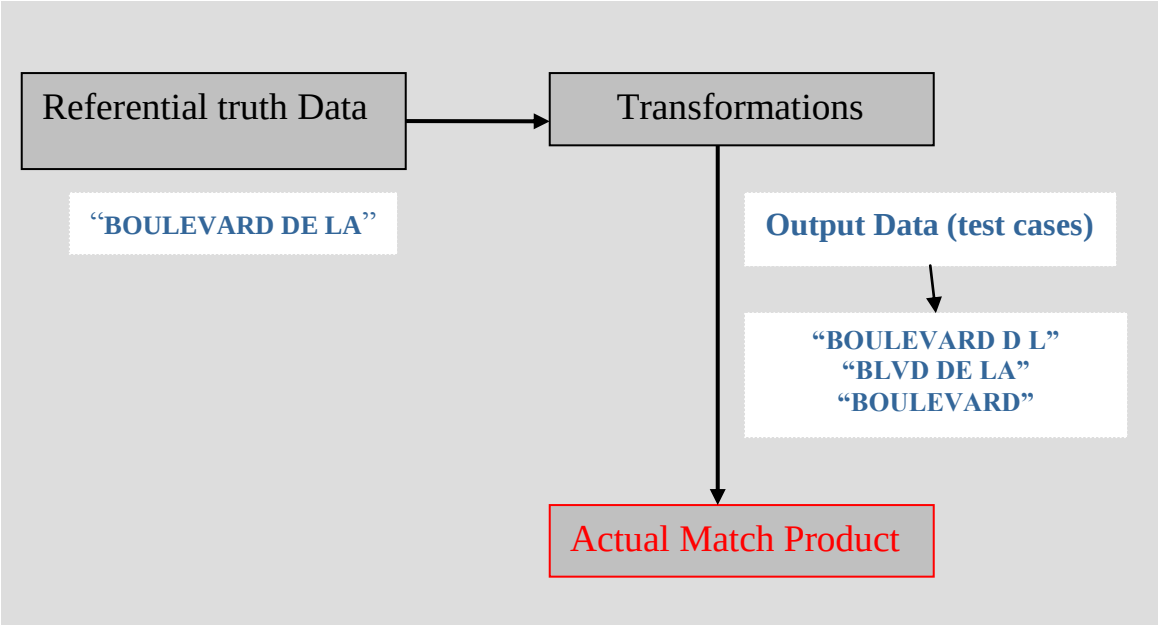


Figure 1 Project output data being used as test input to Match Product

Producing such high quality products requires a rigorous testing process at different levels of product development. Some of these testing levels require large sets of test data. Common types of data sources that SAP customers use are flat file, Excel, XML and database tables. Various test teams at SAP La Crosse branch spend extensive time in preparing test data sets in order to test different products that match the incoming data against a set of referential truth data sets. Designing test input data that validates the lookup and ensures fuzzy match are functioning correctly is a difficult problem. It is a very time consuming process and requires lot of manual effort and also requires a lot of expertise to prepare such data. One approach to solve this problem is, generate the test data from small set of referential truth data. The test input would be produced with knowledge of the matching algorithm (simscore) and would contain enough variations to fully test the matching and lookup logic.

The match process can be illustrated with an example of address data that contains one row of an address with multiple fields such as the following:

"BOULEVARD DE LA", "2", "VILLETTE", "", "", "BLVD", "PARIS", "", "PARIS", "75010", "FRA"

For each field in the selected row, the tester must generate a new data value that is:

- too dissimilar to be matched meaning the new data is more or less completely different from the actual data;
- just below the matching threshold of similarity meaning the new data is almost similar to actual data; or
- exactly at the threshold of similarity meaning the new data is same as actual data.

These are some of the lookup values, but a user can generate as many similarity levels as he/she might like. One approach to generate these variations of field values is to transform the original input, character by character, and then getting a similarity score of the transformed value, repeating until the original input is sufficiently dissimilar. Here is an example of referential truth data and the test input data derived with different variations.

Referential truth data:

"BOULEVARD DE LA", "2", "VILLETTE", "", "", "BLVD", "PARIS", "", "PARIS", "75010", "FRA"

Only one field (Street with prefixes) in the above record is modified in this exercise.

Exact Match - "BOULEVARD DE LA","2","VILLETTE","","","BLVD","PARIS","","PARIS","75010","","FRA"
High Match - "BOULEVARD D L","2","VILLETTE","","","BLVD","PARIS","","PARIS","75010","","FRA"
Medium Match- "BLVD DE LA","2","VILLETTE","","","BLVD","PARIS","","PARIS","75010","","FRA"
Low Match - "BOULEVARD","2","VILLETTE","","","BLVD","PARIS","","PARIS","75010","","FRA"

It is in this context, SAP decided to develop a tool that will assist users in generating test data. This manuscript describes the design and development of such a tool called Test Data Generation Tool (TDGT). This tool assist testers in generating test data in various formats (flat files, excel, xml and database tables) with minimal user interaction. It is written as a Windows-based standalone application in Microsoft .NET C# language. It takes different sources of SAP proprietary referential truth data as input and generates different variations for each field in a record. The referential truth data source can be either database tables or encrypted files and the output generated from this tool is captured in any one of the formats mentioned earlier.

1.2. Application domain dependencies

Since this project is supposed to use SAP referential truth data sources as input, it assumes dependency on some application domain components and their dependencies. Similarly the project uses Match Product related libraries from application domain to calculate similarity score for any given two strings which is the main deciding factor to generate variations.

2. An Overview of TDGT

This chapter is divided into two sections; first section gives a brief introduction of software life cycle models and the specific model used in this project. The later section describes the motivation and background of the project. In this chapter, the term "legacy applications" refers to old versions of SAP applications being used by customers.

2.1. A Brief Introduction to Software Life Cycle Models

A software life cycle describes the activities performed at each stage of software development. It depicts the significant phases or activities of a software project from conception until the product is retired. Typically, the life cycle addresses different activities of a software project: requirements, design, implementation, integration, testing, operations and maintenance. Much of the motivation behind utilizing a software life cycle model is to provide a structure to avoid the problems of undisciplined hacker or corporate IT bureaucrat [9]. There is no single life cycle model that is best suited for every project. Choosing the appropriate life cycle model for a particular project is often a challenging task. Many factors such as project size, team size, available resources, deadlines, team skills and experience, business policies, and the application domain may influence the choices of a life cycle model.

Waterfall model was one of the oldest life cycle models used in many projects. The waterfall model describes a set of stages in which the development activity is seen as a steady flow from top to bottom (like a waterfall) through the phases of requirements, analysis, design, implementation, testing, integration, and maintenance [9]. There are many variations of the waterfall model but they all include the above core phases. An in-depth discussion of the waterfall model is beyond the scope of this report, but additional information can be found in many texts on Software Engineering [7].

One of the software life cycle models is the rapid prototyping model (also called throw-away prototyping model) [9]. Rapid prototyping is used as a means to enhance the requirements gathering process from a customer. During the requirements phase, a quick and dirty throwaway prototype can be constructed to visually show the users what the end product may look like when the requirements are finally implemented. Rapid prototyping involves creating a working model of the system at a very early stage, after a relatively short investigation. The method used in building the prototype is usually quite informal, the most important factor being the early delivery of the prototype to the customer. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When this has been achieved, the prototype is thrown away and the system is formally developed based on the identified requirements. The major advantage of the rapid prototyping model is that it is easy to refine these requirements early in the development cycle. The strength of this prototype lies in its ability to construct interfaces that the users can test and give feedback on.

Another prototyping model is evolutionary model in which an initial prototype is developed as in rapid prototyping. But instead of throwing away the prototype after customer feedback, the prototype is kept as the base. Further development occurs as a stepwise refinement of this initial prototype. In other words, every iteration of the prototype adds new functionalities built on the previous prototype.

In this project, evolutionary prototyping model is used since there were no definite requirements and the application domain was not fully understood at the beginning. During each prototyping stage, the developer interacted with potential users, gathered additional requirements or refined previous set of requirements, built a prototype and demonstrated it to the users. Feedback received at the end of each prototype cycle was used in the next stage.

2.2. Background of TDGT

The Test Data Generation Tool (TDGT) was developed to help quality assurance teams in SAP to quickly create test data and to test products through different phases of software development. The entire project was divided into five stages: (1) Understand the application

domain and capture high level requirements. (2) Input data sources design and implementation. (3) Output data sources and XML data handling design and implementation. (4) Backend and user interface design and implementation. (5) Match Product integration and building transformation logic to generate different variations of test data. The main goal of the first stage was to understand the application domain and to capture high level requirements for the tool. During the first stage, there were discussions exclusively with potential users who are familiar with the application domain and who have an understanding of what the product's intended behavior. The application domain of the project is so vast that the developer was required to understand the different sources of reference data directories (data sources) and to choose only a few sources for this project. Using reference data from these courses, the tool was expected to generate different variations of data records as described in section 1.1 of Chapter1 based on design rules provided by the user (described in subsequent chapters). Further, the tool was required to integrate with different SAP products.

2.2.1. Data Sources

In the second stage, requirements were created for relational and referential truth data access mechanisms. The data sources included Microsoft SQL Server, Delimited flat file, Microsoft Excel and XML data sources. Accessing referential truth data sources requires project integration with different legacy SAP applications. It took considerable amount of time for the developer to understand the legacy applications before writing requirements. The main issue with the integration of legacy SAP applications is the lack of shared libraries. The problem was solved using external process communication in which the referential truth data was converted into XML format before being accessed by the tool. All high level requirements pertaining to different data sources are described in Table 1. Based on these data sets, a prototype was developed with minimal number of user interface screens and a demonstration was given to potential users. The prototype was revised based on the feedback received from potential users. The referential truth data sources contain different customer data components (also called multi-line data) shown in Figure 2. In multi-line

address data format, the address line indicates combination of primary and secondary address into one line, whereas last line address indicates the combination of city, state, and zip code into one line.

Data Source	**Requirement**
US DIR reference data source	This requirement facilitates access to the United States directories through separate process communication. Address and last line related data can be accessed.
AUS DIR reference data source	This requirement facilitates access to the Australian directories through separate process communication. Address, city and postcode related data can be accessed.
CAN DIR reference data source	This requirement facilitates access to the Canadian directories through separate process communication. Address and last line related data can be accessed.
DAS DIR reference data source	This requirement facilitates access to the Generic directories through separate process communication. Often address and last line related data can be accessed along with any other structural data.
MSSQL database source	This requirement facilitates access to the Microsoft SQL Server database through ODBC connectivity. It uses the Bulk Insert feature of SQL Server database. Customized ODBC driver has been developed to fulfill the requirement.
Delimited Flat File data source	This requirement is added to enable a user use delimited flat file data as source. Most of the SAP legacy customers use this kind of data files, and so this feature is handy to test customer issues.

Microsoft Excel data source	This requirement is added to enable a user use Microsoft Excel file data (*.xls) as source. Some legacy customers use this kind of data files, and so this feature is handy to test such customer issues.
XML data source	This requirement is added to enhance the real-time testing capabilities, since most of the real time testing is done through XML data.

Table 1 High level requirements for input and output data sources

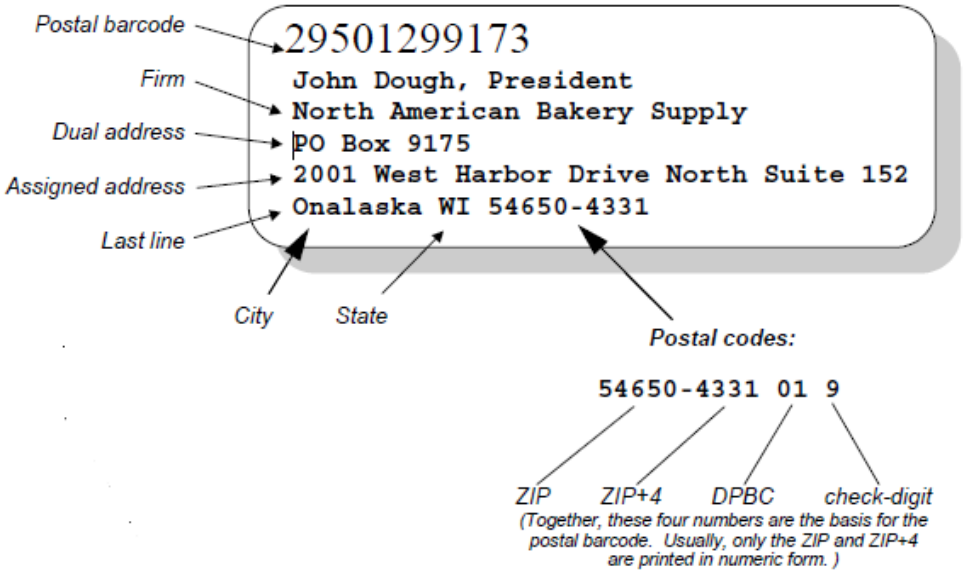


Figure 2 Multi-line address format

In the third stage, requirements were captured to support different output data sources and XML data record handling. The TDGT-supported output sources are Microsoft SQL Server, delimited flat files, Microsoft Excel and XML data files. An output schema consists of data type and length of each field in a data record. The data type of each field is derived from input data record and the length is computed using transformation rule for that

field. Users can modify an output schema and also control the number of records being output. All requirements related to output sources are described in Table 1.

2.2.2. Application logic and User Interface

The fourth stage mainly focused on backend framework, user interface design, and implementation. The developer came up with robust, flexible and reusable framework components that suit input and output data source types discussed above. Users can create transformation rules pertaining to one type of task called "Configuration" which will generate test data from input data. Multiple versions of a configuration can be maintained just by replicating existing versions.

In the fifth stage, requirements and design were created to build the transformation logic to generate different variations of input data record by using user specified threshold scores. This is the core logic of the project and is required to be integrated with SAP Match Product family discussed in chapter 4. The Match Product[1] is responsible for performing matching based on the business rules specified by the user and return the matching records by means of comparing name, address, and other customer data.

2.2.3. Salient features of TDGT

The important features of TDGT are summarized below:

1. Multiple data sources are available for both input and output. This enables users to access data from wide variety of sources such as proprietary data sources, Microsoft SQL server, flat files, Excel files and XML files.
2. The project is designed in such a way that other address plug-ins can be integrated with minimal effort. Reusable configuration objects are available to users to create new

[1] Match Product is one of the software components that are developed by SAP.

configuration by replicating existing ones. Custom UI controls improve efficiency and lead to less code maintenance.

3. Output schema editing is supported. Thus, users can modify output schema (data type, length and number of fields) before writing to target source. This feature gives better control on target source structure and data.
4. Input and output queries are captured and displayed to user. Custom list view controls implemented in the project enable the user to create queries on input and output schema.
5. The project is implemented using Microsoft .NET library and hence is capable of handling Unicode data.
6. This tool can be used in multiple ways, one of which is just to transfer data from one type of source to another type of source. Type of sources supported is described in Table 1.
7. Metrics are generated to analyze transformation of input data to output data. These metrics can also be used for administrative purposes.
8. An XML tree view of the configuration rules provides a clear picture to users on how the configuration looks like and therefore makes it easy to understand the configuration.

3. The Design and Development of TDGT

This chapter explains high-level architectural design, detailed design of the TDGT 1.0, data source and targets design, user interface design, and deployment along with some detailed usability issues that were factors in its construction. It also presents an overview of the challenges encountered during the design, and implementation phases.

3.1. High Level Architectural Design

TDGT 1.0 was developed using Microsoft .Net Framework 2.0 configuration and C# language. It is a standalone windows-based application. Users interact with the user interface to create and execute configurations. The user interface provides options to select the type of input and output data sources, metrics generation, and type of transformation rules to plug-in. The high level architecture of TDGT 1.0 is presented in Figure 3.

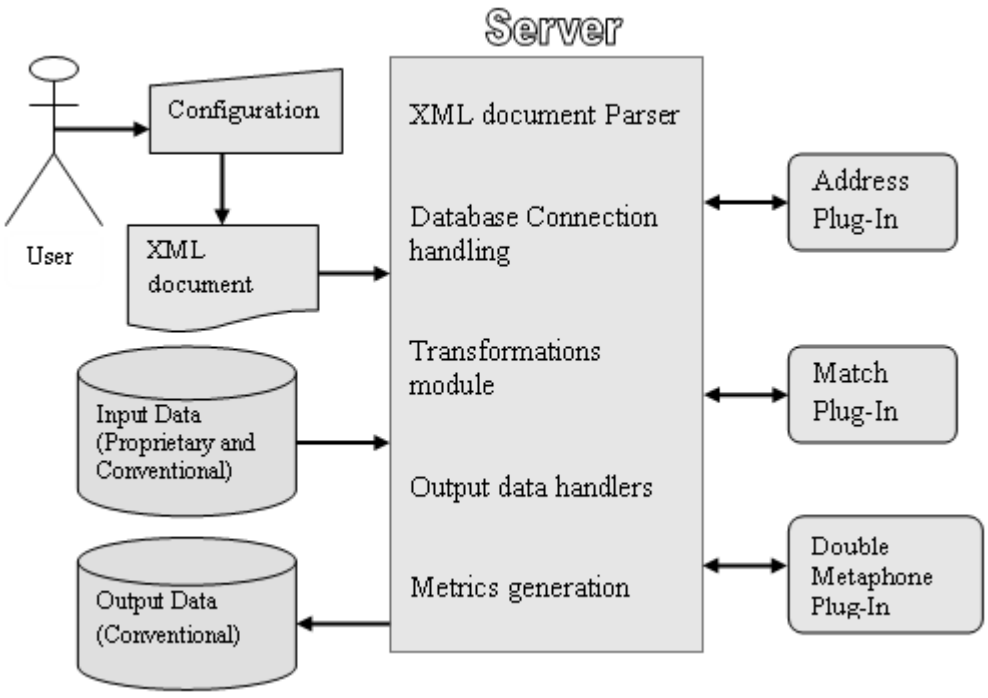


Figure 3 High-Level Architecture of TDGT 1.0

Depending on the values selected on user interface screens, the Server module (backend framework) loads the handlers specified in Figure 3. Results are written to output data sources. The Server module handles important operations such as transformation logic, plug-in integration, database connection, metrics generation, input and output data handlers. The plug-in integration code takes care of integration with SAP Match Product and Double Metaphone algorithms. Double Metaphone is an algorithm to code English words (and foreign words often heard in the United States) phonetically by reducing them to 12 consonant sounds. This reduces matching problems from wrong spelling. This algorithm is used in this project and it returns a rough approximation of how an English word sounds, which should be the same for words or names that sound similar, and can be used as a lookup key.

3.2. Detailed Architecture of TDGT

The classes in TDGT form multiple layers - the presentation layer, the application logic layer, and the database handling layer. The application layer is further divided into four categories - input and output data handling, XML document processing, the transformation module, and the metrics generation module. The application layer performs entire backend processing of the system. The database layer provides interaction between the application layer and the database. Due to space limitations in this manuscript, detailed architecture is shown for only important components of TDGT 1.0 system using UML class diagram in Figure 4.

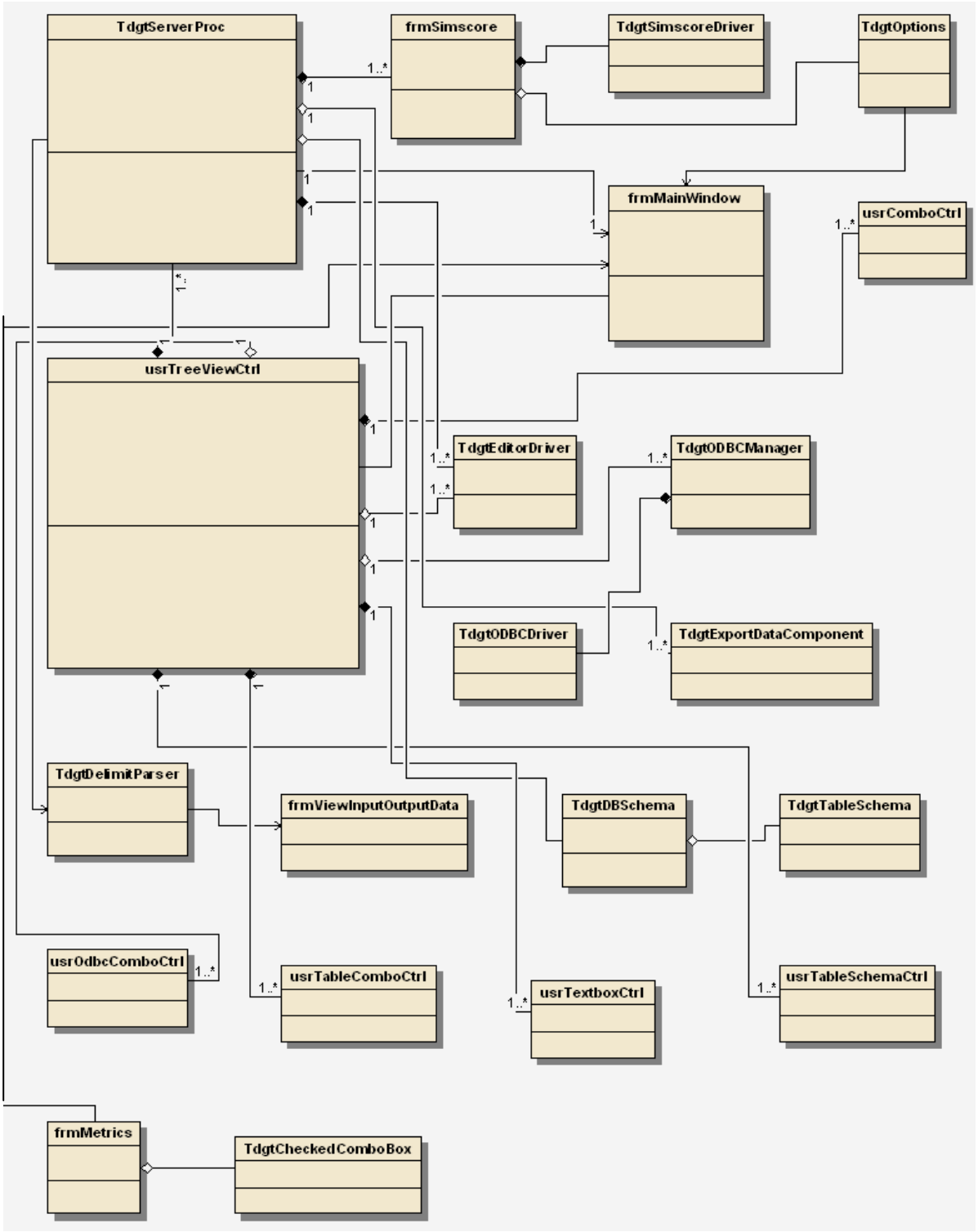


Figure 4 Detailed architecture of important components in TDGT 1.0 system

Important attributes and methods of class **TdgtServerProc** is described in Table 2.

Attribute/Method	**Functionality**
uTreeView	Attribute represents the TreeView Editor class usrTreeViewCtrl.
frmMain	Attribute represents the application Main Window class frmMainWindow.
simscoreTable	Attribute of type DataTable, holds the simscores for all the fields of the data record.
inoutColumnCollection	Attribute of type NameValueCollection, captures the input and output fields and their types.
outSrcColumnSchema	Attribute of type NameValueCollection, holds output source field schema in SQL syntax.
WriteAusDirInterimOutput	This method integrates SAP legacy applications to fetch the Australian reference data directories. Based on user inputs, this method calls appropriate modules and writes the fetched data into a DataTable object. It then calls TdgtSimscoreDriver object, generates the data and then writes the output data by using user output options captured in one of the output classes. It integrates TdgtDBSchema, TdgtODBCManager, TdgtDelimitParser, TdgtExportDataComponent classes for writing the output.
WriteCanDirInterimOutput	Similar to WriteAusDirInterimOutput method but this method process Canadian reference data directories.
WriteUsDirInterimOutput	Similar to WriteAusDirInterimOutput method but this method process United States reference data directories.
WriteDasDirInterimOutput	Similar to WriteAusDirInterimOutput method but this method process Generic Directories reference data.
WriteMSSQLInterimOutput	This method integrates the TdgtDBSchema,

	TdgtODBCManager for input data and TdgtDBSchema, TdgtODBCManager, TdgtDelimitParser, TdgtExportDataComponent classes for writing. As the name suggests this method is to read and write SQL Server data. All other operations are similar to the method WriteAusDirInterimOutput.
WriteFlatfileInterimOutput	This method integrates the TdgtDelimitParser, TdgtExportDataComponent for input data and TdgtDBSchema, TdgtODBCManager, TdgtDelimitParser, TdgtExportDataComponent classes for writing. As the name suggests this method is to read and write flat file data. All other operations are similar to the method WriteAusDirInterimOutput.
WriteProcessedOutputData	This method is called from all of the above methods to write the processed data with different parameters depending on the type of input, output source types. This method parses all output options such as number of records, output source type, character encoding, and replace substitute parameters with actual values before writing the data.

Table 2 Attributes and methods of class TdgtServerProc

Important attributes and methods of class **TdgtSimscoreDriver** are described in Table 3. Architecture diagram for this Class interaction is shown in the appendix A.

Attribute/Method	**Functionality**
strHandler	Attribute represents the TdgtStringHandler class which does all low level string operations.
strSMHandler	Attribute represents the TdgtSmallStringHandler class

	which does all low level operations for small strings to speed up the process.
MatchStrByRemoveChar	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings for a given string by removing a character in different ways. It then iterates through all transformed strings with the original string. If the user specified score matches with the calculated score, the string is returned otherwise it moves to next method.
MatchStrByChangeCase	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings for a given string by changing case of an alphabet from beginning, ending and middle of a string. It then iterates through all transformed strings with the original string. If the user specified score matches with the calculated score then the string is returned.
MatchStrByReplaceChar and MatchStrByReplaceMultipleChars	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings for a given string by replacing a character in different ways. It then iterates through all transformed strings with the original string. If the user specified score matches with the calculated score then the string is returned.
MatchStrByAddingChar	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings for a given string by adding a character to the original string in different ways. It then

	iterates through all transformed strings with the original string. If the user specified score matches with the calculated score then the string is returned.
MatchStrBySplitStrings	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings for a given string by splitting original string into two substrings. It then iterates through all transformed strings with the original string. If the user specified score matches with the calculated score then the string is returned.
MatchStrByReverseIt	This method integrates methods from classes TdgtStringHandler and TdgtSmallStringHandler to compute all possible strings by using reverse string operations. It then iterates through all transformed strings with the original string. If the user specified score matches with the calculated score then the string is returned.
GenerateSimscoreData	This method integrates the functionality related to capturing field threshold scores, and simscore calculation for each field in a record and then finds the expected transformed field string. It determines the order of the string manipulation operations to achieve the targeted score with minimum number of iterations. The method also collects all transformed records data into DataTable for further use.

Table 3 Attributes and methods of class TdgtSimscoreDriver

Important attributes and methods of a class **usrTreeViewCtrl** are described in Table 4.

Attribute/Method	**Functionality**
driverlist	Attribute of type TdgtODBCMngr.TdgtODBCDriver represents the list of ODBC drivers
dsnlist	Attribute of type TdgtODBCMngr.TdgtODBCDSN represents the list of ODBC dsn list.
dataOutput	Attribute of type TdgtServerProc class
m_dbSchema	Attribute of TdgtDBSchema class
tdgtEditor	Attribute of type TdgtEditorDriver
NewConfiguration	This method creates new configuration XML document from the configuration template. The new configuration XML has all the nodes with default values for those options which are mandatory.
DisplayScreen	This method takes the input XML node path and Tree node as input parameters. It iterates through all nodes in Tree View editor by using TdgtEditorDriver object and displays the values of XML document to the corresponding nodes. During this process, it calls all custom controls and binds them to the appropriate Tree Editor nodes.
SaveDialogData	This method takes the XML node path as input parameter and save the values at each node into the XML document. It iterates through all nodes in Tree View editor by using TdgtEditorDriver object. It identifies each custom control associated with the node and saves the values entered by user for that node. If there are any dependant controls then it goes through all those, updates the values in those controls and saves them back in XML.

Table 4 Attributes and methods of class usrTreeViewCtrl

Test data generation for a given input data record is shown in the use case model shown in Figure 5. This diagram includes operations on input and output sources as well.

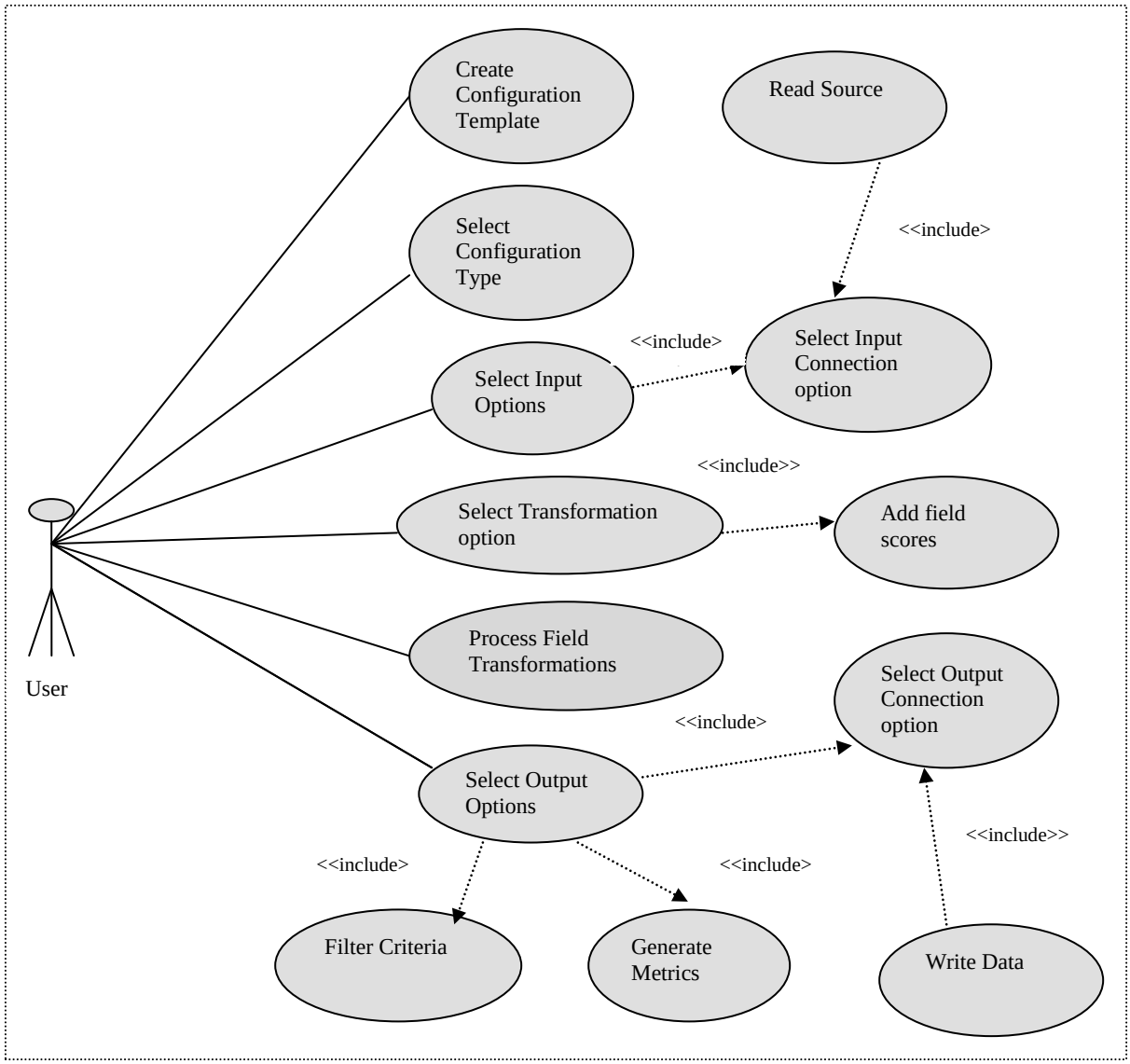


Figure 5 Use case diagram for test data generation of a record at higher level

Use case diagram shown in Figure 6 shows the detailed operations in field transformation process which is part of use case diagram shown in Figure 5 above.

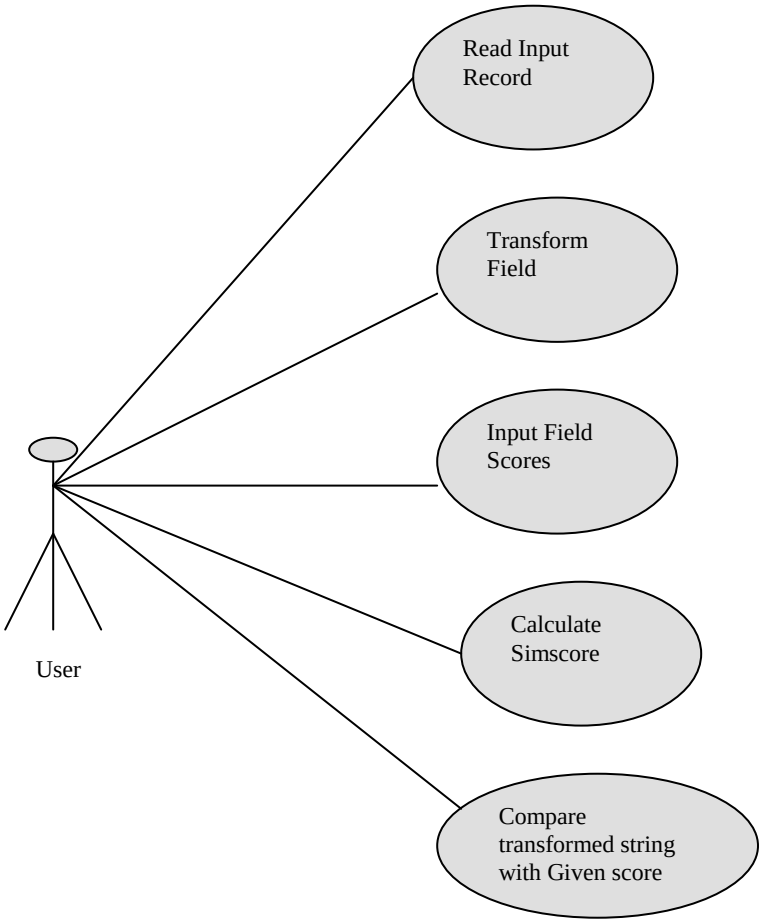


Figure 6 Use case diagram for Field transformation process

3.3. Server Process Design

The input and output data handler module shown in Figure 3 reads data from SAP proprietary, conventional (SQL Server, Delimited file, Excel and XML) sources and writes processed data back to conventional data sources. Since SAP legacy applications (data access components) were written in pure C and are available as static libraries, integration was done by creating different legacy application executables outside this project workspace and integrating them with the project as separate processes. Changes in proprietary data structure and format require rebuilding these executables. Some of the main modules designed to implement this feature are TdgtServerProc, TdgtDelimitParser, and TdgtExportDataComponent. SQL Server database operations are handled in TdgtServerProc, DBSchema etc. Integration between different components shown in Figure 3 became easy since each of them is implemented as independent classes.

The project uses XML format for internal data manipulations. The main reasons behind the extensive use of XML are: (1) XML defines a set of customized tags and also allows for flexibility in the formatting of the rules. (2) Microsoft .NET includes a suite of XML APIs built on industry standards such as DOM, XPath, XSD, and XSLT. (3) The .NET Framework XML classes also support innovations that offer simplicity, better performance, and a more familiar programming model, tightly coupled with the new .NET data access APIs in ADO .NET. XmlWriter, XmlReader, XmlNavigator, XMLTextReader, XMLTextWriter, and encapsulate a number of functionalities that previously had to be accomplished manually. (4) Additional important features of XML document processing are Unicode compatibility and handling of escape characters in. Loading XML, parsing individual nodes and writing processed data to XML document is handled in TdgtOptions.

The data read from the input source is converted to XML document is handled by TdgtOptions, TdgtEditorDriver and TdgtServerProc. TdgtEditorDriver class handles operations such as binding XML nodes with XML tree editor schema, mapping XML nodes with that of schema editor, replicating nodes, finding repeating nodes, copying nodes, retrieving XSLT paths from xml hierarchy, saving nodes etc.

The data transformation is implemented in a separate module and is integrated with SAP Match Product technologies. This module handles various string manipulations, and operations on XML data sets such as DataTable, and DataSet. The data transformation module is implemented in such a way that it is expandable to integrate any other SAP plug-in with little effort. Based on the scores given to each field in a record, the tool computes matching strings. The transformation algorithm first iterates through individual fields in a record, applies several string manipulations and Match (simscore) algorithm calculates scores for each variation by comparison to the original string. When a score matches the user specified score, the variation string is returned. Similarity score calculation is done by the API's that are accessible through Match Product integration. Figure 7 shows how the Transformation module generates different variations of a string and Match Product integration with Transformation module. The Match Product is not developed by the developer and is integrated with this project to perform similarity score calculations of original string and the variation of original string. To decide whether two key fields match, the simscore algorithm measures the similarity of their content. Simscore algorithm looks for matching characters or fragments, and then evaluates the matching fragments as a portion of the whole. The amount of similarity is recorded as a percentage. For example, if the key fields being compared are *John* and *Joh,* then the simscore algorithm concludes that those two fields are 90 percent alike.

By itself, the simscore algorithm doesn't determine whether either or both of these key field comparisons should be considered a match - it just calculates the score. To determine whether that score should be considered close enough to call these key fields a match, the simscore algorithm must compare the similarity score to a value that is set by the user.

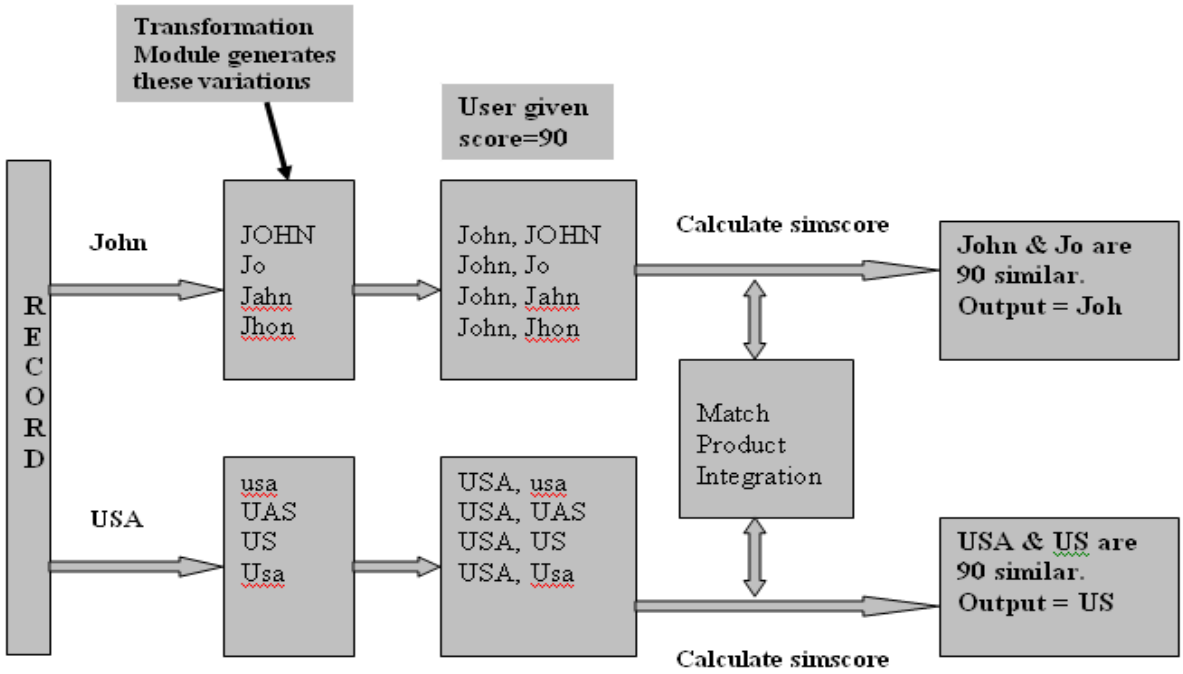


Figure 7 Match integration with Transformation module and simscore calculation

The two classes TdgtStringHandler and TdgtSmallStringHandler handle string operations by encapsulating similar type of operations in one group. The TdgtSimscoreDriver class is the main one which integrates Match Product API's with the project. It handles the integration in such a way that the string transformation operation is most efficient and is less resource intensive.

In addition to Match Product integration, the Double Metaphone algorithm is also implemented and integrated in the project using three classes TdgtDoubleMetaphone, TdgtShortDoubleMetaphone, and TdgtDoubleMetaphoneManager. This integration requires large set of Metaphone related names in order to process transformations for wide variety of data sets. It is implemented just to show that the project is capable of integrating any kind of address, and match related plug-ins.

3.4. User Interface Design

The user interface for the Test Data Generation Tool supports both Multiple Document Interface (MDI) and Single Document Interface (SDI). It contains a thick client or windows desktop user interface. This interface consists of .NET custom controls which were derived from existing .NET controls, customized properties, methods and events for custom needs. These controls enable users to work with a small number of windows and navigation is expected to be easy even for novice users. When the application is opened, the user is presented with the main window shown in Figure 8 and consisting of menu bar, tool bar, working panel, display panel, and .NET custom Tree control to navigate through configurations.

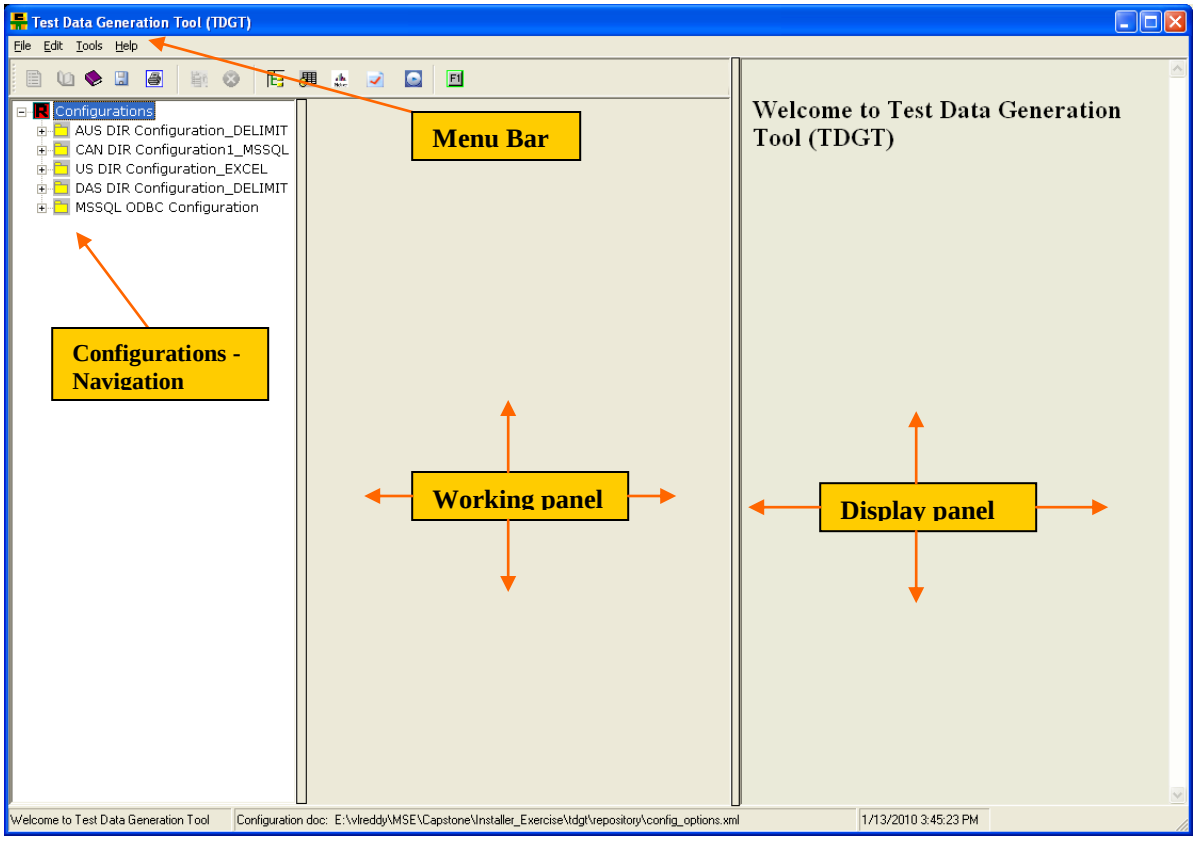


Figure 8 Main Application Window

Users can either start by creating new configurations or open existing configurations, using file menu or tool bar icon. The tool provides a template for new configurations which

consists of five base configurations to cover all user needs. A user can replicate as many configurations as required from those five base configurations. Each configuration performs certain types of data processing by using specific rules. The leftmost panel is the .NET Tree view control used to navigate through all configurations. This control provides a folder view of configurations through which users can drill down to a single option level. Menu options such as execute, copy, delete, save configurations are provided through the ContextMenuStrip menu, main menu and tool bar buttons. Of these operations only execute, copy, delete operations can be performed on configuration nodes.

The working panel window hosts user interface screens to capture user input. The working panel displays different sets of control, each set depending on the node selection on tree view control. Users can edit the control values in this panel. The controls in this panel have dependency relationships among themselves.

The display panel facilitates viewing configuration rules in tree format using Microsoft web browser control. Figure 9 shows how the .NET ListView control is customized as SQL schema editor where a user can select different table fields and place the SQL selection criteria. Based on the field selection and WHERE clause criteria, the SQL select statement is framed.

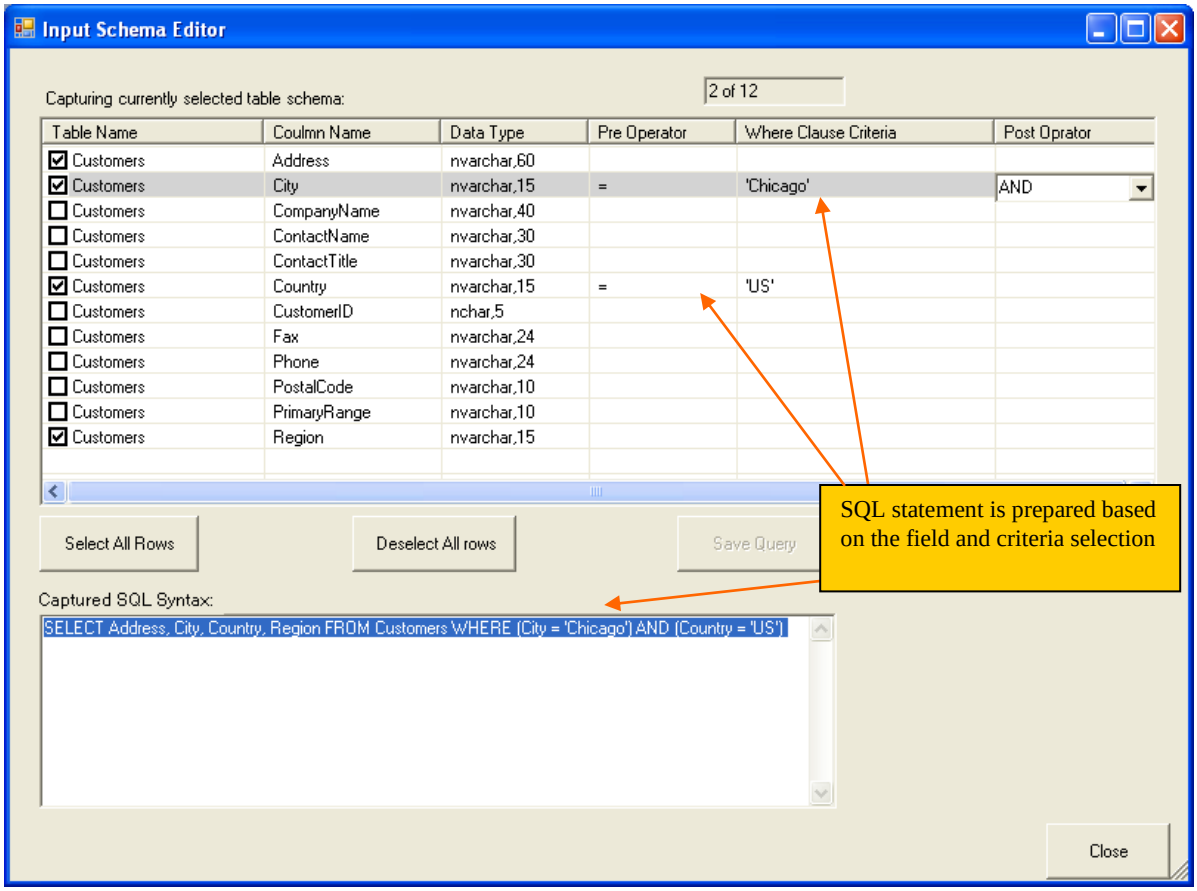


Figure 9 Customized .NET ListView control - SQL Schema editor

Figure 10 shows how the .NET Data Grid View control is customized to display metrics generated for each configuration execution. Metrics are stored and manipulated in XML format. The default .NET Data Grid View control does not handle such XML structures and hence this control is customized to display such data. When a user clicks on any cell, it will display row and field number of that cell in grid so that user can track the values in a particular row and field. There is a customized combo box called "Configuration" which contains all generated metrics names as items. Each item is identified with a check box and so a user can select multiple items to display their metrics or uncheck to remove them from the display.

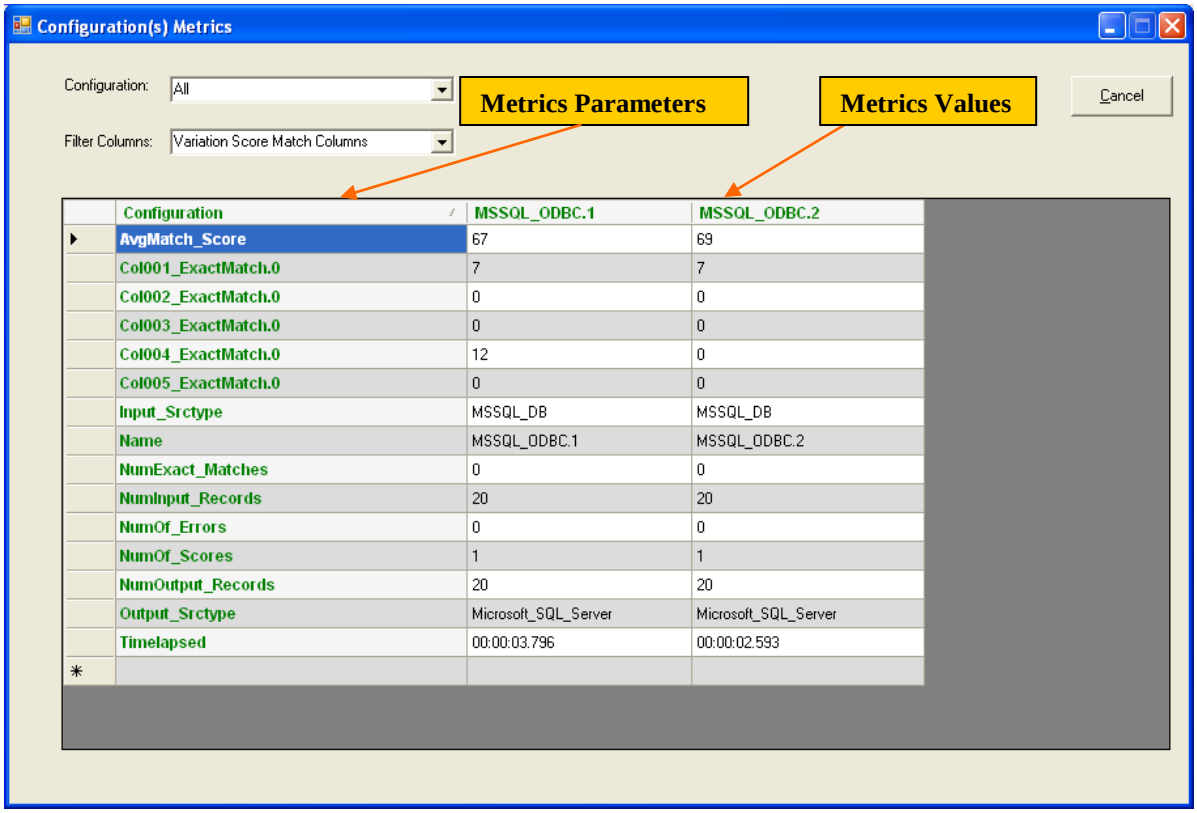


Figure 10 Customized .NET Data Grid View control

3.5. Metrics Design

After completing main functionalities of the project, the developer discussed the necessity of metrics for this project with the project adviser and users. Metric values represent the actual output data in multiple dimensions. Users do not need to go through the actual data in order to validate the data. Instead, they validate metric values and tune the configuration options one by one and execute them again until the required output data is arrived.

The metrics module is implemented in such a way that the application knows the format and versions of all generated metrics. Each run of a configuration generates metrics and is assigned with tag name and version number to keep track of the metrics for each configuration execution. The .Net Grid View Control is customized to view generated

metrics values. Figure 11 shows the XML structure for two metrics of ODBC configuration execution.


```
<Metrics>
 <MSSQL_ODBC.1>
  <Col001_ExactMatch.0>7</Col001_ExactMatch.0>
  <Col001_Match100pVar.0>6</Col001_Match100pVar.0>
  <NumOf_Errors>0</NumOf_Errors>
  <Col001_Match25pVar.0>1</Col001_Match25pVar.0>
  <Col002_ExactMatch.0>0</Col002_ExactMatch.0>
  <NumOf_Scores>1</NumOf_Scores>
  <Col001_Match75pVar.0>4</Col001_Match75pVar.0>
  <Col001_Match50pVar.0>2</Col001_Match50pVar.0>
  <NumExact_Matches>0</NumExact_Matches>
  <Timelapsed>00:00:01.453</Timelapsed>
  <Col002_Match50pVar.0>9</Col002_Match50pVar.0>
  <Col002_Match100pVar.0>3</Col002_Match100pVar.0>
  <Col002_Match25pVar.0>4</Col002_Match25pVar.0>
  <Output_Srctype>Microsoft_SQL_Server</Output_Srctype>
  <AvgMatch_Score>86</AvgMatch_Score>
  <Col002_Match75pVar.0>4</Col002_Match75pVar.0>
  <Name>MSSQL_ODBC.1</Name>
  <NumInput_Records>20</NumInput_Records>
  <NumOutput_Records>20</NumOutput_Records>
  <Input_Srctype>MSSQL_DB</Input_Srctype>
 </MSSQL_ODBC.1>
<MSSQL_ODBC.2>
 <Output_Srctype>Microsoft_SQL_Server</Output_Srctype>
 <Col003_Match50pVar.0>2</Col003_Match50pVar.0>
 <NumOf_Errors>0</NumOf_Errors>
 <Name>MSSQL_ODBC.2</Name>
 <Col003_Match75pVar.0>10</Col003_Match75pVar.0>
 <NumOf_Scores>1</NumOf_Scores>
 <NumExact_Matches>0</NumExact_Matches>
 <Timelapsed>00:00:01.796</Timelapsed>
 <Col003_Match100pVar.0>7</Col003_Match100pVar.0>
 <Col004_Match50pVar.0>8</Col004_Match50pVar.0>
 <Col003_ExactMatch.0>0</Col003_ExactMatch.0>
 <AvgMatch_Score>86</AvgMatch_Score>
 <Col004_Match100pVar.0>0</Col004_Match100pVar.0>
 <Col004_ExactMatch.0>12</Col004_ExactMatch.0>
 <Col004_Match25pVar.0>0</Col004_Match25pVar.0>
 <NumInput_Records>20</NumInput_Records>
 <Col004_Match75pVar.0>0</Col004_Match75pVar.0>
 <NumOutput_Records>20</NumOutput_Records>
 <Input_Srctype>MSSQL_DB</Input_Srctype>
 <Col003_Match25pVar.0>1</Col003_Match25pVar.0>
</MSSQL_ODBC.2>
</Metrics>
```


Figure 11 XML structure generated for configuration execution

3.6. Project Challenges

Most challenges for this project came from the application domain as well, as from the implementation phase. Initially, the reference data was supposed to be derived from one relational database source but due to other SAP products and internal constraints, this was not possible. So requirements were changed to include other data resources such as Australian directories, Canadian directories, US directories, Generic directories (DAS), and MSSQL database with the sponsor's approval. Substantial time was required for the developer to understand these reference data sources both in terms of application domain features as well as implementation.

Another challenge was the use of different drivers such as those for Delimit text file, Excel file and ODBC. These drivers were implemented in such a way that reading input data and writing the output data for all supported source types should be convenient for additional feature extensions. The implementation of transformation algorithms and integration of Match Product needed more time than anticipated because the developer was not familiar with the Match Product before this project was started.

One of the challenges during the implementation phase was to customize the GUI controls. Since many GUI controls were redesigned, customized and reused in several parts of the project, design of these controls and their reuse posed a challenge.

Another challenge was the binding of user controls and the tree view editor, and their synchronization for user actions. Each data input through a custom control has to be tracked against the corresponding tree node, saved and displayed back to the node. This becomes complex when multiple copies of the same configuration exist, as the code has to take care of multiple versions of the same node with different values at different node paths from the root node.

Since the configuration rules are in the form of XML document, a lot of consideration was devoted to parameterizing the option values. The developer came up with appropriate configuration parameters so that users can define variables and their values on the fly, and use them in individual option values instead of hard coded values. This feature provides easy maintenance when dealing with many configurations.

4. Implementation

This chapter describes server component and user interface implementation details of the project. It also includes the technologies used to develop the project.

4.1. Application logic and User interface

The application logic module contains important operations such as transformation logic, plug-in integration, database connection, input and output data handlers, and metrics generation. The user interface module handles all user interaction through windows screens. As described in previous chapters, all backend processing is done through XML data. All options and rules specified by the user are captured in XML documents in different option groups. Each group holds options and rules related to one common task and all groups are combined into one XML schema object called Configuration. A Configuration is the highest level of object available in the project and it is reusable (see Figure 12). To confine the building blocks of XML document and to validate the correctness of it, a formal Document Type Definition (DTD) or XML schema is required. In order to provide the DTD or XML schema, a separate schema editor class was created. The schema file consists of XML schema for all possible configurations with each individual XML element and their hierarchy in that configuration. Attributes such as title, type of control, height of control, control order, repeatability, and path from its root node are described for each element. The schema editor class binds each individual schema element with the actual XML option and each XML option is tightly bound with the respective schema editor element. The XML configuration verification is done through the schema editor so that the configuration XML can be controlled during configuration design and execution. All possible configuration objects are already built-in and made available as base templates, so they can be used when new project is created. There are five different configuration objects available and they are: (1) US DIR Configuration, (2) AUS DIR Configuration, (3) CAN DIR Configuration, (4) DAS DIR Configuration, (5) MSSQL and other types of Configurations. Any of these

configurations can be replicated and modified according to the needs. Users can't delete any built-in configuration objects but can delete the replicated configuration objects.

Object	Properties
Configuration	This is the highest level of object and is the root node on which user can perform execute, copy, save, delete etc. It is reusable in the sense that a user can copy the configuration from another user and save it as another version.
Option Group	Group of options related to one common task are combined together under one node. For example, "Connection Options" is an Input Options Group. This is not reusable, and a user can't perform execute, copy and delete operations on this node.
Option	This is the lowest level of object in the configuration object. Each individual option corresponds to one user interface control through which user can enter value. It is not reusable, and a user can't perform execute, copy and delete operations on this node.

Table 5 Different Objects properties in the project

Configuration Object hierarchy diagram is shown in the Figure 12 below.

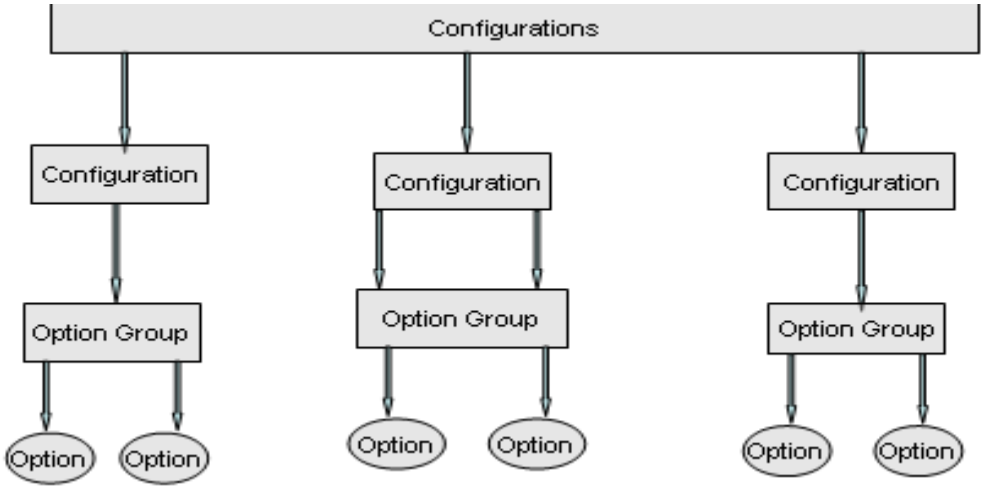


Figure 12 Object hierarchy diagram

Each custom control is implemented in a separate class with its associated properties and events. All custom controls created in the project are described in Table 6. Out of these, the Tree View control holds all configuration objects display and editing. Tree View control is embedded into the main window panel which holds all configuration objects (XML options) in Folder view. Tree View keeps track of XML options with the schema editor elements by using associated schema files. Custom Tree View control class performs operations such as displaying and saving the configurations into a file with XML node hierarchy and their corresponding values when user add/delete/modifies the values in user controls in editor window. Since Custom Tree View control and other user interface controls are residing in two different panels, the communication between these two is not automatic and is driven by user actions. So when a user updates any values in user interface controls in editor window, Tree View control needs to be focused in order to update the nodes in Tree View.

Control	**Functionality**
usrTextboxCtrl	.NET Textbox control is customized in usrTextboxCtrl class and is bound with individual XML option which is tied to Tree view control. Operations customized in this control are ‘set’, ‘get’ and ‘modify’. It is the most widely used control in the project.
usrComboCtrl	.NET Combo box control is customized in usrComboCtrl class to hold multiple values and is bound with XML option which is tied to Tree view control.
CheckedComboBox	.NET Combo box control is customized in CheckedComboBox class to list multiple items with check boxes against each item. The main advantage of this control is to provide the user to select multiple items from the combo box control.
usrOdbcComboCtrl	.NET Combo box control is customized in

	usrOdbcComboCtrl class to list multiple ODBC data sources and is bound with XML option which is tied to Tree view control.
usrTableComboCtrl	.NET Combo box control is customized in usrTableComboCtrl class to hold multiple values of table names which are retrieved from database. These values are bound with XML option which is tied to Tree view control.
usrFileOpenCtrl	This is a combination of .NET Textbox and Button controls and is customized to bind with the Directory path selection.
usrTableSchemaCtrl	.NET List View control is customized in usrTableSchemaCtrl class to capture SQL schema on Microsoft SQL input and output sources. This control populates all fields from the selected table and captures SQL statement based on the fields and 'where' clause criteria selection. Each cell in the list view control has 'select', 'edit', 'delete', 'mouse down', 'key press' events added. A user can select any number of fields and all available operators available in 'where' clause pre and post operands. In another class ListViewEx, List View control is customized slightly different way to implement configuration parameters window. Operations performed on this control are 'display', 'add', 'edit', and 'save'. The same class is also used in output schema editor window.
usrTreeViewCtrl	.NET Tree View control is customized to display all Configurations in folder view. All nodes in tree view

	are bound with individual controls in editor window described above. Each value entered through user controls mentioned above is tied to tree view nodes. So for every user control there is a corresponding node in tree view and vice versa. All option values entered through user interface controls are saved in a file in XML format.

Table 6 Customized .NET controls used in the project

4.2. Transformations through Match Product integration

Transformations on each field in the input data record utilize user-supplied threshold scores. This is the core logic of the project required by SAP Match Product family. It took a while to implement the integration successfully since Match Product relies upon static objects in the Visual Studio .NET 2003 compiler environment. In order to integrate C++ component with .NET C# framework, the C++ component should be available as Dynamic Link Library (DLL). Since the developer is not familiar with the Match Product development environment, the developer had to depend on Match Product team to build the DLL successfully. It took some time to overcome the problems in building the match products DLL in Visual Studio .NET 2005 environment. In order to access C++ match DLL in .NET C# project, it has to be included in the project reference and should be imported as shown in the Figure 13.

```
[DllImport("matchrules.dll", ExactSpelling = false, SetLastError = true)]
static extern int simscore_wrapper(string str1, int len1, string str2, int len2, short[]
options, int traspose);

private int getMysimscore(string str1, int len1, string str2, int len2, short[] options,
int traspose)
{
    int percent;
    return percent = simscore_wrapper(str1, len1, str2, len2, options, traspose);
}
public int calculateScore(string firstName, string secondName)
{
    ....
    return this.getMysimscore(firstName, firstName.Length, secondName,
secondName.Length, options, transpose);

}
```

Figure 13 Match C++ DLL integration code snippet

The code in Figure 13 shows some high level functions used in integration. The function “calculateScore” calls “getMysimscore” function which in turns calls the native match function “simscore_wrapper” to compute the simscore of two given strings. For each threshold score specified on a given field this function computes a simscore by modifying the second string through iterations. Different string operations such as removing one character, replacing one character, adding one character, reversing the string, changing the case of string characters, and so forth can be employed. All these string operations are implemented in separate a string library class. String operations in each category are grouped into one module for ease of integration and to determine the order of module execution during simscore calculations. Modified strings in each category of string operations are collected in separate lists and the above function computes the score for all strings in a list against the original string and this process continues on other lists till the matching score is found. When the user given score matches the score computed by this function then the process is terminated for this scenario.

A user will be able to enter multiple threshold scores for each field in a record. Each field is allowed to hold nine score values; scores of zero and one hundred are not allowed as they don’t make any sense in matching process. Based on threshold score values given to each field in a record, the integration component computes matching string. The same

process is continued on remaining fields in a record and is repeated for all scores on all fields in a record. Each field string goes through several iterations till it gets the targeted scores. Another option called “Variation” has been implemented to provide some flexibility to compute simscore because there may be times the computation score may not exactly match the user given score. In such cases the user given score is evaluated between *targeted score* and *(targeted score + Variation)* values. For any given field and a score, the integration component determines the order of the string manipulation operations to achieve the targeted score with minimum number of iterations instead of a sequential order. For example, the country field in a record will always have two-character strings and the string manipulation operation should come from the lower order of operations such as reverse string, casing or adding a character in order to get the matching string quickly. Improving the performance using multithreading and other parallel processing mechanism is out of scope of the current project as discussed with the project sponsor. Performance bottlenecks will be seen if the customer processes hundreds of thousands of records on a "lower end machine[2]". If the customer requires processing such huge number of records then it is recommended to use "high end machines[3]" to get the better performance. Figure 14 describes how this computation is performed.

[2] Windows Computer with basic hardware configuration such as single processor, 1 or 2 GB RAM.
[3]Windows Computer with multi-processor, higher processor speed, and more than 4 GB RAM.

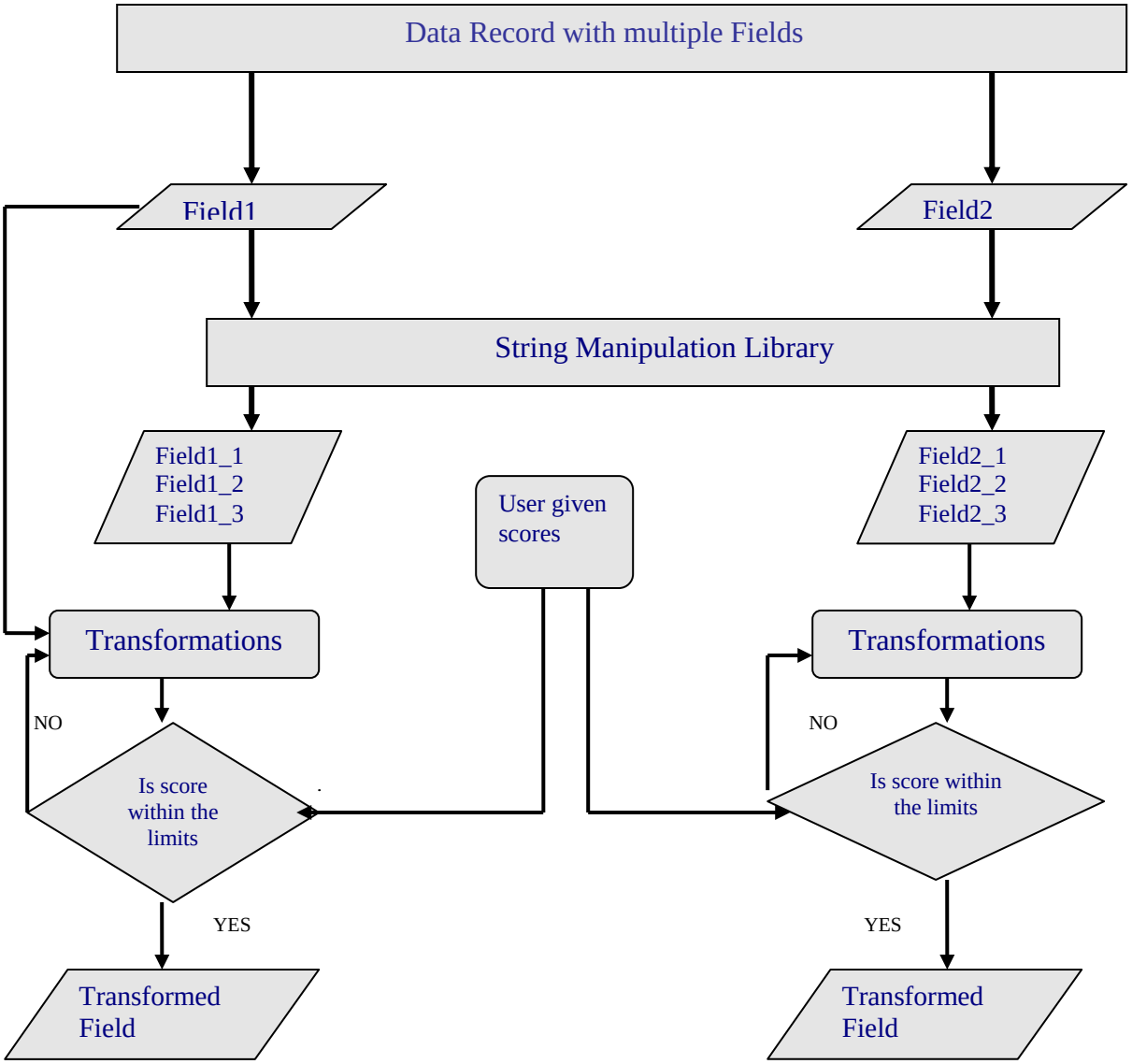


Figure 14 Field transformation chart

Metrics were created to analyze the behavior and functionalities of the tool. Users can use the metrics generated in a particular execution to determine how the output data is varied with regard to changes in the input options. When user wants to generate test data suitable to one particular test scenario, metrics are handy to determine whether the generated output data is what the user is looking for. Each execution of a configuration will result in a separate metrics table with a tag using the configuration name. Microsoft grid control is

customized to suite the project needs to display the different metrics tables. In order to display each metrics table field wise, the rows of a grid are flipped before display and should keep track of them for further user operations such as sorting or filtering. Context menu option is provided to user to delete the metrics table. Metrics developed in this project are of primitive type to make it easy for the user for analysis purposes.

4.3. Technologies

One of the challenges during the project design was to determine the best suitable technologies. Even though many technologies available, the main deciding factors are Custom control support, Database support, XML document capabilities, and Unicode data support. The obvious choices are Microsoft .NET and Java, and out of these two; Microsoft .NET C# language was selected as the project is a Windows application and doesn't require web support.

Microsoft .NET C# language, SQL Server 2000 & 2005, XML document is used in back end development and .NET custom controls are used in UI development. SQL Server is supported for data read and write operations. The back end process is done through XML document manipulations.

4.4. Deploying TDGT 1.0

The Test Data Generation Tool is deployed by installing the application on Windows machine that has the .NET framework installed. The Microsoft windows installer is created with x86 target platform in Visual Studio .NET 2005 project in order to support the installation on both 32 and 64 bit Windows machines. The installer creates different folder structures to install different SAP proprietary reference data directories, dependency libraries, XML schema files, and project XML template files. The proprietary reference data

directories are encrypted data files which are the sources of data for some types of configurations.

Dependency libraries are required to be integrated with proprietary data sources in the project. There is a bin folder which contains XML schema editor file “tdgt-editor.xml”, and a project configuration template file “config_template.xml” from which new project configuration is created. The configuration parameters file “tdgt_options.xml” is used to store all substitution parameters used in the template. Users can add more parameters during the usage. The bin folder also contains “tree-view” XSLT and cascading style files to display XML configuration in user readable tree view using web browser.

The input folder contains few input files to verify the installed software. Another folder called “repository” is added to hold the project working files such as project configuration XML files.

5. Limitations

The first limitation comes from the architecture of the application. Since the application is implemented as Windows-based thick client architecture, users have to install the user interface and backend server components. Re-installation is required for any bug fixes or enhancements and updates. Consequently, the application has to be compiled and reinstalled for any simple change in the software.

Another limitation stems from the proprietary reference data directories. Since the content and the format in these directories changes from version to version, external processes integrated with the project need to be rebuilt and deployed. Changes in the reference data structure will sometimes impact TDGT.

There is no implementation for user and data level security since it was not in the requirements. It was not considered as the main requirement because the project is going to be deployed internally at SAP quality department. Schema validation and configuration templates are XML based files, and hence any tampering on these files can corrupt proper TDGT behavior. Since there is no login requirement, no user level access rights are implemented.

Performance was not a high priority requirement; hence the developer didn’t implement multithreading or parallel processing techniques to improve performance of the application. Nevertheless, performance was given due importance in implementation.

6. Continuing Work

Since this is a data processing tool, it would be nice to have multithreaded and parallel processing support across all components to improve performance. Due to other high priority tasks and deadlines, this feature was removed from the list of initial requirements. During the implementation, developer has included only partial multi-threading.

The schema editor and configuration template are simple XML files. Redesigning them as a relational database will provide better security and scalability. This requires major changes to both the application layer as well as the database model.

A user should be able to execute the project in command-line mode with the required parameters. This feature provides the ability to automate the process and to be integrated with other tools.

Another useful feature would be support for different relational data types such as numeric, date, binary etc. at input and output sources. Such a feature would enhance the tool's capability to generate test data from all kinds of customer data.

7. Conclusion

This manuscript describes the features, design and implementation details of Test Data Generation Tool - a program to improve the efficiency of the Quality Department at SAP, La Crosse. TDGT generates test data from existing reference data sources with minimal user inputs. Using the rules provided by the user, the tool transforms the input data and generates different variations of test data. TDGT is a Windows-based software tool and is easy to use by any common user without having technical knowledge. It accepts different kinds of data sources such as SAP proprietary (Australia, Canada, US and Generic Directories) and conventional sources such as Microsoft SQL Server (ODBC), Delimited flat file, Excel data and XML. The repository within the application is file-based and so there is no need of relational database unless a user wants to process the data from SQL Server. Using SQL server as source and target, the user is provided with effective user interface screens to select appropriate fields and add some constraints. The tool outputs SQL queries. Users can edit the schema at the source and targets to suit the needs. The generated data can be directly fed into test systems without any modifications.

Primitive metrics are captured and provided to fine tune the output data by changing different input options through an iterative process. The tool also provides design of rules in an easily understandable tree view.

Design of the application made this tool easy for future enhancements without modifications to the basic architecture as described in chapter 6 on continuing work.

8. Bibliography

[1] E. Brown, *Windows Forms Programming in C#,* Manning Publications, 2002.

[2] IEEE Recommended Practice for Software Requirements Specification, IEEE Standard 830-1998 (Revision of IEEE Std 830-1993), IEEE Computer Society Press, 1998.

[3] C. Garrett, *GDI+ Programming: Creating Custom Controls in C#,* Wrox publication, 2002.

[4] M. Kay, *XSLT 2.0 Programmer's Reference, 3rd Edition*, Wrox publication, August 2004.

[5] P. Kimmel, *Advanced C# Programming*, Osborne publication, Sept 2002.

[6] D. Marshall, *Programming Microsoft Visual C# 2005: The Language*, Microsoft Press, 2005.

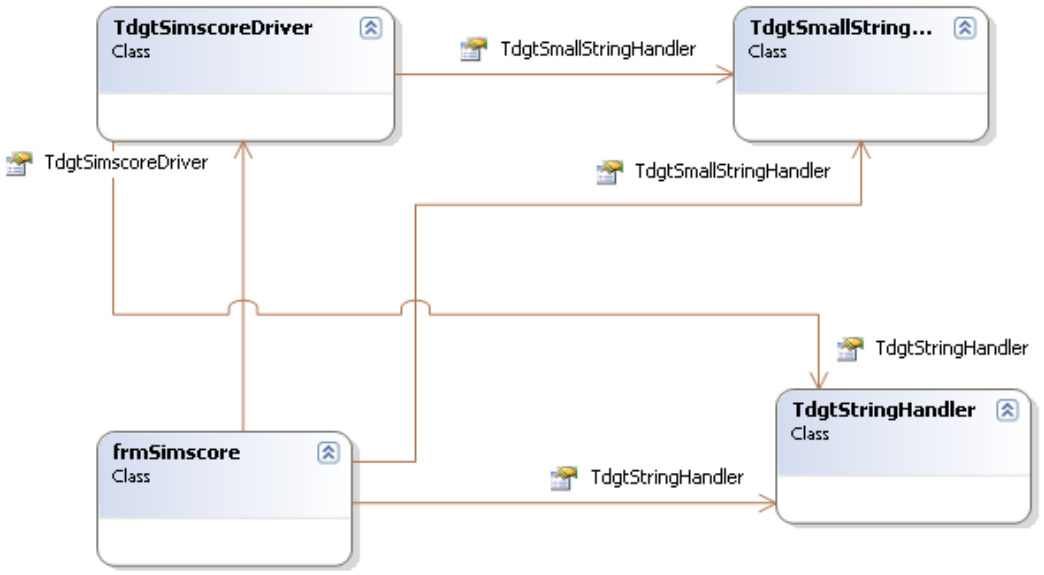
[7] I. Sommerville, *Software Engineering 6th Ed.*, Addison Wesley, 2001.

[8] http://msdn.microsoft.com/en-us/library/system.data.aspx, .NET Framework Library, Microsoft Corporation, 2008.

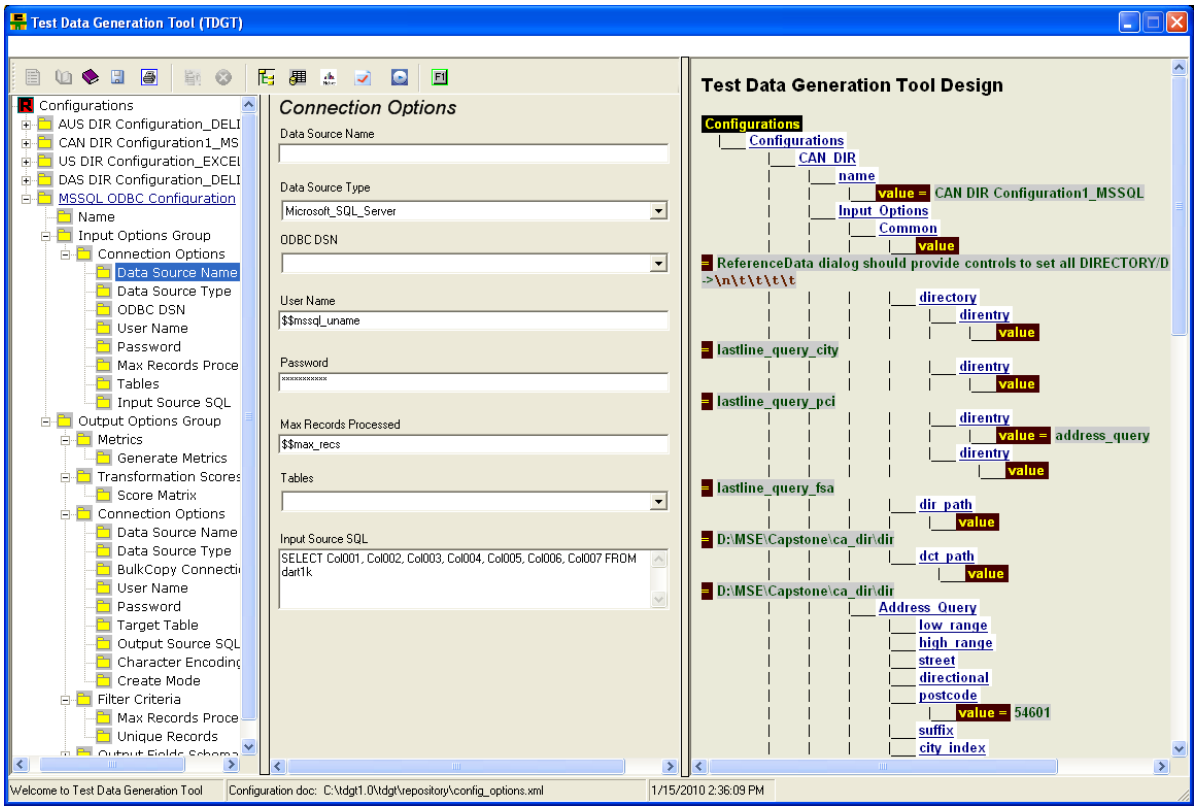
[9] http://www.softpanorama.org/SE/software_life_cycle_models.shtml, Life cycle models, Software Educational Society,

[10] http://qt.nokia.com/, Trolltech QT Framework, Cross-Platform application and UI framework.

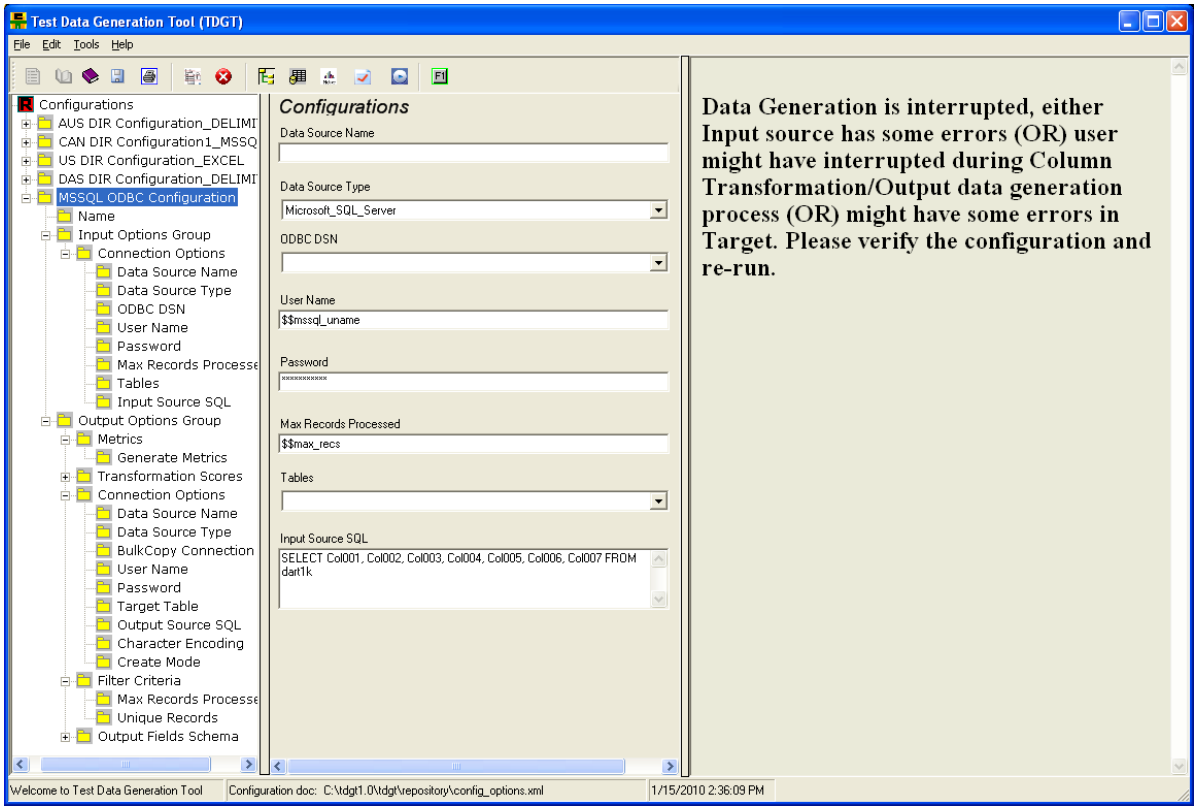
APPENDIX A: Selected TDGT 1.0 Screen Shots



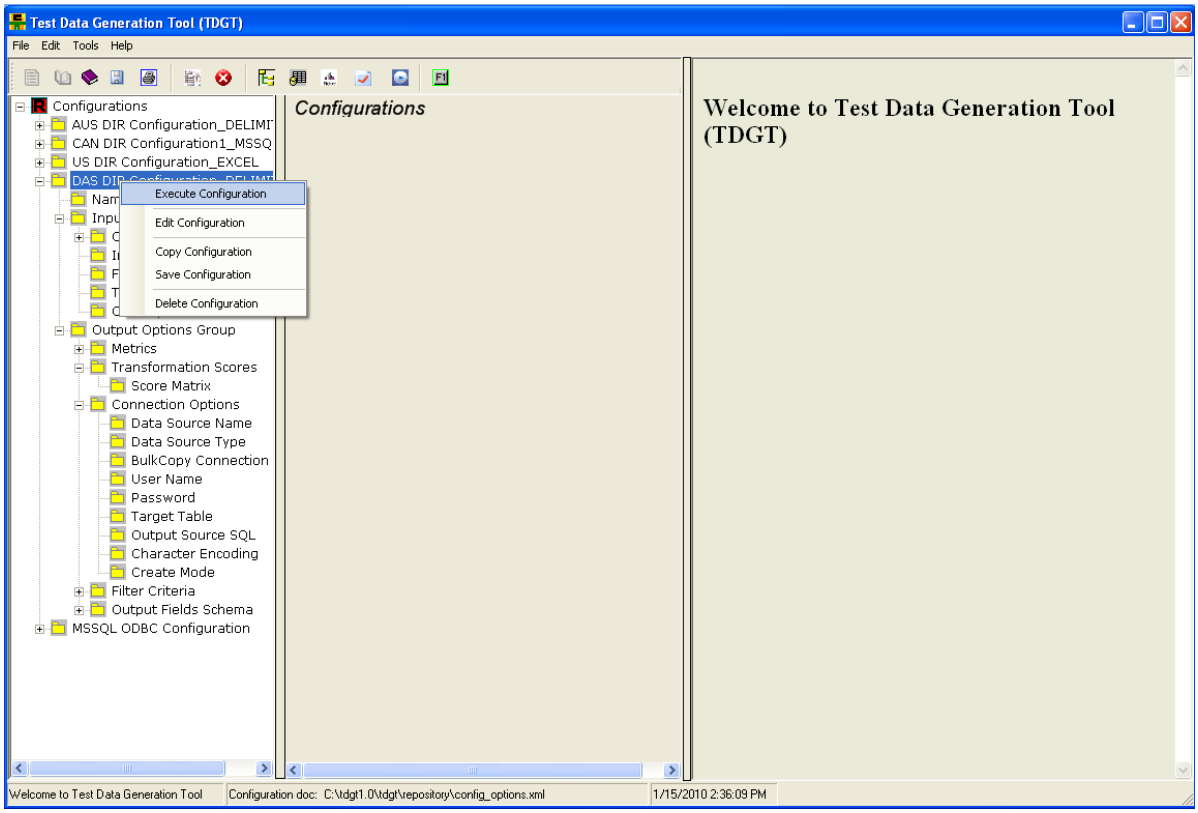
Class Diagram: Field Transformation through Match product and String Class library



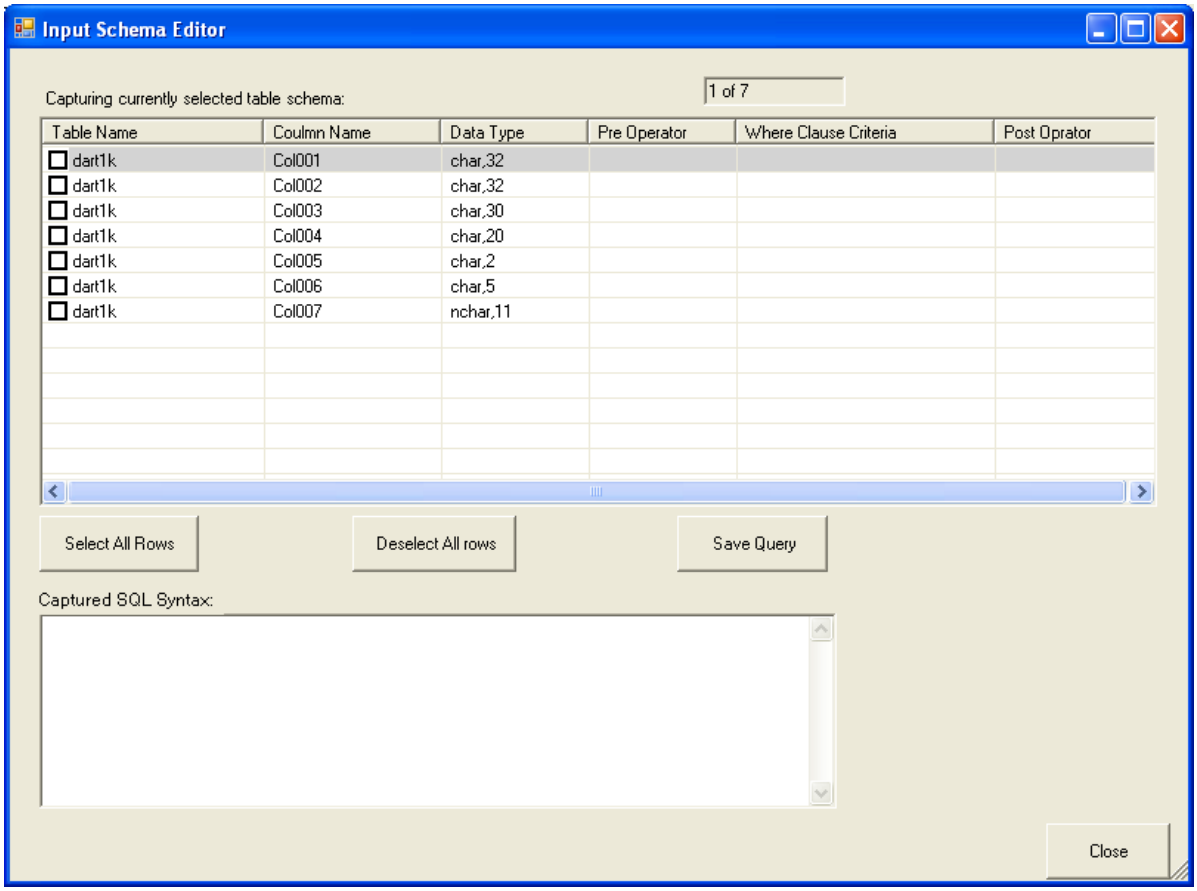
Main Application Window: Options and Design View



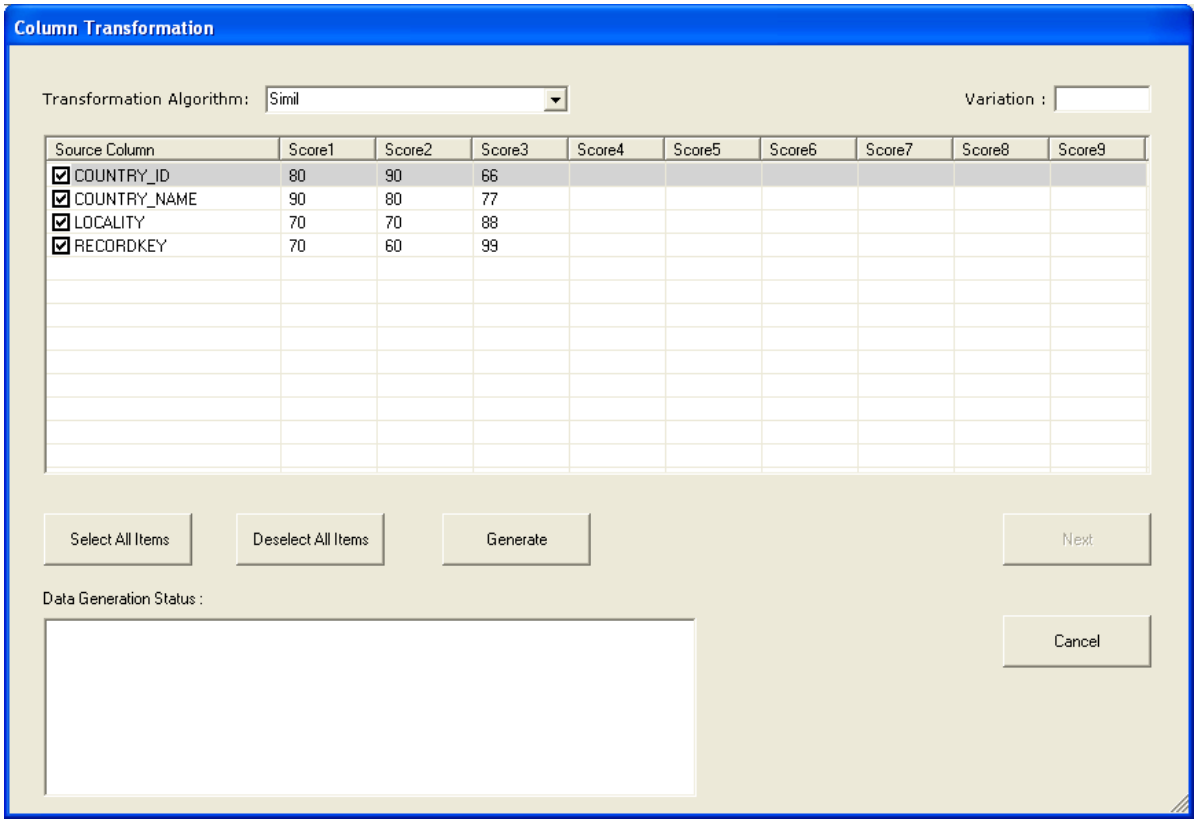
Main Application Window: Error message displayed in display panel



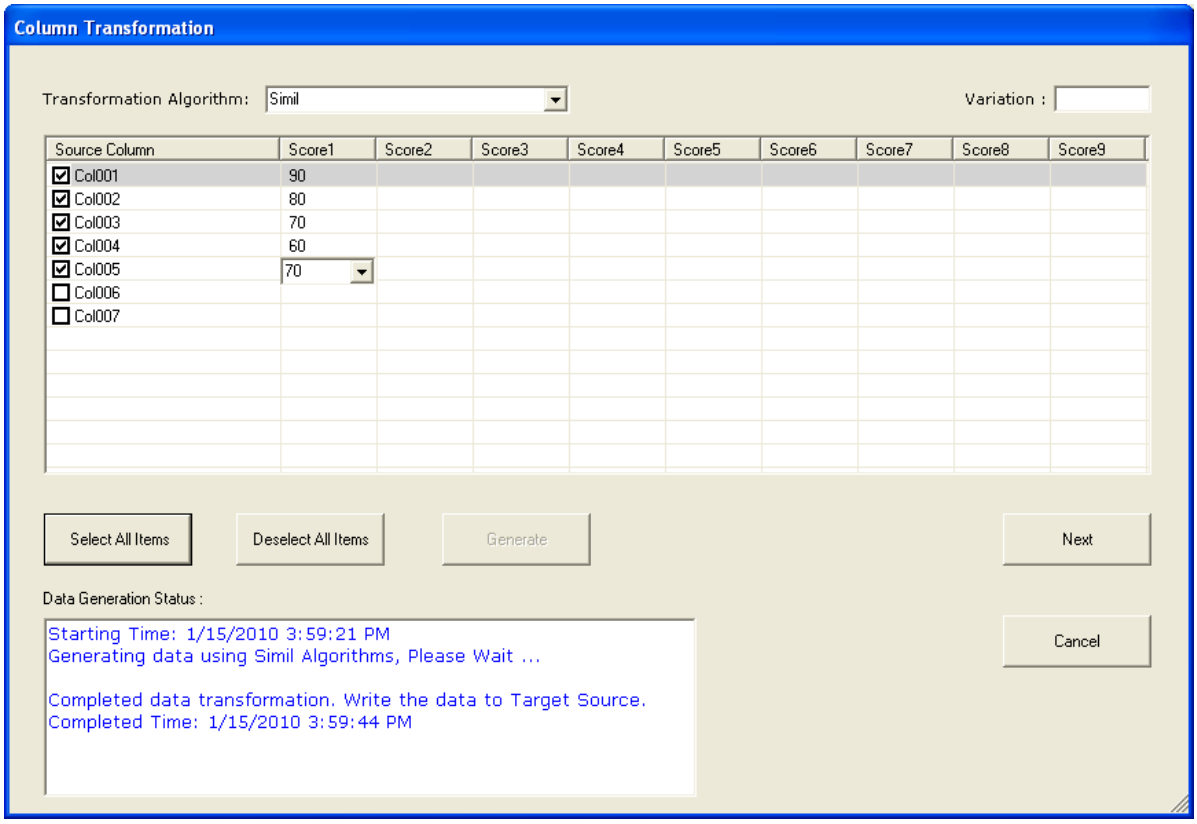
Main Application Window: Context menu options on mouse right click



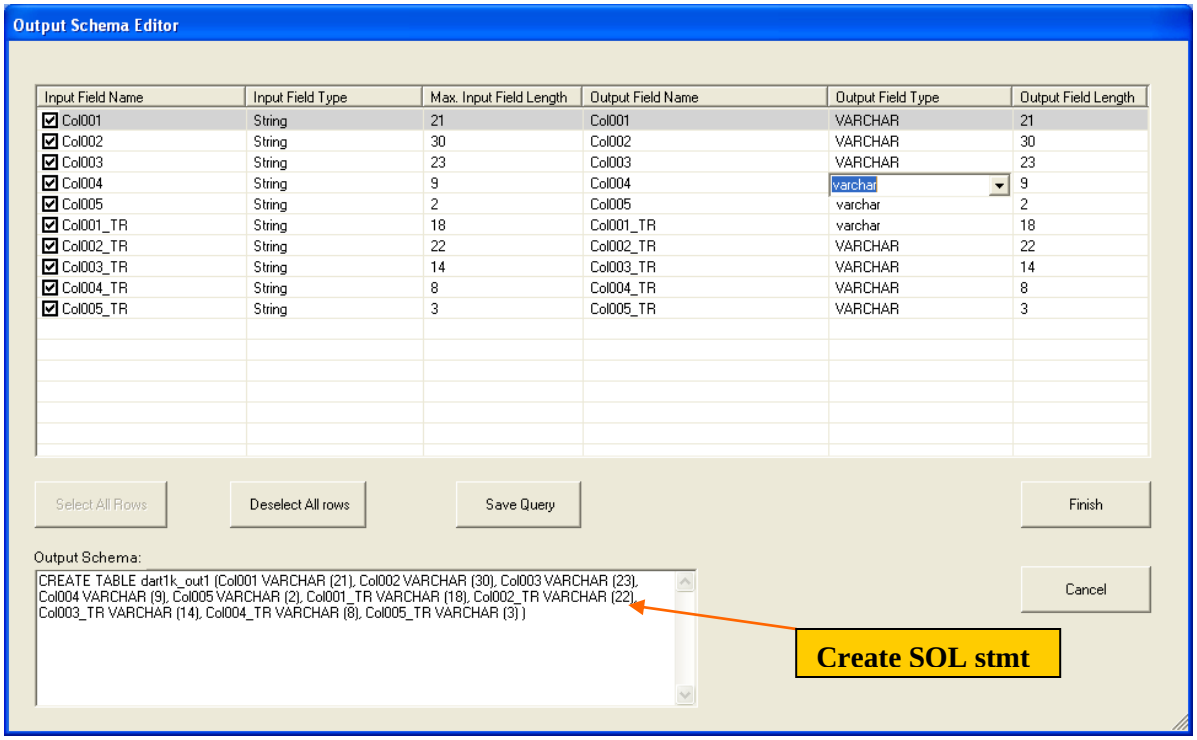
Input Schema Editor: Displayed when user selects Table in I/P options group



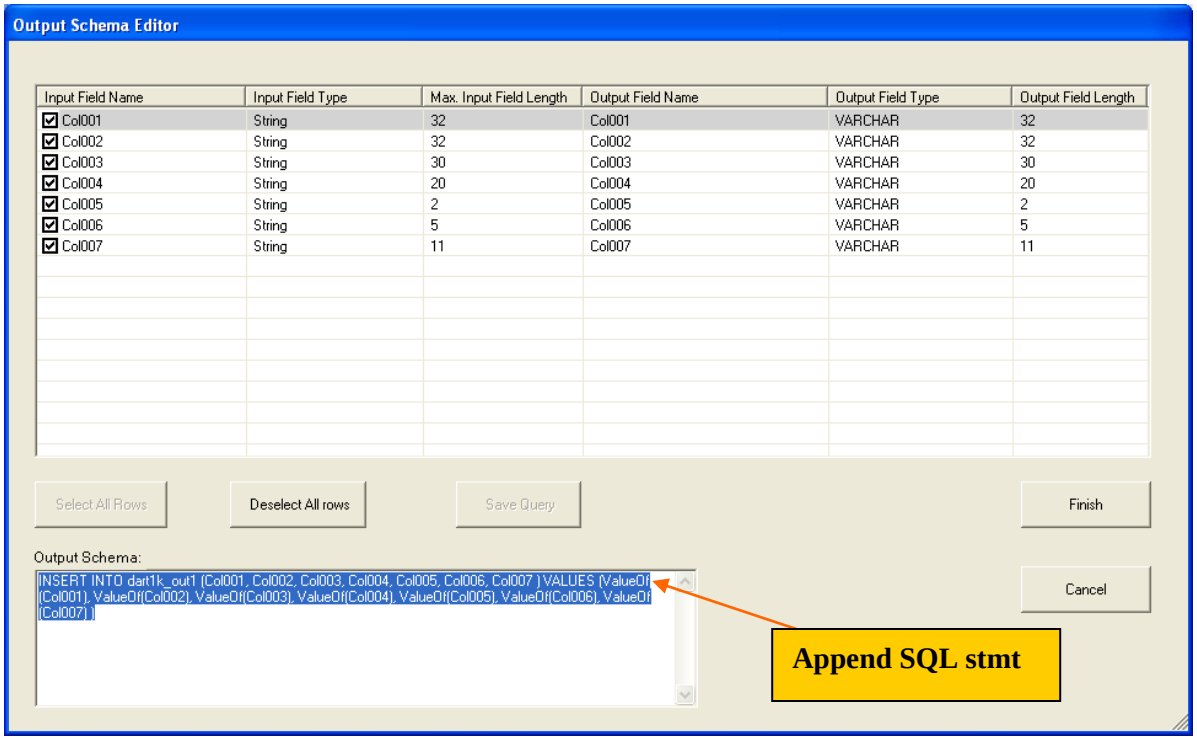
Field Transformation Window: User entered three sets of scores



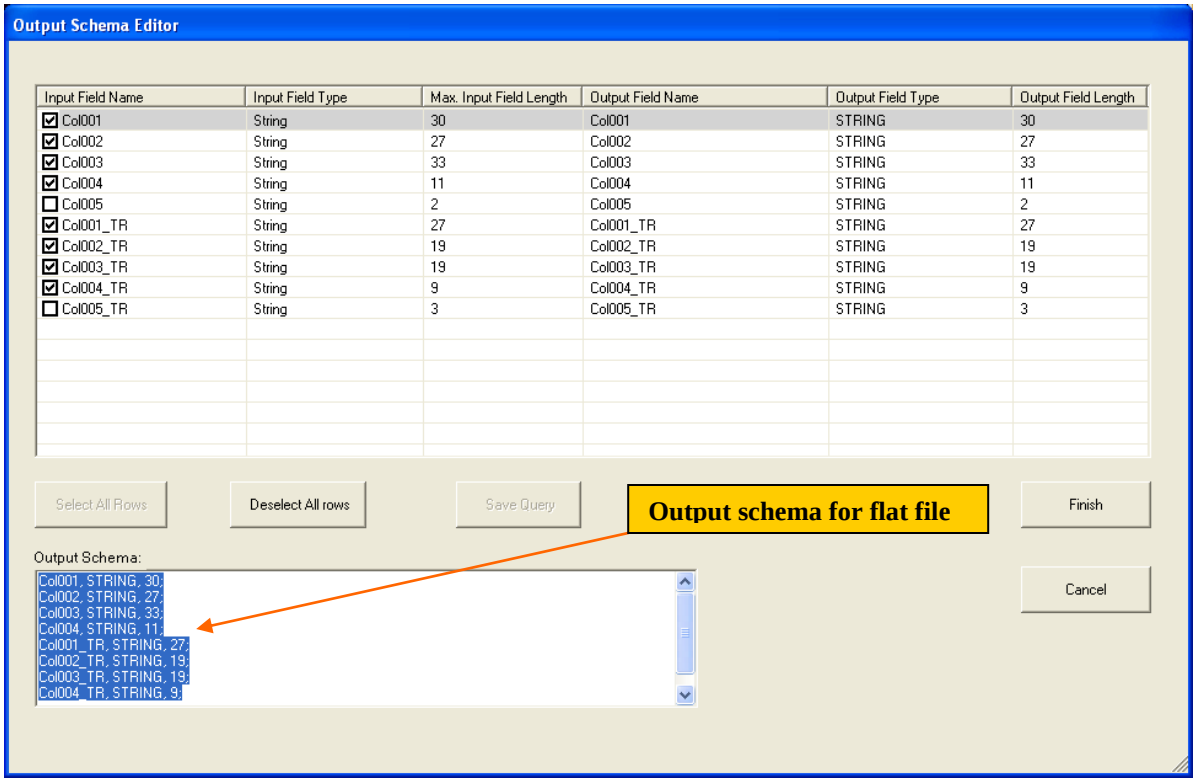
Field Transformation Window: Data generated message for selected fields



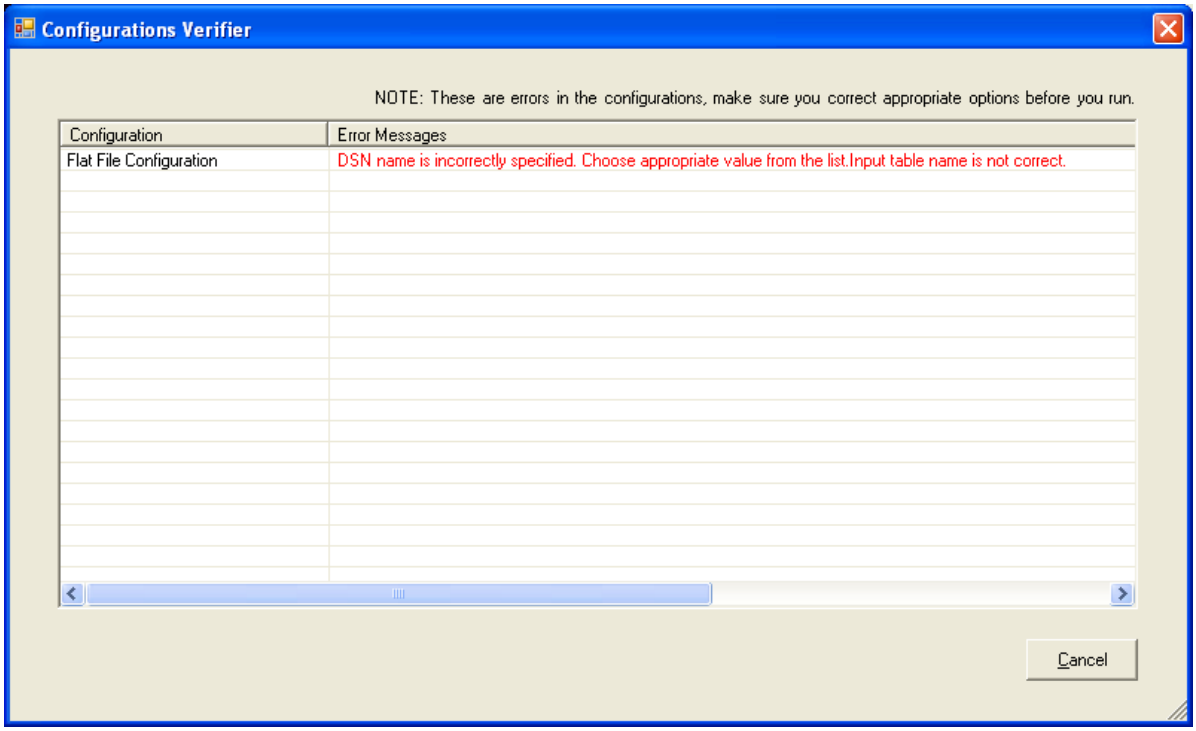
O/P Schema Editor: Edit schema for CREATE TABLE statement to override data



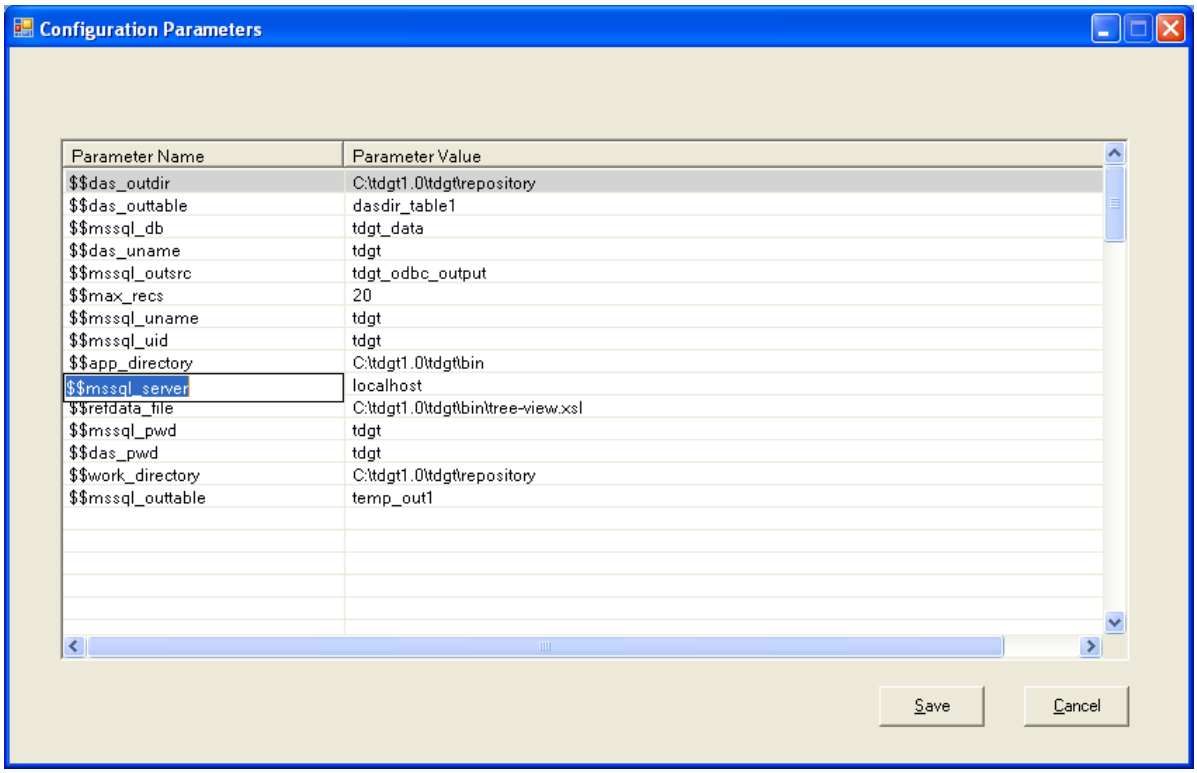
O/P Schema Editor: Edit schema for INSERT TABLE statement to appending data



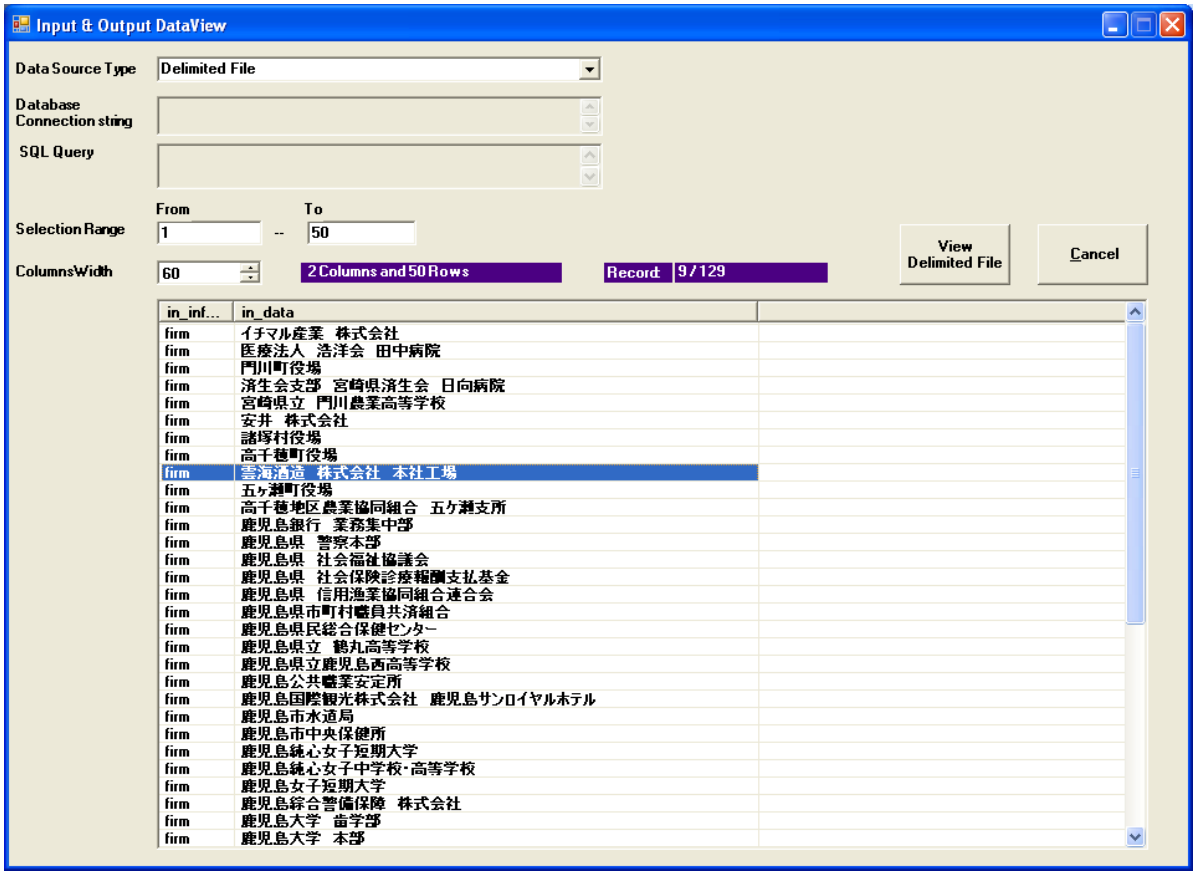
O/P Schema Editor: Output schema for Flat file output source.



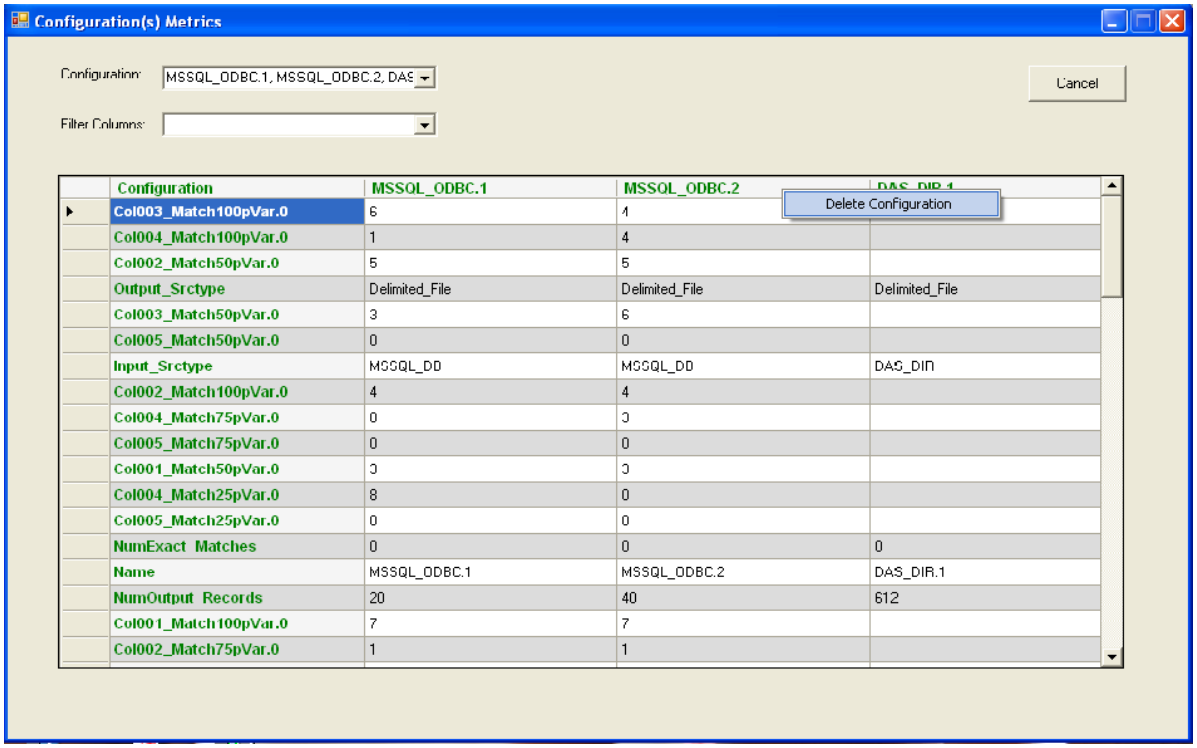
Verifier Window: Verifies all or single Configuration (‘s) and display error messages if any issues found



Configuration Parameters: Window where user can define substitution parameters used in configurations

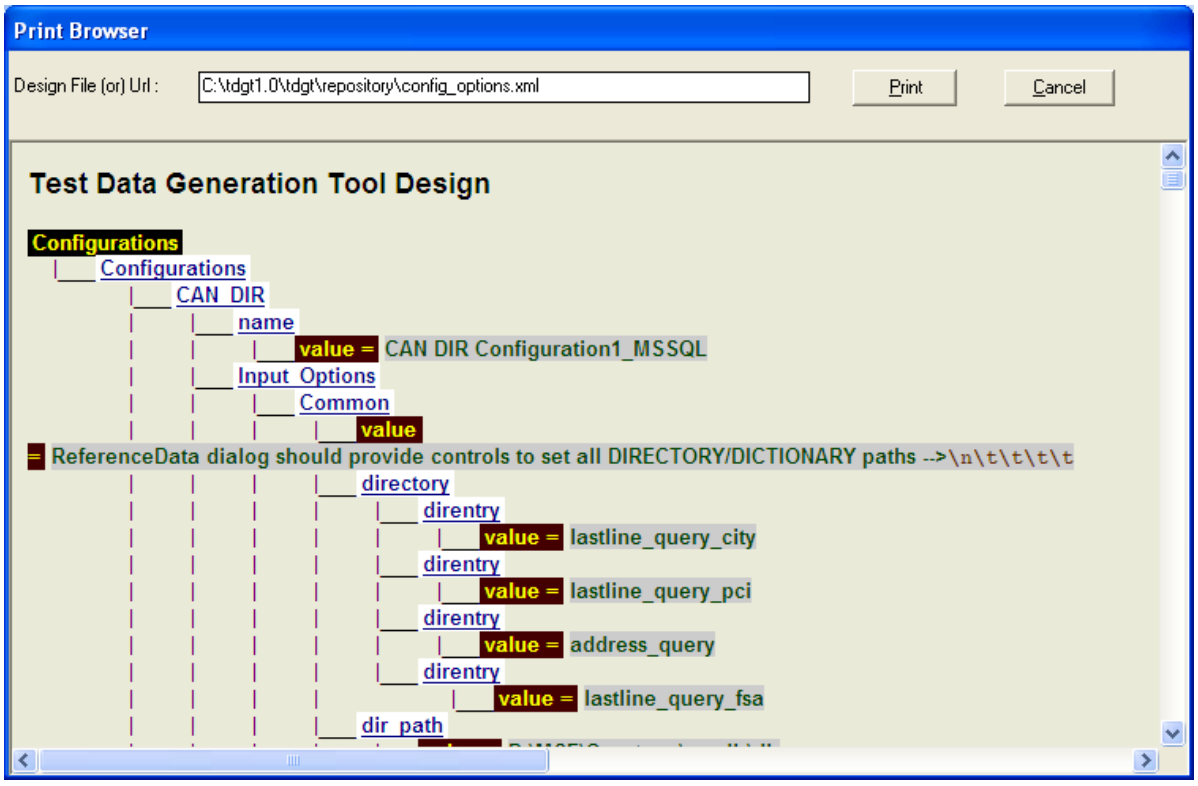


Input & Output Data View Window: View the Input & Output Data (ASCII/UNICODE) through this editor

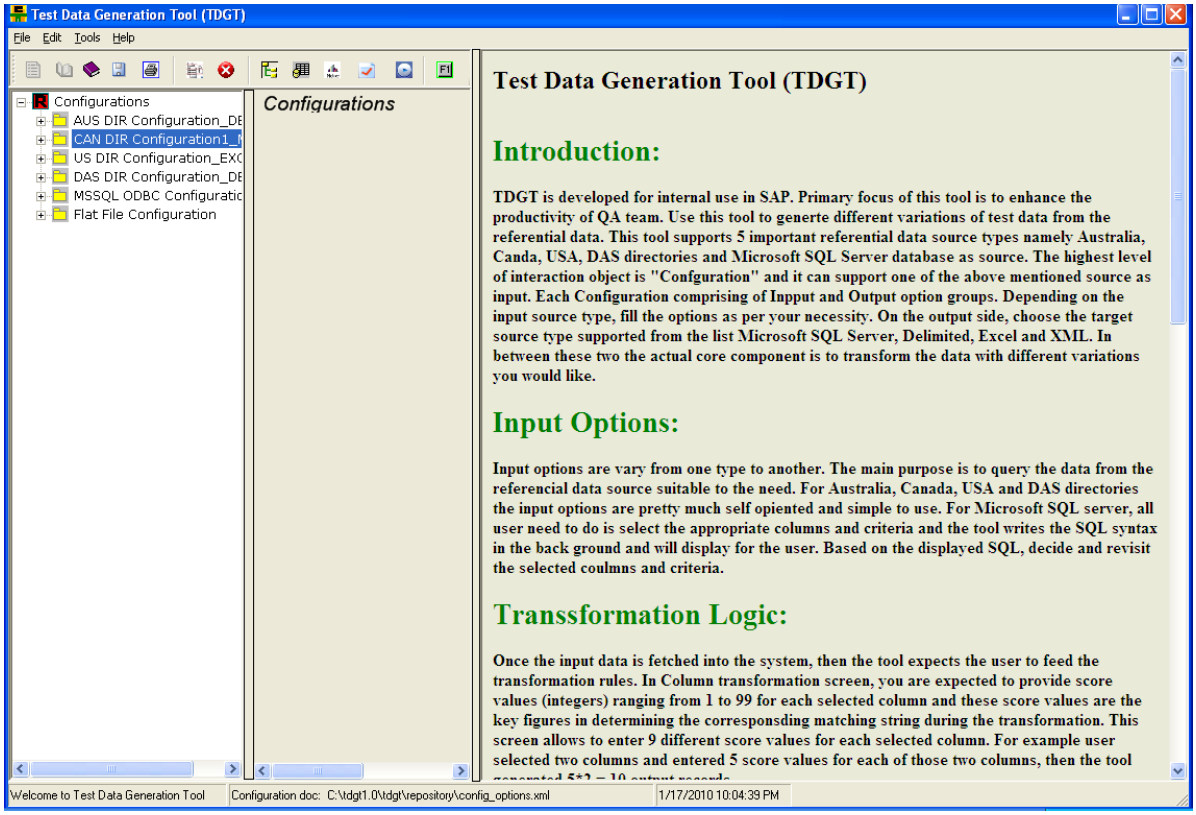


Configuration	MSSQL_ODBC.1	MSSQL_ODBC.2	DAS_DIR.1
Col003_Match100pVar.0	6	1	
Col004_Match100pVar.0	1	4	
Col002_Match50pVar.0	5	5	
Output_Srctype	Delimited_File	Delimited_File	Delimited_File
Col003_Match50pVar.0	3	6	
Col005_Match50pVar.0	0	0	
Input_Srctype	MSSQL_DB	MSSQL_DB	DAS_DIR
Col002_Match100pVar.0	4	4	
Col004_Match75pVar.0	0	0	
Col005_Match75pVar.0	0	0	
Col001_Match50pVar.0	0	0	
Col004_Match25pVar.0	8	0	
Col005_Match25pVar.0	0	0	
NumExact Matches	0	0	0
Name	MSSQL_ODBC.1	MSSQL_ODBC.2	DAS_DIR.1
NumOutput Records	20	40	612
Col001_Match100pVar.0	7	7	
Col002_Match75pVar.0	1	1	

Configurations Metrics: View three configurations metrics and the Delete Configuration option



Print Browser: Print the Configurations design in user readable format



Main Application Window: Display panel browser is displaying help for F1 button click