A .NET FRAMEWORK FOR RULE-BASED SYMBOLIC DATABASE VISUALIZATION IN 3D

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Edward J. Heyne

August, 2011
A .NET FRAMEWORK FOR RULE-BASED SYMBOLIC DATABASE VISUALIZATION IN 3D

Edward J. Heyne

Thesis

Approved:  
Advisor  
Dr. Yingcai Xiao

Accepted:  
Department Chair
Dr. Chien-Chung Chan

Committee Member
Dr. Zhong-Hui Duan

Dean of the College
Dr. Chand K. Midha

Committee Member
Dr. Kathy J. Liszka

Dean of the Graduate School
Dr. George R. Newkome

Date
This thesis introduces a framework to create versatile 3-dimensional visualizations to symbolically represent records in a dataset. It provides a description of the flexibility in this framework to handle special visualization needs of individual datasets without writing custom software. As a part of this framework there is a database engine, rules engine, template editor and a graphics engine. The rules engine allows for application specific rule definition that requires knowledge about the data. Rules are defined to show variations in the data through symbols. Additionally, the rules identify where in the 3D world the symbols are to be placed. The template editor also requires knowledge of the application domain in order to build specific 3D models (symbols) for a specific 3D world. Within the template, regions must be defined that are used by the rules engine to position the symbols appropriately within the 3D world. Only the rules and the template model need be substituted to alter such a visualization system to diverse applications.

A rule-based database 3D visualization system has been constructed based on this framework. As a test of this system, a 3D library was created based on a dataset of books. Rules and templates are defined to represent data about the books.
ACKNOWLEDGEMENTS

This work would not have been possible without the supervision and guidance provided by thesis advisor Dr. Yingcai Xiao, who not only helped with the topic of this thesis but also provided assistance throughout all phases of the project and providing support and encouragement to see the project through to the end.

I am grateful to all of those whom I have had the pleasure to work with during this project. Thanks go out to Tammy Stitz from the Science and Engineering library at the University of Akron for supplying the large dataset of library data. Thanks also to Michael Conrad in his assistance in the project implementation aspect. Thanks to Dr. Zhong-Hui Duan and Dr. Kathy J. Liszka for graciously accepting my request to read this thesis and participate in the defense of it.

No one has been more important to me in the pursuit of this project than my wonderful wife, Rebecca, who not only acted as an editor but also provided unending inspiration and whose love and guidance were with me throughout this endeavor.
# TABLE OF CONTENTS

**LIST OF FIGURES** .......................................................................................................................vii

**CHAPTER**

I. INTRODUCTION .............................................................................................................................. 1

II. BACKGROUND .............................................................................................................................. 4

   .NET Framework ......................................................................................................................... 4

   Windows Workflow Foundation ................................................................................................. 6

   Windows Presentation Foundation ............................................................................................. 8

III. .NET FRAMEWORK RULE-BASED SYMBOLIC DATABASE VISUALIZATION PROJECT . 10

   Project Goals ............................................................................................................................... 10

   System Architecture .................................................................................................................. 12

   Database Engine ......................................................................................................................... 13

   Template Editor .......................................................................................................................... 20

   Rules Engine .............................................................................................................................. 25

   Graphics Engine .......................................................................................................................... 36

IV. RESULTS AND DISCUSSION ..................................................................................................... 40
V. CONCLUSION AND FUTURE WORK ................................................................. 44

VI. REFERENCES .................................................................................................. 47
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The .NET Framework Programming Model</td>
</tr>
<tr>
<td>1.2</td>
<td>The System Architecture</td>
</tr>
<tr>
<td>1.3</td>
<td>Database Engine Main Window</td>
</tr>
<tr>
<td>1.4</td>
<td>Open Previous Session Dialog</td>
</tr>
<tr>
<td>1.5</td>
<td>Database Access Dialog</td>
</tr>
<tr>
<td>1.6</td>
<td>Change Data Source Dialog</td>
</tr>
<tr>
<td>1.7</td>
<td>SQL Connection Dialog</td>
</tr>
<tr>
<td>1.8</td>
<td>Data Mapping Dialog</td>
</tr>
<tr>
<td>1.9</td>
<td>Main Window with Data Set</td>
</tr>
<tr>
<td>1.10</td>
<td>ExecuteSelectStatementActivity Designer View</td>
</tr>
<tr>
<td>2.1</td>
<td>Application Folder Structure</td>
</tr>
<tr>
<td>2.2</td>
<td>Template Editor Window</td>
</tr>
<tr>
<td>2.3</td>
<td>Template Editor File Menu</td>
</tr>
<tr>
<td>3.1</td>
<td>Rules Engine Main Dialog</td>
</tr>
<tr>
<td>3.2</td>
<td>Rules Editor File Menu</td>
</tr>
</tbody>
</table>
3.3 Rules Engine Template Selection Dialog ........................................ 30
3.4 Condition Builder Dialog ............................................................. 31
3.5 Rules Editor Tools Menu ............................................................... 33
3.6 Assignment Property for Full Chaining ......................................... 33
4.1 Graphics Engine Main Window ....................................................... 38
4.2 Graphics Engine Object List Window ............................................ 40
5.1 Result Visualization of the library data ........................................... 43
5.2 Chaining Demonstration ............................................................... 44
5.3 Visualization of Chaining Rule Demonstration ............................... 45
5.4 Audio CD Object Listing ............................................................... 45
CHAPTER I

INTRODUCTION

Databases contain trillions of bytes of data. Often times the columns in database tables contain discrete data that do not have a lot of meaning when standing alone. Data visualization can help to identify this meaning. Communicating information clearly and effectively is the primary goal of data visualization. Visualization of data can make the significance of the data stand out in a graphical representation. Since the world to the human eye is seen in three dimensions, it only makes sense to likewise represent this data in three dimensions. According to Xiao (2001), database visualization is the mapping of “non-graphical data stored in databases into graphical elements for the purpose of visual representation and visual analysis.” The concept of “rule-based” is where the decision making is determined by a well-defined set of rules. Giarratano and Riley (2005), state that “rule-based systems represent knowledge in terms of multiple rules that specify what should or should not be concluded in different situations” (p. 34). They go on to give examples of rules being, “If there is a fire then leave” or “If my clothes are burning then put out the fire” (p. 17). Rules are of the format of IF some condition is true THEN take some action. This project brings all of these concepts together and discusses the technologies and integration strategies used to form a framework for rule-based database visualization in 3D.

Visualization of data is done every day using any one of the many spreadsheet applications that are available. Card et al. (1999), classify data into four types: Normal, Ordinal, Quantitative and Quantitative Spatial. The data that is easily represented in graphs as curves and surfaces in generic spreadsheet applications are the Quantitative and Quantitative Spatial types. According to Tan et al. (2006), qualitative attributes, should be treated more like symbols. For non-quantitative data, generic
spreadsheets can only display their statistical information through pie charts. But many real world applications require symbolic representations, such as virtual display of books in a library or groceries in a grocery store. Applications of this nature require specialized software to render the display. This thesis describes a framework and creates an application that can visualize any non-quantitative data symbolically. The data that is represented in this paper is the Quantitative Spatial types of data, which is not easily represented in a generic application. This paper describes the creation of an application that can visualize any type of data.

The .NET framework is a well-established and matured framework that is used worldwide and will be used for years to come to create academic and business software solutions. As widely used as it may be, there are no rules-based symbolic database visualization programs that exist today that will provide all of the features that this project does. The first feature is a flexible database interface that can be used with most major Relational Databases as well as some of the more common database engines like Microsoft Access. The second feature is a fully functional feature-rich rules engine with chaining capability. A third feature is the generic file formats that are used throughout the subsystems that can easily be ported over to other applications or converted. The final feature is the three-dimensional graphical representation presenting a third person view point thereby providing the capability to walk through the 3D world and view the data from many different perspectives. This project shows that the many different parts of the .NET Framework can work together to provide all of these features for implementing rule-base database visualization in 3D.

This system is broken down into four basic modules. These modules are the Database Engine, Rule Engine, Template Editor and the Graphics Engine. These four modules are implemented in three different software applications. The three different applications are the database/rule engine, the template editor and the graphics engine. This thesis describes in detail the technologies used and the design and implementation of each one of these modules. The next chapter gives a background overview
of the .NET Framework and the foundations that comprise it. It then goes into more detail for the areas that significantly impacted and played an important role for this project.

Chapter III entitled “.NET Framework Rule-based Symbolic Database Visualization System” explains the goals of this project and gives a description of the System Architecture and goes into detail about how to use each of the features in the applications and how the applications interface with each other.

The fourth chapter entitled “Results and Discussion” goes into the results of two different sets of test data. The results are discussed in detail and explanations are given about the results.

Finally the Conclusion wraps up the entire thesis and presents some future work.
CHAPTER II

BACKGROUND

2.1 .NET Framework

The .NET Framework is a software abstraction that contains generic functionality to create and execute applications on the Microsoft Windows operating system. Doke and Williams (2005) maintained that the .NET Framework consists of three central components, .NET compilers, the Common Language Runtime (CLR), and the Framework Class Library (FCL). .NET compilers are software programs that enforce specific programming language syntax as well as .NET language specifications. They generate Microsoft Intermediate Language (MSIL) code. The generated code forms an assembly, which can be executed by the .NET Framework. An assembly consists of modules. Modules consist of Types. Types consist of Members. The CLR takes the assembly and “Just-In-Time” compiles that MSIL code into native binary code that can be executed by the operating system. The CLR consists of three important pieces, the Common Type System (CTS), the Common Language Specification (CLS) and the “Just-In-Time” Compiler (JIT). The CTS defines the standard .NET data types used by all .NET languages. The CLS contains the general rules of a .NET language. The JIT is a compiler that takes the IL code and compiles it into native CPU instructions which can then be executed. As the name “Just-In-Time” denotes, this compilation occurs just before the code is executed. The third part of the .NET Framework, the FCL, is a collection of classes containing general functionality that can be used by .NET applications.

As part of the FCL there is a group of classes that are provided for the dynamic creation of assemblies, modules and types. These classes make up the .NET Reflection namespace. Reflection is important to this project in that it is used to allow for the Workflow Foundation rules engine to execute rules on objects that are created at run-time based on the data that is chosen by the user in the Database.
Engine. This is what gives this project the flexibility to allow the user to choose data from a number of different data sources for the data that needs to be visualized. Later versions of the .NET Framework starting with version 3.0, have yielded three foundations, which are Workflow, Communication and Presentation. The two that are exercised in this project are the Workflow and Presentation foundations. I will first go into more depth describing these technologies as this is pertinent to show their necessity. I will then go into detail describing how each foundation was implemented into the system.
2.2 Windows Workflow Foundation

Windows Workflow Foundation is a Microsoft technology for defining, executing and managing workflows. According to Shukla and Schmidt (2006), a workflow is generally defined as a sequence of operations representing work of a person, group of persons or machine. As it relates to software, a workflow is a series of tasks within some transaction that produce a final output. Windows Workflow Foundation is commonly referred to as WF.

WF is a reactive, distributed workflow architecture that supports thread agility and process agility. According to Shukla and Schmidt (2006), it is reactive in that it acts upon stimuli from external entities. It is a distributed application because it can execute on two or more computers. It supports process agility in that it supports the ability to pause and re-start processes and make dynamic changes to a process without having to start or re-start it. WF is able to achieve this without affecting existing running instances as well as by allowing the serialization and deserialization of workflows. WF supports thread agility by providing the ability to switch threads in the middle of processing a transaction. This allows a transaction to begin executing on one thread and to end on a different thread and still arrive at the desired output.

WF contains discrete re-usable components that are designed for a specific purpose in mind. These components are called activities. Activities are the fundamental unit of construction for WF. There are two basic types of activities. The first is a simple activity, which is responsible only for its own work. The second type, composite activities contain one or more activities as its children. Composite activities are responsible for ensuring the proper execution of its children. WF comes equipped with numerous activities that can be used by any application. Some examples of WF activities are code activity, flow control activities, state management activities, event handling activities and more. WF also allows for the development of custom activities which are used to solve specialized business problems.

Workflows in WF are a collection of activities that are executed in a defined sequence. These activities work together to perform some specific task. There are two basic types of workflows in WF.
The first is the sequential workflow. Sequential workflows have a clearly defined start and end point. They allow you to define a fixed sequence of execution steps. MacDonalds (2007), tells us more about the other type of workflow, which is a state machine workflow. A state machine workflow does not have a defined sequence of steps, but uses states. Each state can have one or more activities. These activities are triggered by events. When an event is received the states activities are executed before transitioning to the next state. For this discussion we will only concern ourselves with the sequential workflow as the activities executed for both database engine, rules engine and graphics engine have a predefined sequence of execution.

WF is not a stand-alone application. WF was designed to be hosted in the application in which it is being used. This is a desirable feature in that there are no restrictions on the type of application or application platforms that WF can be used on. WF is not an application itself but needs to be hosted by an application. The engineers designed the workflow runtime engine as part of WF to ease the pain of managing workflows. The host application is responsible then for incorporating the runtime engine and for creating and starting workflow instances. The host application can be a console application, windows forms application, windows service, web service or a web page.

The database engine, rules engine and the graphics engine use WF. The Template editor and graphics engine were more focused on the User Interface therefore used the Windows Presentation Foundation. No good use of WF was identified for the Template Editor, thus it did not utilize WF.
2.3 Windows Presentation Foundation

The Windows Presentation Foundation is commonly referred to as WPF. I will hereto forth use WPF to refer to Windows Presentation Foundation. WPF is a Microsoft technology that provides an API for creating rich and flexible graphical user interfaces (GUI) for applications that reside on the Windows operating systems. Nathan (2006) tells us that a WPF application can execute on a desktop or can be hosted in a web browser. WPF provides the means to create a declarative UI similar to using CSS and HTML in web development. According to Sells (2004), WPF provides for declarative programming by introducing the eXtensible Application Markup Language, commonly seen as XAML, pronounced “Zammel”. Nathan (2006) describes it as also providing the developer with the tools to create a rich UI using animation, 2D or 3D graphics and multi-media support. WPF has support for rich and flexible content layouts. It uses panels to accomplish this. Sells (2005) also tells of the basic panels built into WPF being the Canvas, StackPanel, WrapPanel, DockPanel, and Grid. In addition to control for layout, WPF also has a rich control library containing all of the common controls used to build user interfaces. The controls in WPF are very customizable. It is possible to use pictures or videos inside of dropdown lists, buttons and most other controls that WPF has in its library. Besides the basic panels, WPF provides a Viewport3D for the building of three dimensional worlds.

Figure 1.1 – The .NET Managed Programming Model
WPF is a .NET technology, which means it has all the features that come along with managed code, including memory management and error handling. However there are some aspects of the framework that are unmanaged. There are three primary code pieces to WPF. They are the PresentationCore, PresentationFramework and the milcore. The first two are managed assemblies. The PresentationFramework is responsible for the data-binding, animations, layouts, as well as other features. The next primary code piece in WPF, the PresentationCore contains the entire collections of base types used in the UI components. The third primary code piece, milcore, is unmanaged to provide integration with DirectX. The suffix “mil” is an acronym that stands for “Media Integration Layer”. Everything in WPF is displayed using milcore which uses the DirectX engine so that all UI can render richly, smoothly and efficiently. Direct3D is the technology that is used by DirectX which is the 3D engine used for most games. WPF makes 3D much simpler than DirectX ever did. It was at one time that games were the only place that 3D graphics or even 2D animations could be found. With WPF any application can easily use these features by expressing them in either XAML or in code. WPF is also capable of including WinForms (.NET 2.0 Windows Forms) controls inside of WPF controls making it backward compatible.
CHAPTER III

.NET FRAMEWORK RULE-BASED SYMBOLIC DATABASE VISUALIZATION SYSTEM

3.1 Project Goals

The motivation for the project came out of a 2D rule based database visualization framework that was developed by Dr. Yingcai Xiao et al. at the University of Akron. Dr. Xiao was the driving force behind the development of a similar .NET solution that would provide a rules engine that would support creation of rules, management of rule sets, dependencies of rules, chaining of rules and the re-evaluation of rules. Another objective of this project was to provide generic file structures that could be easily ported over to other applications. To make this rules based database visualization project stand out above any other, it was decided to provide the visualization of the data in three dimensions instead of the typical two dimensions. Given all of these requirements, the .NET framework was an obvious technology to use since it had support for all of these aspects: Rule Engine (WF), Generic file structure (XAML), and support for a three dimensional graphical user interface (WPF).

One of the goals of this project was to create a solution that utilized the .NET Framework. The above mentioned 2D Framework was written in the Java programming language. The decision was made to use the .NET Framework because it is a very powerful and well established framework that provides support for development on both a client and server based platform. This paper describes the client side solution with the forethought of porting this over to a web-based platform. This could be accomplished utilizing the Silverlight technology which is also built on the .NET Framework.

A second goal was to display the visualization in three dimensions instead of a two dimensional flat world. This was done to provide a more meaningful visualization of the data. The third dimension
would enable the showing of more data and the relationships of the data as it relates with other data and the world that it exists in.

A third goal was to utilize a standardized file format for the output. Though the database is able to query most database management systems, the data output for all modules of this platform utilize a standard XML file format. To take it a step further the .NET Presentation Foundation utilizes the XAML file format for a textual representation of the graphical user interfaces. Likewise the Workflow Foundation also utilizes a form of XAML that represent sets of rules. This file format was utilized in this framework to satisfy the goal of a standardized file format, whereas the previous Java 2D solution did not provide this.

The final goal was to incorporate a true rule engine. The Java 2D solution’s rule engine consisted of complex and clever SQL queries to retrieve the data. Whereas the usage of SQL is second nature to a developer and executes very fast it does not provide for the true capabilities that a rule engine would. This platform supplies a UI to generate rules and these rules are then executed in the WF Rules Engine as actual rules. This allows for the capability of forward chaining to be used, which is a concept described in later sections.
3.2 System Architecture

The design of the project called for three very important parts working together to form the final visualized data. The first part of the design is a Database Engine. The purpose of the Database Engine was to allow for a robust and extensible framework that could retrieve data from any source. The second piece is the Rules Engine. The Rules Engine utilizes the WF rules engine and utilizes the chaining behavior and rule execution variations that the WF rules engine supports. The final piece of the design is a Graphics Engine. The Graphics Engine utilizes the WPF GUI API and implements the visualization in three dimensions.

A .NET Architecture for Rule-based Visualization in 3D

Figure 1.2 – System Architecture

As a supporting tool for these three main sections, it was necessary to develop a Template Editor that would allow for the customizable creation of a template that would be used for the three dimensional visualizations. The following sections go into more detail regarding each of the parts of the project.
3.3 Database Engine

The Database Engine is a basic WinForm application with a very basic user interface. The main window consists of a menu bar for initiating the commands and a DataGrid that will hold the data returned from the selected database.

![Database Engine Main Window](image)

Figure 1.3 – The Database Engine Main Window

This is the main window of the entire project. Everything must start from this window. The Template Editor, Rules Engine, Execution of Rules, and the Graphics Engine can all be invoked from the Tools menu of this window. But before anything else can start, the first step in the process is to gather the data. The Database Engine has only one option not including exiting. This one option is Open. The open command provides a means for the user to gather the data either by opening a previous session or by stepping them through a series of dialogs gathering specific details of the data. If there is a previous session that is available, the “Open Previous Session” dialog appears.
This dialog allows the user to make a selection of a previous database, table, and column selection and open that rather than having to re-select each piece of the dataset over again. This is a feature that expedites the database open process if the user is found re-opening common database/table/columns combination. This data is read from a Microsoft Access table where the session data is stored. To avoid duplication a new session is not saved when the user selects to open a previous session, but only when a new database definition is used will the session data be persisted to the database.

If the user decides not to open a previous session then they are given the option to select the database, table and columns to use. The first dialog in this process is called the Database Access dialog.
The first detail that is gathered is the data source. There are four primary data sources that are supported. The four data sources are Microsoft Access 97-2003, Microsoft Access 2007, Microsoft SQL Server (MSSQL), and MySQL Server. Once a data source is selected a data provider must then be chosen. Most data sources have only one choice for a data provider, others have multiple choices. The Microsoft Access 97-2003 has three possible data provider choices. They are OLE DB 4.0, OLE DB 12.0, and ODBC. The Microsoft Access 2007 allows only the OLE DB 12.0 provider. The MSSQL source allows only the MSSQL Server provider and the MySQL source allows only the MySQL Server provider. Once a data source and provider are chosen, depending on the data source selected, different dialogs are presented. If either of the Microsoft Access choices are selected the user is taken back to the Database access dialog where they must enter the file name of the Access database. The browse button aids in the database file discovery process and shows an open dialog box where the user can navigate to the desired Access database file. Once a database file is chosen it is displayed in the Database file name box on the Database Access dialog. If the user selects either the MSSQL or MySQL options they are presented with the Connection dialog.
In this dialog the user is expected to know the name of the server where the database resides, the username and password for the server and the database name. When all of these fields are entered and the ok button clicked, the user is taken back to the Database Access dialog and the server name and database name appear in the Database server/name text box. Once the user has completed the database access entry, they are presented with a list of the tables that exist in the database. At this time the selection of only one table is allowed. After the table is selected the Data Mapping dialog is presented.

This is where the user must select the columns that they desire to use for the data mining.

After all of the selections for the dataset are made, the data is run through what is referred to as the interrogation process. As part of this process the total record count is retrieved and is multiplied by a factor of one half of .1 percent (.0005). This is done to find common values among the given fields. For those columns that have less than or equal to this number of unique values, the values will be captured and stored to later be used in the rule creation process. These unique values are then stored in the session database. This is done to allow for the creation of the rules to be more efficient by supplying a list
of values that can be selected from. This will be described in more detail later in this document. There are some rules that are followed to expedite this interrogation process. For instance, if the column is a primary key, then the interrogation is not done for that column. Also, if the field is indexed then there is no interrogation done. The assumption for this tool is that there will be large sets of data and there is no sense in providing only 25 values when it is known that there will be probably thousands of unique values in the dataset. After the interrogation process is performed the persistence of the session is done. Finally a query is executed against the identified table and the data is retrieved and populated in a data grid on the main window.

![Figure 1.9 – Main Window with Data Set](image)

At this time the user may view the data in the table as is loaded or by selecting one of the columns to sort the data. The data can be sorted for each column in ascending or descending order. This is the data the rules engine will use to execute each rule against.

The open dataset process uses a workflow, DataSetDefinitionWorkflow, to appropriately handle the chosen meta-data so that it can provide the desired results. The workflow is an extensible design using out-of-the-box activities as well as custom activities. This workflow requires numerous UI input in order to gather the meta-data about the dataset so that the desired data is provided. Workflows by default are executed asynchronously. The UI in this example is also the host of the workflow. To handle the interaction between the workflow and the host, the usage of an external data exchange service is needed. WF has out-of-box activities to allow the host and the workflow to communicate via this external data exchange service. These activities are called the CallExternalMethod and HandleExternalEvent activities. These activities are used throughout this workflow for each dialog that is presented to the user.
One particular custom activity that was created for this workflow is the ExecuteSelectStatementActivity activity. This activity is a composite activity and contains other custom activities. Each data provider has its own custom activity to implement the specific logic for reading the table schema and the data.

![ExecuteSelectStatementActivity Designer View](image)

Figure 1.10 – ExecuteSelectStatementActivity Designer View

Figure 1.10 shows what the workflow designer looks like for ExecuteSelectStatementActivity activity. An out-of-the-box IfElse activity is used and depending on the branch of execution a different custom activities for getting the data is used. This activity is used numerous times throughout the workflow for interrogation process as well as for getting the final dataset.

The workflow also includes a custom activity called GetColumnsFromDataTableActivity which, if given the data table, will supply a hash table of the columns that are in the table. This makes it possible for the user to see what columns are available and select those that are pertinent to the visualization.

There is also a necessity to build a dynamic database column type, module and assembly. As part of using the WF rules engine the rules engine needs to have an object that contains the data that it needs to check conditions on. Because we are interacting with dynamic data from a database, this object will always be different for every data source. The data is always changing and will be based on the chosen database, table and columns. As part of this workflow we have the BuildDynamicDatabaseColumnTypeActivity custom activity as the last step in the workflow. This custom activity uses reflection to build a dynamic assembly which contains a dynamic module which contains a dynamic type which contains dynamic
properties representing the columns of the dataset. A later discussion will address how an instance of this type is created and populated in the Rules Engine.

There are some possible future enhancements to the database engine. Currently the database engine only supports the selection of a single table. One future enhancement is to allow the selection of multiple tables and therefore allow more complex queries to be executed. The database engine also does not implement the Oracle database interface.
3.4 Template Editor

The next application in the chain is the Template Editor. This program allows a user to create and manipulate the elements of the virtual world, including the background and category regions. 3D objects in XAML format are currently supported by the engine. The Template Editor is an independent application that supports standard GUI functions, such as panel hiding and pane resizing, as well as basic editing functions, such as cutting, copying and pasting. The Template Editor also supports file saving (to XAML format), and opening. It’s possible to run the template editor without going through the main window, as this piece of the global application only deals with .xaml template files. Since these templates are not tied to a specific dataset, one well-designed model could support an infinite number of distinct datasets.

The Template Editor contains a Viewport3D which allows a user to define any type of a three dimensional model in XAML and use it as the background to the 3D visualized world. The Template Editor does not support the creation of the 3D models, but there are numerous XAML editors that already exist where the 3D objects can be created. The Template Editor searches the Tools\3d\meshes folder (see figure 2.1, B) underneath the application execution folder for any XAML file. These files should contain the Viewport3D object with whatever is desired to display as the background objects. The template editor reads in the Viewport3D and moves all child objects including lighting into the Template Editor’s main window Viewport3D. The only child object of the file’s Viewport3D that is not pulled in is the camera as the Template Editor’s Viewport3D will have its own camera defined. This feature allows for the user to get as creative as they want in developing the 3D world.

The Template Editor is also where the user can define regions to represent categories of objects. Regions contain a label, location, and scale value that will be saved in XAML format to be read in later by the rules and graphics engine. Each region can be scaled independently, with independent X, Y, and Z axis values if desired.
Under the 3D Tools panel will be the different XAML elements that a user can add, which will later be assigned to regions via the Rules Editor. Just like the background meshes, the elements can be any 3D object. The Template Editor searches the Tools\3d\elements folder (see figure 2.1, A) from the application folder for XAML files (see figure 2.1, C) to find the available objects. These XAML files must contain a Viewport3D object. The Viewport3D object can contain any number of child objects. Once again the user can define these objects in any of the existing XAML editors and copy the Viewport3D control into a text file and move the text file to the Tools\3d\elements folder so that the Template Editor can find it. For each file that exists in this folder there is a button shown in the Tools pane (see figure 2.1, D) to represent that object.

![Application Folder Structure](image)

Figure 2.1 – Application Folder Structure

The name of the file is the name that will be used on the button so it is wise to name the file well to represent the object that is contained within the file. Just like the background mesh the camera of the element’s Viewport3D is ignored. Different from the background mesh, the lighting is also ignored. The lighting will use the background mesh’s lighting. The ability to customize the elements to anything the user desires is a feature which demonstrates how fully customizable this application is. The user is only restricted by WPF in what types of 3D objects can be created.
The Template Editor main window allows the users to move each one of the regions and the background mesh to their desired locations. It also allows the user to show the environment from different camera perspectives so that the user can view what the 3D world will look like from different angles. A slider in the status bar (see figure 2.2, A) lets a user zoom in and out to help them get a feel for how their final environment will look. Also the selection of regions allows them to be renamed to more accurately represent the data that will be contained in that region. There are also sliders in the property pane to allow for the scaling of regions, in the event that the pre-defined size is not exactly the size desired in the template.

The Template Editor also allows for the selection of the above mentioned objects that need be included in the template. The objects selected will show in the lower pane (see figure 2.2, B) which runs horizontal below the main window. These objects can be customized from within the Template Editor allowing for the user to select the same object numerous times and give it different looks for each instance of the object. The properties that can be customized include the Background, where the user can select either a color or an image to paste on the element. The Font is customizable as well as the Foreground color and the Label text. The user has the ability to rotate the element so that the positioning of the object in the world may be adjusted to show the appropriate face of the element in the graphics engine. Another feature that was added was the ability to scale the object as it might be necessary to show that one object is larger or smaller than another. The rotation and scaling features are shown in Figure 2.2 D.

The last feature to speak of in the Template Editor is the ability to modify the cameras two properties, position and look direction. Figure 2.2 E shows that in the left pane below there are two groupings of buttons. The first group is the Position group where the X, Y and Z of the cameras position can be adjusted. Likewise the second group of buttons provides the ability to adjust the cameras X, Y and Z look direction. The ability to adjust these camera properties provides the user the ability to view all angles of the template to see how the template appears from that angle.
Once a template has been created, the user can save the template. This action will cause the entire template with background mesh, regions and objects to be serialized to the XAML format to a user given file name and stored for later use in the rules and graphics engines. The templates are stored under the \Templates folder located in the application base folder. To edit an already existing template the template can be opened from the File menu and will be presented in the same way it was saved previously and can be modified in the same fashion as it was created originally. Figure 2.3 also shows an “Update Mode” option. In the situation where you have an already existing template and the model that was used in that template has been modified, this option can be used to pull in the modified model without impacting the objects or the regions. This enables the same template to be used with an updated model without needing to recreate the entire template for a newer version of the model.
Figure 2.3 – Template Editor File Menu
3.5 Rules Engine

The WF contains a rules engine which provides a straightforward model to defining rules. Rules engines in general have been more typically known as a specialized technology only used by highly specialized groups. Most rules engines require domain experts to develop the rules. Since Microsoft technologies are widely known as the choice for developing business applications, it made sense to use a rules engine developed by the same community. The WF rules engine is simple to grasp and easily integrated into any .NET application. Even though it may be relatively simple to understand and implement, it still provides an adequate evaluation engine that supports complex rules. WF rules engine also includes rule evaluation features such as forward-chaining and priority-based execution.

The two most popular methods of reasoning are forward-chaining and backward-chaining. Before understanding what either one of these is, it’s important to understand what chaining is. More generally, chaining is the dependency of one rule on another rule. More specifically, Griffin. and Lewis (1989) tell us it is the dependency on the actions of rules to the conditions of other rules. For example, rule2 has an action that modifies X. Rule1 has X as part of its condition. Based on alphabetic rule evaluation rule1 would be evaluated first followed by rule2. Rule2 modifies X which is used in the condition of rule1. This would force rule1 to be re-evaluated. This is an example of forward-chaining. MacDonald (2007) tells us that there is also the backward-chaining reasoning method. This is generally the idea of taking a hypothesis and working backward to determine if there is data to support the hypothesis. An example of backward-chaining is a medical diagnosis where you start with the symptoms of the problem and work backwards to prove the hypothesis of a diagnosis.

Rules are thought of as basic statements of facts about the data. WF organizes rules into rule sets which are merely logical groupings of rules. WF supports the ability to apply dependencies on rules to other rules. This dependency is important to understand in order to realize the actual order of execution of the rules when using methods of reasoning such as forward-chaining which is discussed in greater detail in another section.
Rules in WF are composed of three parts: condition, then action, and else action. This structure is one of the fundamental control structures that exist in most if not all programming languages, the if-then-else construct. Rules are executed by first evaluating the condition part and determining a Boolean result of either true or false. If the condition evaluates to true the then actions are executed. If the condition evaluates to false the else action is executed. Each rule is required to have a name. The name is assigned by the user defining the rule. Rules are evaluated by default in WF in alphabetical order by the name of the rule. Rules can also be given a priority level. By default the priority level of a rule is 0. This level can be changed to any integer value. Rules with higher priority levels will be evaluated first before rules with lower priority level. Rules with the same priority level will be evaluated by the alphabetic order of the name of the rule. Setting the priority level or naming the rule based on desired order of evaluation is not usually necessary when forward-chaining reasoning is used. The forward-chaining reasoning is on by default, but can be turned off if desired.

There are three different ways the rules engine discovers dependencies when forward-chaining is turned on. They are implicitly, attribute-based, and explicitly. Griffin and Lewis (1989), demonstrate that implicit dependencies are discovered by the rules engine automatically. The rules engine will assess each rule and check to see if any rules actions modify properties that may be used in another rules condition. The rules engine will ensure that each rule that has this dependency is re-evaluated when the property used in the condition is modified. The attribute-based discovery mechanism uses a more brute force method to decide on rule dependency. This discovery mechanism is used for when methods are invoked as part of the then or else actions. Three attributes exist that may be applied to the method to help with dependency discovery. These three attributes are RuleRead, RuleWrite, and RuleInvoke. The RuleRead attribute indicates that it reads the indicated property. The RuleWrite attribute indicates that it modifies a given property. RuleInvoke attribute can be used to indicate that one method call is linked to another method call. Explicit dependencies are discovered by using a specific update statement. This update statement would require a specification to the path of the property that is getting updated. Only when
the update statement is used on a specific property will the dependency be in place and cause a previously executed rule to be re-evaluated if that property is used in the condition.

Rule sets are expressed declaratively. They are expressed as CodeDOM (Code Document Object Model) which is stored in a XAML file with the extension of .rules. Having the files declared in a XAML document means that the rules can be modified by any text editor as long as the editor uses appropriate CodeDOM syntax. This is what makes the execution of rules dynamic. The .rules file can be modified while the rules engine is executing and the rules will be updated without re-starting the rules engine. CodeDOM stands for Code Document Object Model.

Once the database table is loaded in the main view then it’s time to generate the rules. The rules engine is combined with the database engine in the same application because it is necessary for the rules engine to know the table and the columns that the rules will be executed on. In addition, the rules engine performs queries against the dataset to find the possible values for a given column in order to provide a better user experience for defining rules. The rules engine is accessed through a menu option on the Database Engine window. Under the Tools menu the Rules Editor option can be found. Clicking this menu item will invoke the Rules Editor dialog as seen in figure 3.1.
Figure 3.1 – Rules Engine Main Dialog

Since the database engine and the rules engines were the first pieces of the project that were written they are still using the WinForms technology. This is due to the fact that the early design of the project did not include WPF. The reasons it did not include WPF are because WPF was relatively new and not a very mature technology and it was originally thought that managed DirectX could be used for the graphics aspect of this project. Because managed DirectX became no longer supported by Microsoft, the design called for another 3D engine. That’s where WPF came into the picture. WPF actually contains some aspects of Managed DirectX. WPF will be covered in more detail in the WPF section.

At this point there was also a design decision that needed to be made for the rules engine user interface (UI). WF comes with a rule set editor and provides an API to incorporate the dialog into other
applications; however, this dialog was geared more toward developers. Knowledge of objects and the ‘this’ operator were necessary to understand how to build rules using this UI. Because the object that was needed to write rules about is not known until run-time, it is difficult to use the WF rule set dialog. As a result the decision was made to write a rule set editor dialog that was easy to use by anyone including non-developers. This decision, however, did come with its own set of hurdles. It was now required that the custom rules UI do a conversion from the UI objects to CodeDOM which is what the RuleSet object in WF uses. This required gaining the knowledge of CodeDOM and learning the .NET libraries that allow the writing of CodeDOM.

Once the rules engine is invoked there is another menu bar on this dialog where the options for the opening or creating of a new rule set exist.

![Rules Editor File Menu](image)

Figure 3.2 – Rules Editor File Menu

WF has the concept of a rule set being a collection of rules. This same concept of rule sets and rules was carried forward to the rules engine. The rules engine has objects which relate closely to those of the RuleSet and Rule objects in WF. These objects contain properties that will map to the properties in the WF objects. But before covering the specifics of rule sets and rules, it is important first to understand how the rules engine knows about the database table columns that were chosen in the database engine. Upon invoking the rules engine, there are three arguments that are given to the rule engine. The first parameter is the WorkflowRuntimeManager that is used simply to invoke work flows that are used in the rules engine. The second parameter is a Hashtable which contains a list of the columns that will be used
to generate rules. The third parameter is the simple .NET type of Type. The ExecuteSqlWorkflow workflow has the second and third arguments as output parameters. In the database engine section it was discussed that the third parameter of type Type, is the type of the dynamically generated data type containing properties that represent the columns that were selected in the database engine. This parameter is not used in the rules engine UI at all, but is necessary to pass to some of the workflows involving the serialization and deserialization of the rule sets.

As stated earlier there are basically two options available to the user once the rules UI is showing. The options of either building a new rule set or opening an existing rule set. Of course the option to exit the dialog also exists but is not of any importance to the rules engine discussion. Upon creating a new rule set, the user is given the option of which template to base the rule set on.

Once the user selects a template, the template is opened and read to gather two of the three basic elements that the template provides. The first element that the rules engine cares about is the region. The second element that the rules engine cares about is the graphics that will be used to represent the data. Once the template has been read, it can populate the rules UI with the appropriate region and graphics options that will be used to build rules. Finally after the template is read, the new rule set logic invokes the new rule logic by default. There is no point in having a rule set without any rules so the first rule is created once the user enters a name to give the first rule in the dialog that is presented. Once the rule set and the first rule are created, the fields on the UI become enabled to allow for the user to begin defining the rule.
Rule definition is composed of three main elements. These three parts are the condition, the graphic element and the region. First the user must define the condition. There is a custom Condition Editor that is invoked from the condition button click that allows the user to define the condition.

The Condition Editor is given two arguments, both of type Hashtable. The first argument contains the list of column names that was passed into the Rules Editor. This list is used to provide a dropdown list of all the available columns. The second argument contains a list of values that are currently used for each of the columns in the table. To define a condition the user selects a field and adds it to the condition. The user then selects a relation or logical operator button from the buttons provided. Currently the available operators are equal, not equal, less than, less than or equal, greater than, greater than or equal, AND, and OR. Once the operator is added to the condition, the user then must either select a value from the drop down box containing possible values or enter a value in the provided text box and then add this to the condition. The user can see the rule in the preview window to see what it will look like and if it is in the appropriate format. Though the Condition editor is not totally clear on its usage, it does at least provide a bit of guidance on building a condition. This is an area of opportunity for improvement for the next version of the software. Once the condition is built the user selects ‘OK’ on the dialog, the condition is
applied to the rule, and the user is taken back to the rules editor. Once the condition is built the user now must decide what graphics object to use for the positive result of this rule. The property grid contains a property called GeometryDrawing where the user can see a list of graphics that are available. This list is one of the elements that came from the template. Once the user selects a graphic, they must then select a region. The region will decide the location of the graphic when the entire world view is generated.

These are the three basic elements that comprise a rule.

There are other options as well if it is necessary to expand the rule. One additional option is to include an else action. There is a check box immediately above the property grid that the user must select if an else action is desirable. When the user selects this box, another tab is displayed with another property grid with the same properties as the Then action property grid. The user must once again select a graphic (GeometryDrawing) and a region. Another option for a rule is to inactivate it. If for some reason a rule becomes unnecessary to execute but necessary to preserve for future purposes, the user can inactivate the rule by clicking the inactivate button. When the user inactivates a rule, an asterisk will appear for that rule stating that it is inactivate as an additional notification to the user. The other options that are available for a rule are in regards to the rule execution control.

Rule execution can be controlled in a number of different ways. First at a rule set level, meaning all rules, there is an option on the tools menu to indicate the chaining behavior of the rule set. The two options are Full and None.

Though WF supports three chaining behaviors, it was thought that for the purposes of this project only the two options would be necessary. The three options of chaining that WF support are Full, UpdateOnly
and None. The UpdateOnly chaining behavior requires explicit identification of dependencies between rules. This feature did not appear to add any value to this solution, therefore was not developed. In WF the Full chaining behavior is the default therefore it is also the default in this rules engine. Because Full chaining behavior changes the actual data and it is not desirable to modify the dataset data values directly in the table, the data when using chaining is stored in a collection object in memory. In the Properties section there is a property called “Assignment”. This is where the assignment statement for use with chaining would be defined.

![Assignment Property for Full Chaining](image)

Though this option is available, the restriction is that one and only one field may be updated. This is an area of opportunity to add the ability to perform multiple assignments.

Secondly, rule execution can be controlled at the rule level. There are two ways of doing so in this rules engine. The first way is dependent on the setting of the chaining behavior. If chaining is ‘Full’, then the rule reevaluation behavior is available. If chaining is ‘None’ then the reevaluation feature has no meaning. Once again reevaluation on is the default setting in WF therefore the default setting here.

Reevaluation can be suppressed by clicking on the Reevaluation button on the toolbar. If the button is highlighted, then reevaluation is on. If the button is not highlighted then reevaluation is off. Turning off reevaluation means that the rule will execute only a single time regardless of whether chaining is on or not. Another way to control rule execution is the priority field. The priority field is merely an integer value ranging from 0 to the number of rules in the rule set. The priority of a rule controls the sequence in which the rules in the rule set will be executed. The higher the priority the earlier in the sequence the
rule will be executed. The priority field is zero by default but can be changed to any value within the given range per rule. If the priority is the same for more than one rule then the default WF functionality of sequencing by alphabetical ordering of rule name comes into play.

Once all the rules have been defined for a rule set, the rule set can be saved to a file. When the user selects that save option on the menu, they are prompted to enter a rule set name. Once the rule set name is identified, a workflow (called SerializeRuleSetWorkflow) is invoked to perform the act of serializing the rule set to a file. The UI objects that are storing all of the rules and rule set data are passed as one larger object into this workflow. The template name is also needed to be stored with the rule set so that the graphics and regions can be attributed appropriately in the graphics engine. The database column data type also is needed so that the ‘then’ and ‘else’ actions can be built to set the appropriate fields in the dynamic object. This workflow consists of two custom activities. The first activity being the one in which WF RuleSet object is created from the UI objects. The second custom activity is the one that takes this RuleSet and serializes it to a file.

Once a rule set is saved to a file, it can then be executed. On the main window menu bar under the tools menu option there is the option to execute rules. Selecting this option will pop up a dialog where the user must select from a dropdown list the name of the rule set that was given when it was saved. Once the desired rule set is chosen, clicking the OK button starts the process to execute the rules. It is worth noting that the database connection must be open at this time so that the rows in the chosen table can be interrogated with the rule set execution. There are three main pieces to the rule set execution that take place at this time. First the RuleSet object must be deserialized from the file. The workflow DeserializeRuleSetWorkflow carries out this responsibility. This workflow contains only a single custom activity named DeserializeRuleSetActivity. Once the RuleSet object has been deserialized the RuleSet is executed with another workflow called RuleSetExecutionWorkflow. This workflow is a fairly complex workflow due to the usage of granular reusable custom activities pieced together to perform this action. The first activity is PopulateDatabaseColumnDataActivity where an object of that dynamic type
that was generated back in the database engine is created and populated from a single record in the
table. The second activity is in a code activity which creates an object of a specific type, the same specific
type of a property defined in the same dynamic data type that was populated in the previous activity.
This object will be populated by the WF rules engine upon execution of the rule set, which is the next step
in the workflow. This is performed by the ExecuteRuleSetActivity activity. This activity performs a rule
validation on the RuleSet first and then executes the RuleSet on the given row of data. The result is
stored in the object created in the previous activity. The last activity PopulateResultsController takes the
result object along with the row data and puts it in an object which is then stored to an array. These steps
are repeated for every row in the table until no more rows exist at which time the
RuleSetExecutionWorkflow is complete. The result array is returned back to the main program and
serialized to a rule execution results file and placed in a folder of the name RuleExecutionResults. This
XML based file is given a default file name based on the current date and time. The last piece of the rule
execution process is the starting of the graphics engine. The graphics engine being a separate application
needs to be invoked through a new process. The file name of the rule set execution results are given to
the graphics engine through a command line parameter. The graphics engine can then use this file to
generate the three dimensional world.

Future changes to the Rules Engine will include scripting in the label property for the flexible
display of values for the label when rendered in the graphics engine. Currently it is statically set to print
the region name followed by the number of elements in that region per total elements and the
percentage. Another enhancement would be to have the flexibility to position the text where desired. It
is currently fixed to be positioned to the top left of the region. A more complete set of commands for the
condition definition is desirable to give more flexibility in defining rules. For instance, it may be necessary
to have a rule to find all records that start with the letter “A”. Currently this is not possible.
3.6 Graphics Engine

The graphics engine is a WPF based application that utilizes the three dimensional features of the foundation. The design of the graphics engine is a very simple one. There is an App class associated with all WPF Application projects. This is the starting point of the application. Besides the app class there are currently only two windows in the graphics engine. The first window is the main window that shows the 3D world. The second window provides a list of the data rows that form each region.

The app class of the graphics engine has three basic responsibilities. The first one is to prevent multiple instances of the graphics engine to run. The second responsibility is to parse the command line arguments. The app class uses a static Hashtable to hold all of the command line arguments. Primarily there are only two command line arguments that are valid. They are the import command which requires a rule execution results file name and the open command which requires a visualization file name. These are the two main features that the graphics engine contains, import from a rule execution results and open saved graphics visualizations. The rules engine passes an imports statement in the command line to the graphics engine. The third responsibility of the app class is to start the main window.
The main window is where all the processing begins. The main window uses the Viewport3D WPF control to show the three dimensional world. To provide for user interaction with the 3D world the Viewport3D control is wrapped with what is called the TrackballDecorator. This is not a part of WPF but was written by the WPF team as a separate library called 3DTools. This tool supports the mouse click and drag interaction which enables the user to move the camera position of the 3D world. There was also a zoom feature added to the mouse wheel event to allow the zooming in and out of the camera in the 3D world. Figure 4-1 also shows a status bar where there exists buttons to control the position and look direction of the camera.

Once the main window is created and begins to load the command line arguments are interrogated to see if an import or open command has been given. If there is an import command (/import:RuleExecutionResults.xml) then the process of importing a rule execution results file begins. This process was threaded out due to the slow nature of loading 3D graphics from a XAML file. As part of the rule execution results file the template is stored along with the results from executing the
rules. The first step in the process of importing into the graphics engine is to read the template. The template is parsed and all the regions that have been defined in the template are stored in a Hashtable. All of the graphics elements are stored in a separate Hashtable. The remaining objects are moved into the 3D world and become the background or world object or objects. Once the template is loaded the next part is to iterate over all of the rule execution results. Each result has a region that it maps to and a graphics object that represents it along with the data from the row in the table. Since there may be thousands of results, depending on the size of the database table and the number of matching rules, only one graphics object of each region is expressed in the 3D world. All of the results are stored in separate Hashtables based on their identified region. So if there were three regions defined, then three Hashtables exist to hold all the results that fall under each region. Once the regions are defined the graphics objects are rendered to the 3D world in the location specified by their region. Once all the objects are rendered the 3D world is complete and ready for the user to interact with it.

The user can interact with the 3D world in a few different ways. First left clicking the mouse and dragging the mouse will allow the user to see the world in different positions. Using the mouse wheel the user can zoom in or out of the world. Also, right clicking the mouse over one of the graphics objects, excluding the background object(s), will pop up a message box asking if the user would like to view all the data that is associated with that object. If the user responds with ‘No’ then they are taken back to the 3D world. If the user responds with ‘Yes’ then they are taken to the second window in the graphics engine.
This second window displays all of the rows of data that are in that particular region. The user can view the data and perform sorts on any of the columns of data by simply clicking on the column name. An arrow next to the column name appears representing whether the data was sorting in ascending order or in descending order. Arrow pointing down indicates descending, pointing up indicates ascending. When the user has finished viewing the data they may close the window and open another one in the same manner.
CHAPTER IV
RESULTS AND DISCUSSION

Two differing sets of data were used to test out this solution. The first dataset that was used was a very large dataset that consisted of 250,000 records of library book data. The Library data set included 11 columns including but not limited to Title, author, subject, volume and location. The analysis of this data set consisted of a sample of 20 different rules all checking the Location column for a different value. Each rule evaluated the ‘LOC’ column for a different value and placed those records that evaluated to ‘true’ to a particular region. These are very basic rules but the point of this test was to determine how the rules engine and graphics engine would perform under a large dataset. The rules execution process of this rule set took 30 minutes and 17 seconds with Full chaining turned off. The execution process was also attempted with Full chaining turned on to see the difference in performance. This test showed that Full chaining was slower in that it took 37 minutes and 12 seconds to finish. It was realized that the column names that were used in this dataset contained characters that were not valid for XML. This was discovered when trying to build the model in the Graphics engine. The graphics engine was unable to perform this task. As a result, one of two possible changes to the system is required. The first possible change is in the database engine to restrict the characters that are allowed in the dataset column names. A second possible change is to change how Full chaining stores the updates in the data records by storing them to a temporary table. The second option is a more desirable change in that it does not require restrictions to the column names in the dataset.
One of the important aspects of the execution of the rule set that may impact performance is whether or not chaining is turned on. It was possible to make one performance enhancement with chaining turned off. The enhancement was to generate SQL statements in lieu of storing all of the data in memory. This removed the need to write all of the data to a file. It made it so that only the SQL statement and the primary keys are written to the file. The graphics engine is able to parse the results file and check for the existence of the SQL statement per rule. This greatly enhances performance for both the rule execution and the graphics visualization. The rendering of the images on the Graphics engine took 23 seconds. This is not a significant amount of time and an operation that could be waited on.

The second dataset that was used was a very small set of an individual’s private book library. There were 17 records for demonstrating the chaining functionality.
The dataset contained a list of book names, author, format, type (subject) and location. Think of a library that has a specific location for all the books based on type and format. All nonfiction books are located in one section and all biographies are in another section and so on for all the various types of books. One exception to this rule is that all audio books are kept separate in another location. Since the database was populated by type the location of the single audio cd (See Figure 5.1 A) in the list of books shows a location of 2 with all of the other biographies. To accommodate this, a rule is created with a condition of format = “Audio CD” (See Figure 5.1 B) with the assignment of Location = 5. There are five other rules within this ruleset that define where each of the 5 different locations will be located in the model. The result of executing this ruleset on this set of data is shown in figure 5.2.
This figure shows how there is one single book in Region5, where all audio cd formats were located. When looking at the objects in that region we see that there is only the single audio cd in the list.
CHAPTER V
CONCLUSIONS AND FUTURE WORK

In this thesis a framework was introduced which allowed for the creation of 3D visualizations to symbolically represent non-quantitative data. A detailed description of this framework was followed by a demonstration of the special visualization needs of individual datasets without the need for custom software. The framework contained four main parts: the database engine, rules engine, template editor and graphics engine. It was shown how application specific rules could be defined, executed against a dataset and represented using symbols in the graphics engine. The importance of the template editor was demonstrated in the building of a 3D model, the creation and positioning of regions and the customization of symbols. Only the rules engine and the template editor required knowledge of the application domain and the data. This allows for the framework to be unchanged as the application domain changes. The creation of a new model and the definition of new rules are the only changes necessary to accommodate a new application domain. To test this framework, a large dataset of books was used to model and visualize a library application in 3D.

Discussions have evaluated the various parts of a Rules-based Symbolic Database Visualization solution utilizing the .NET Framework. The technologies used to design and implement the application were discussed, as well as the frameworks that were used. This project describes how useful the Windows Workflow Framework (WF) is to use along with the feature of a Rules Engine. While the WF Rules Engine from a developer perspective is fairly easy to use in a static sense, significant effort went into using it in a dynamic fashion. Using reflection to create a new object for each execution is not the most straightforward of coding patterns, it is however much simpler than using the CodeDOM pattern. Though, CodeDOM is not very straightforward or easy to use, it does give the WF Rules Engine the ability
to perform updates to rules without a need to recompile any code. While this architecture shows that WF is very useful for mapping out workflows and the rules engine has its place in defining rules, it is obvious that the rules engine leaves much to be desired in the way of performance. For smaller data sets the performance is acceptable, for larger data sets, it is unbearable.

This project describes how feature rich WPF is. Though a little more effort may be necessary to do so, WPF allows for the creation of a very rich UI. The Graphics Engine shows how WPF can read in a generically formatted XML file and generate a three dimensional world.

The .NET solution for a Rules Based Database Visualization included four very important pieces. The Database Engine allows flexibility in connecting to numerous types of datasets and dynamically creates a type and object of that type to be used for the Rules Engine. The Rules Engine is able to read a data set and create and execute rules on individual rows of that data set to yield a generic XML formatted file. The Template Editor enables the creation of a template that can be persisted to a file in the XAML format. The Graphics engine is able to use this XAML file generated by the Template Editor to present a 3-dimensional visualization of the data provided by the Database engine.

Each one of these pieces independently may be insignificant but put the pieces together and truly the sum of the applications is greater than the parts of the whole. It is readily apparent how useful this collection of tools truly is when imagining symbolic data with no meaning can be transformed into an object in a three dimensional world. This application would be an invaluable asset to any team or organization that searches for information out of simple elements of data and its ease of use make it so that anyone, even non-technical users, can perform each step in the process.

Stepping through the sequence of events from the database to the graphics engine uncovered some potential for creating new features and fixing existing features. One feature that requires fixing is the way full chaining persists modified records. Instead of serializing the data out to an XML file, the updated records should be persisted in perhaps a temporary database table. Providing this fix would allow for flexibility in the naming of columns in the dataset.
A future enhancement for the database engine is to include the ability to select multiple tables for the dataset. Currently only one table may be selected. This restricts the dataset that can be used in this visualization framework and may require some up front work to put the data in a format that is useable by this system.

A future enhancement for the rules engine is to provide a scripting feature for an elements label in the rules definition interface. This would allow for the flexibility of the text that would appear in the visualization, potentially giving more information about each region. Another feature enhancement would be to have a more complete set of commands for the condition definition. It is desirable to give more flexibility in being able to find all records that start with the letter “A”.

A future enhancement for the graphics engine is to provide the ability to visualize the data within a browser. Currently it only displays in the provided Rules Engine window. Providing a multi-platform visualization opens up the visualization to be used by more users running different software and different operating systems. Another option that would be nice to have would be to allow for the support of performing animations to the objects on some event like on a click event.
REFERENCES


