**IEEE STANDARD 1016: InvestiFull Design Specification**

**The Software Design Specification Outline**

**1. Introduction**

**1.1 Purpose of this document**

This document outlines the design of the development of InvestiFull. InvestiFull is software that provides topological information on fullerenes that exist within a given range of up to two hundred carbon atoms. InvestiFull is a python program that interfaces with an SQLite database, and the Fortran program ‘Spiral.f’ to produce an adjacency matrix and chart for each desired fullerene.

InvestiFull uses the Model View View-Model design pattern. The program’s main purpose is to model and view Fullerenes. The three components are the Model, View, and View-Model. Therefore, this accurately represents the structure and purpose of Investifull. View contains the user-interface and functionality in order to interact with the view-model. While the view-model component accesses the model and representation of the Fullerene. Finally, the Model component contains the SQLite database and the data of each Fullerene entered into the InvestiFull program.

**1.2 Scope of the development project**

InvestiFull produces an adjacency matrix, and chart to represent the topology of fullerenes based on the range of carbon atoms input by the user. The goal of the project is to dynamically populate a database of ready to use adjacency matrices that aid graph theorists in their research.

**1.3 Definitions, acronyms, and abbreviations**

* Fullerene
  + Any series of hollow carbon molecules that form either a closed cage or a cylinder. [2]
* Isomer  
  Molecules that have the same numbers of the same kinds of atoms (and hence the same formula) but differ in chemical and physical properties. [4]
* Spiral Algorithm
  + This is in reference to a spiral algorithm that produces topological information on fullerenes. A Fortran implementation retrieved from GitHub user 'csgorham' utilizes in InvestiFull. [1]
* Schlegel Diagram
  + A Schlegel diagram is the projection of a polytope from n-dimensional space into n-1 dimensions. [3]
* Topological Chart
  + A chart displaying the geometric properties and spatial relations of a Fullerene.

**1.4 References**

[1] Csgorham, “csgorham/kinked\_chain,” GitHub, 05-Aug-2015. [Online]. Available: https://github.com/csgorham/kinked\_chain. [Accessed: 17-Feb-2020].

[2] D.R. Walton and H. W. Kroto, “Encyclopædia Britannica,” Encyclopædia Britannica, inc, 13 03 2019. [Online]. Available: https://www.britannica.com/science/fullerene. [Accessed 18 02 2020]

[3] E. M. Lazar, “Schlegel Diagrams,” Bar-llan University, [Online]. Available:

http://u.math.biu.ac.il/~mlazar/schlegels.html. [Accessed 20 2 2020].

[4] M. Jones, “Encyclopædia Britannica,” Encyclopædia Britannica, inc, 18 10 2016. [Online]. Available: https://www.britannica.com/science/isomerism. [Accessed 26 02 2020]

**1.5 Overview of document**

IEEE standards for document formatting and references to outside sources are followed throughout documentation for the InvestiFull project.

**2. System architecture description**

**2.1 Overview of modules / components**

This subsection will introduce the various components and subsystems.

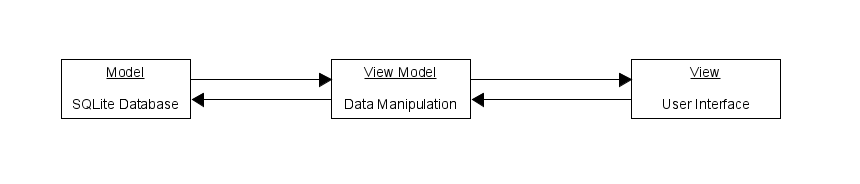
* Model
  + The model consists of a database holding topological information on fullerenes.
* View
  + The view is the user interface containing the options:
    - New
    - Append
    - Delete
    - Search
* View Model
  + This component handles translating the data in the Model to a format proper for the user to see in the View.

**2.2 Structure and relationships**

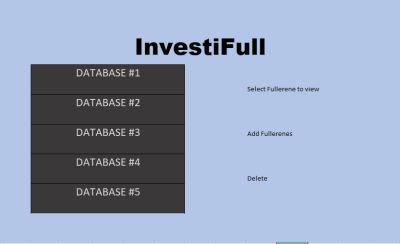
The Model handles any database queries based off the arguments provided to it from the View Model and passes back the result of these queries to the View Model to manipulate.

The View Model takes care of any data manipulation needed for doing tasks such as generating the SQL statements. These statements utilize the output of the spiral.f algorithm which returns the fullerenes created from the number of carbon atoms the user specifies. The View Model gets the user input from the View.

The View provides a user interface to the user, allowing them to enter the values needed to perform the operations available in the system. After the View Model takes care of its data manipulation, it will pass this data along to the View to display.



**2.3 User interface**



Screen #1

Includes drop-down menu of selectable databases to be used.

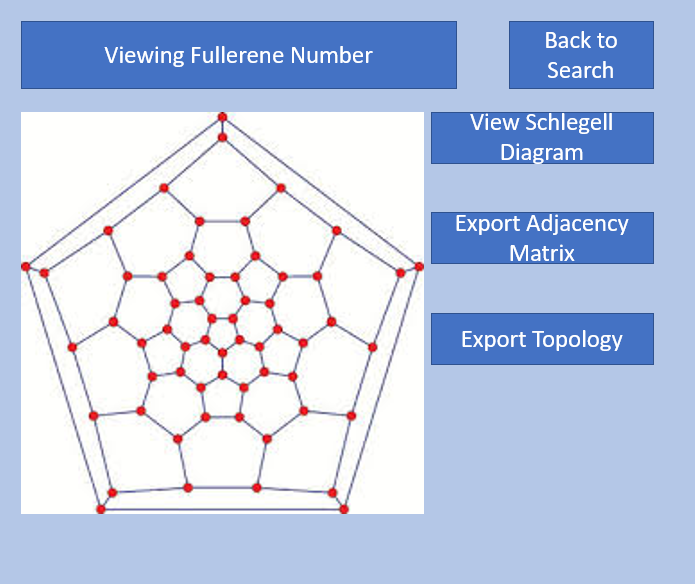
All options are only selectable after a database has been chosen by the user.

|  |  |
| --- | --- |
| Process | Description |
| Select fullerene to view | Sends user to “Screen #2” |
| Add fullerenes | Generates prompt to ask the user for the number of carbon atoms they would like to generate fullerenes for and if they would like to include touching pentagons. |
| Delete | Prompts user “Are you sure?” If they choose yes, then a database is deleted. |



Screen #2

|  |  |
| --- | --- |
| Process | Description |
| Enter Range | User Provides Range to search the database for Fullerenes |
| Displayed Fullerene | Creates Popup window that displays screen 3 for the selected fullerene based on user provided range. |



**Screen #3**

|  |  |
| --- | --- |
| Process | Description |
| Viewing Fullerene number | Display the number of the fullerene currently being examined. |
| View Schlegell Diagram | Display the Schlegell diagram of the fullerene currently being examined. This diagram will be displayed in a separate window. |
| Export Adjacency Matrix | Prompt user for the name of the text file they would like to create. Exports the adjacency matrix to a text file. |
| Export Topology | Prompt user for the name of the text file they would like to create. Exports the topology of the fullerene to a text file. |
| Back to Search | Return to screen #1 |

**3.0 Detailed description of components**

**3.1 Component template description**

* **Model**: Contains and manages the SQLite Database which contains the fullerene information and diagrams relating to it.
* **View**: The view contains the user interface of the software. It provides the user a visual representation of the view-model, allowing for the display of the data represented by the model, as well as provides
* **View-Model**: The View-Model controls and manages access to the Model component. It allows the user to search the Model data.

**3.2 Model**

|  |  |
| --- | --- |
| Identification | Model |
| Type | Class |
| Purpose | Store topological information on fullerenes |
| Function | Store the topological information provided from the ViewModel’s calculations. Protects access to this data. |
| Dependencies | View Model |
| Interfaces | Functions outlined in the pseudocode are utilized for communication to other modules |
| Resources | SQLite utilized for databasing. |
| Data | SQLite Database tables |

**3.3 View**

|  |  |
| --- | --- |
| Identification | View |
| Type | Class |
| Purpose | Provide user interface and observe the view-model |
| Function | Visual interface that connects to the View Model. Allows the user to input a range of carbon atoms which is then passed to the View Model component for processing. |
| Dependencies | View-model |
| Interfaces | Functions outlined in the pseudocode are utilized for communication to other modules |
| Resources | Io channels, system services |
| Data | Fullerene range provided by user |

**3.4 View Model**

|  |  |
| --- | --- |
| Identification | View Model |
| Type | Class |
| Purpose | Manipulate the data provided from the Model or View and present the manipulated data to correct component. |
| Function | Take actions requested by the view to perform on the model, or present the data from the model to the view. |
| Dependencies | View & Model |
| Interfaces | Functions outlined in the pseudocode are utilized for communication to other modules |
| Resources | Sprial.f |
| Data | Fullerenes adjacency matrices generated from Spiral.f |

**4.0 Reuse and relationships to other products**

Reuse is heavily considered regarding the implementation of InvestiFull. Due to most of the functionality coming from Spiral.f, most of the work involves creating a fully functional UI model that can be used to generate 2D images via inputting adjacency matrices. This UI should be designed with abstraction in mind and should have minimal dependence on the implementation of Spiral.f.

**5.0 Design decisions and tradeoffs**

Model View ViewModel was chosen over the more traditionally known Model View Controller because it fit our problem slightly better. Similar to the Model View Controller, the Model holds the data. The View is in charge of handling the user, and the ViewModel is in charge of translating the data from inside the Model to a format fit for the View. The difference is in the Controller vs the ViewModel. A controller is more heavy weight and makes modifications to the data, whereas the ViewModel doesn’t do any modification to the data and just presents it to the view so the user can see it. It acts more as a bridge between the Model and the View.

**6.0 Pseudocode for components**

**6.1 Data Base Module (Model)**

**6.1.1 Constructor(name)**

Establish sqlite connection to database name

Set currtable to Invalid Table

**6.1.2 Destructor( )**

Close SQLite connection

**6.1.3 GetTables( )**

Select names from SQLite master of type table

Return list of table names

**6.1.4 TableCount( )**

Select count of names from SQLite master of type table

**6.1.5 GetCurrTable( )**

Return currtable

**6.1.6 ChangeTable(name)**

If name in list of table names:

Set currtable to name

Else:

Print error message

**6.1.7 NewTable(name)**

If a table with of the same name doesn’t already exist:

Create table with given name in sqlite

Else:

Print error message

**6.1.8 DropTable(name)**

Drop table of given name in sqlite

**6.1.9 AddRecord(records, iso)**

Insert list of given records to the current table

**6.1.10 CheckTable(name)**

Attempt to select table of given name form sqlite master

If result of query is less than one:

Return false

Else:

Return true

**6.1.11 GetRecords( )**

Select all records from Database and insert them into an iterable list

Return records list

**6.2 View Model Module**

**6.2.1 Constructor( )**

Set currtable to Invalid Table

Create instance of database object called datb

**6.2.2 GetTableNames( )**

**tablenames = datb.GetTables( )**

**Return tablenames**

**6.2.3 TableCount( )**

num = datb.TableCount( )

Return num

**6.2.4 ChangeTable(name)**

namelist = GetTableNames( )

If name in namelist:

Set currtable to name

Datb.ChangeTable(name)

Return True

Else

Return False

**6.2.5 AddTable(name)**

If table of this name doesn’t exist:

Datb.NewTable(name)

Datb.ChangeTable(name)

Set currtable to name

Return true

Else:

Return false

**6.2.6 DropTable(name)**

namelist = GetTableNames( )

If name in namelist:

If name is currtable:

currtable = InvalidTable

datb.DropTable(name)

Return true

Return false

**6.2.7 GetRecords(name)**

Return datb.GetRecords

**6.2.8 AddIso(iso, pentaconnect)**

Inputstring = iso + pentaconnect //string concatenation

Fork spiral.exe

Feed inputstring to spiral.exe

Read output from spiral.exe

For line in output:

Append to records list

Datb.AddRecord(records, iso)

Return datb.GetRecords( )

**6.2.9 AddIsos(lowerBound, upperBound, pentaconnect)**

For iso in range(lowerBound, upperBound):

AddIso(iso, pentaconnect)

Return datb.GetRecords( )

**6.2.10 GetCurrTable( )**

Return currtable

**6.3 View Module (Command Line)**

**6.3.1 GetInput(VM)**

Print options to user

Read in user input

return user input

**6.3.2 HandleInput (userInput, VM)**

switch(userInput)

Case A: call add table method

Case Q: Quit program

Case C: Call GetTables()

print out return from GetTables()

Case V: Get user input for number of carbon atoms

VM.AddIsos(user input)

Case T: VM.GetTableNames()

get user input for table name

Check that table exists

if table exists switch to table

Case P: VM.GetRecords()

Print records returned from VM

Case D: VM.GetTableNames()

print names returned

get user input for name

if a table exists with that name, VM.DropTable(name)

**6.3.3 Main**

VM = ViewModel()

userInput = GetInput()

While userInput not quit

HandleInput(userInput, VM)

userInput = GetInput()

**7.0 Appendices**

**Appendix A: Glossary**

* **Fullerene**

Any series of hollow carbon molecules that form either a closed cage or a cylinder. [2]

* **Isomer**  
    
  Molecules that have the same numbers of the same kinds of atoms (and hence the same formula) but differ in chemical and physical properties. [4]
* **Spiral Algorithm**

This is in reference to a spiral algorithm that produces topological information on fullerenes. A Fortran implementation retrieved from GitHub user 'csgorham' utilizes in InvestiFull. [1]

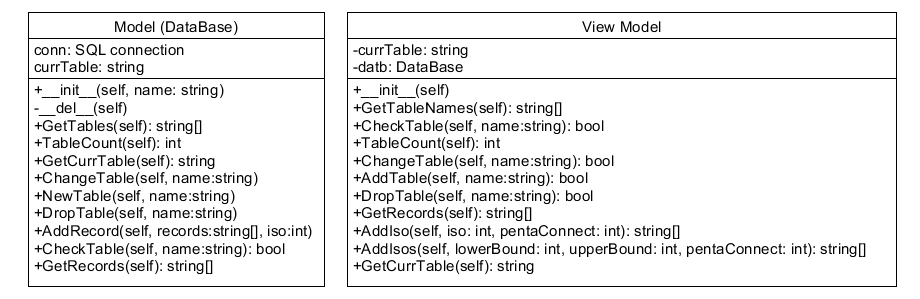
* **Schlegel Diagram**

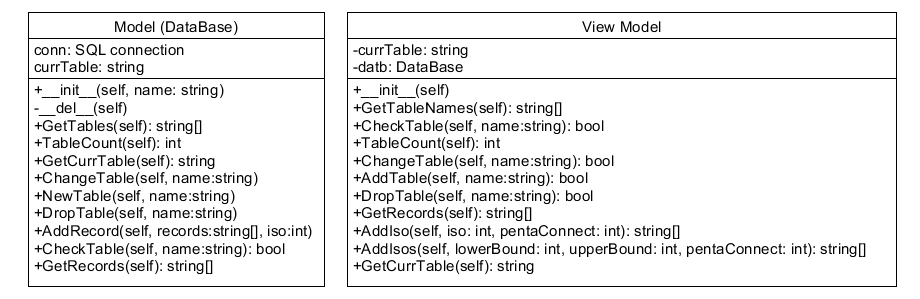
A Schlegel diagram is the projection of a polytope from n-dimensional space into n-1 dimensions. [3]

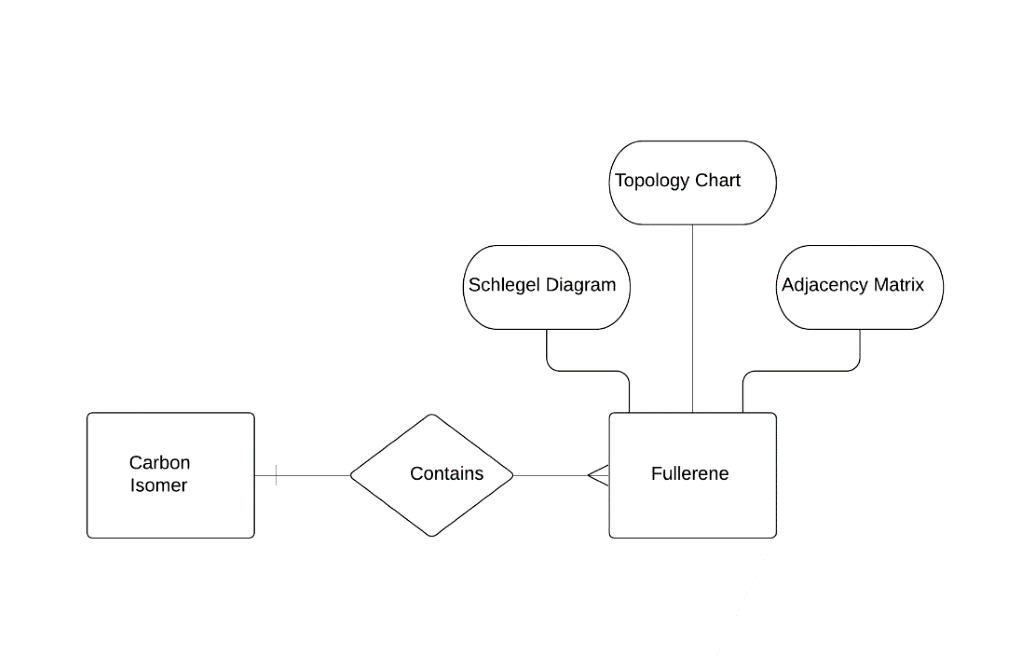
* **Topological Chart**

A chart displaying the geometric properties and spatial relations of a Fullerene.

**Appendix B: Analysis Models**







**The Testability of ‘Spiral.f’**

The spiral algorithm implemented by the Fortran program, Spiral.f, is only reliable in calculating the topology of fullerenes produced with up to two hundred carbon molecules. Consequently, this limitation is present in InvestiFull. Additionally, outputs from ‘spiral.f’ are assumed correct at all points in the operation of Investifull, as the documentation for ‘Spiral.f’ contains signification information regarding testing and efficiency of the algorithms implemented within it.