Molecular Trajectory Analysis

Presented by:

SUPER GROUP 3

**List of Contributors**

|  |  |  |
| --- | --- | --- |
| 1 | MIT2019002 | Ankita Hariyale |
| 2 | MIT2019003 | Fanishwar Satyam |
| 3 | MIT2019004 | Prashant Kushwaha |
| 4 | MIT2019005 | Abhinav Anand |
| 5 | MIT2019006 | Deepali Jindal |
| 6 | MIT2019007 | Hemant Kumar Lader |
| 7 | MIT2019009 | Kavita Singh Dhruv |
| 8 | MIT2019017 | Pragati Pawar |
| 9 | MIT2019020 | Biswajit Majumdar |
| 10 | MIT2019023 | Gaurav Rajput |
| 11 | MIT2019024 | Shivam Kasat |
| 12 | MIT2019042 | Komal Garg |
| 13 | MIT2019051 | Mayank Kabra |
| 14 | MIT2019052 | Shekhar Gupta |
| 15 | MIT2019054 | Shubham Kewat |
| 16 | MIT2019055 | Mani Shankar Javvaji |
| 17 | MIT2019056 | Yempalli Siva Kumar |
| 18 | MIT2019065 | Sonu Kumar |
| 19 | MIT2019066 | Koushal Kumar Sharma |
| 20 | MIT2019071 | Himanshu |
| 21 | MIT2019085 | Riya Khanna |
| 22 | MIT2019086 | Khushbu Jayeshkumar Mevada |
| 23 | MIT2019087 | Aadil Ali |
| 24 | MIT2019089 | Ajay Kumar |
| 25 | MIT2019097 | Praveen Kumar Raghav |
| 26 | MIT2019112 | Harish Kumar |
| 27 | MIT2019114 | Rohit Kumar |
| 28 | MIT2019117 | Sumit Kumar |
| 29 | MIT2019121 | Sudhanshu Ranjan |
| 30 | MIT2019123 | P.N. Janani |
| 31 | MBI2019011 | Utkarsh Singh |

**TABLE OF CONTENTS**

1. List of Contributors………………………………………………
2. Table of Contents……………………………………………

WHY THIS PROJECT? ……………..……………………………….

1. Software Specifications……………………………………………..
2. Software Requirements………...………………………….…...
3. Software API’s and Dependencies……………………………..
4. Software Process……………………………………………....…….
5. Designs………………………………………………………
   1. DFD Level 0………………………………………....…...
   2. DFD Level 1…………………………………....………...
   3. DFD Level 2……………………………………………...
   4. Class Diagram…………………………………….............
6. Usage Commands……………………………………………....
7. Software Validation…………………………………………….
8. Software Outcomes……………....…………………………….

BIBLIOGRAPHY……………………………………………...

**Why This Project?**

The software build in this project is to optimize the time complexity of other similar systems like cpptraj that works sequentially or uses parallelization for small number of functionalities. Such systems cannot take advantage of the growing computational power. The project aims at developing a system that can use distributed system architecture in order to provide faster results for the analysis.

**1.Software Specification:**

This Software System study the big dataset of Molecular Dynamic simulation and computes for Mask/Select, center, imaging, superposition, Average Structure, PDB generation, RMSD.

**2. Software Requirements:**

To speed up the process of simulation, we need parallel processing with distributed computing so Apache SPARK as big data processing engine and SCALA as language will be required. All the simulation will be compared with **cpptraj**. In this there will be dataset stored in files. Each file will contain frames of molecular dynamics Simulations.

* **Spark - Developed** at “UC Berkeley in 2009”, Spark is an open source big data processing engine, which has faster cluster computing platform. It works as nearby Real Time processing system as well as Batch Processing System. It was designed for fast computation. Spark is an optimized engine that provides high level expressive APIs in Java, Scala, Python and R which allows big data professionals for fast computation.
* **Scala -** Scala is a general-purpose, high-level, multi-paradigm programming language. Scala has been created by Martin Odersky and he released the first version in 2003. Scala smoothly integrates features of object-oriented and functional languages. It is a pure object-oriented programming language which also provides support to the functional programming approach. Scala programs can convert to byte codes and can run on the JVM (Java Virtual Machine). Scala stands for Scalable language. It also provides JavaScript runtimes. Scala is highly influenced by Java and some other programming languages like Lisp, Haskell, Pizza etc.
* **Cpptraj** – Cpptraj (the successor to ptraj) is the main program in Amber for processing coordinate trajectories and data files.Cpptraj has a wide range of functionality, and makes use of OpenMP/MPI to speed up many calculations, including processing ensembles of trajectories and/or conducting multiple analyses in parallel with MPI. Here, Trajectories with different topologies can be processed in the same run. Several actions/analyses in cpptraj are OpenMP parallelized.

**3. Software API’s and Dependencies:**

**Dependencies:**

* **netcdfAll-5.1.0.jar** - The netCDF library implements the full Common Data Model (CDM) model, including all other jar dependencies.
* **slf4j-jdk14-1.7.9.jar** - SLF4J stands for Simple Logging Facade for Java. It provides a simple abstraction of all the logging frameworks. It enables a user to work with any of the logging frameworks such as Log4j, Logback, JUL (java.util.logging), etc. using single dependency.

**API’s:**

1. MTA.PasingData.getTopologyArray(topologyFile pointer)

Input : Topology File

Output : Array containing the topology contents

Description : Takes the topology file and parses it into an array that has all the attributes required.

1. MTA.PasingData.getCoordinateArray(CoordinateFile pointer)

Input : Coordinate File

Output : Array containing the Coordinates contents

Description : Takes the coordinate file and parses it into an array that has all the X, Y and Z coordinates.

1. MTA.PasingData.generatePDB(topology Array, Coordinate Array, Topology File pointer)

Input : Array that contains the topology, Array that contains the Coordinates and topology File

Output : PDB File written onto Disk

Description : Merges the topology and coordinates array along with the Terminate statements to generate the PDB file onto the disk.

1. MTA.Masking.maskTopoCoord(Topology DF, Coordinate DF) : Masked DataFrame

Input : Coordinate Array and Topology Array

Output : DataFrame containing the contents after masking.

Description : Applies masking to the given coordinates and topology file.

1. MTA.Masking.maskPDB(PDB DF) : Masked DataFrame

Input : PDB File

Output : DataFrame containing the contents after masking.

Description : Applies masking to the PDB file.

1. MTA.findAvg()

INPUT: X,Y,ZCo-ordinates of crdfile.

OUTPUT: crd file of average X, Y,Z Co- odinates.

DESCRIPTION: performing average operation on X,Y,Z.

1. MTA.DistanceAngle.getFilteredCoordinate
2. MTA.DistanceAngle.calculateDistnceOfAtom

INPUT: String of multiple 3D (x, y, z) Coordinates of two Atoms

OUTPUT: Returns string of multiple distance between atoms in Å.

DESCRIPTION: Using 3D coordinates of atoms provided in input we will find distance between them.

1. MTA.DistanceAngle.calculateResiduewiseDistance
2. MTA.DistanceAngle.calculateAnglesOfAtoms

INPUT: String of multiple 3D (x, y, z) Coordinates of three Atoms

OUTPUT: Returns string of bond angle between the bonds formed by three atoms.

DESCRIPTION: It calculate the angle between bonds formed by three atoms with second atom as a vertex angle.

1. MTA.DistanceAngle.calculateAngleOfResidueWise
2. MTA.HBond.list(distance: Double, angle:Double,outFileName: String)

Write the number of hydrogen bonds and its information such as occurrence , maximum resistance time and list of occurrence for all frame to <outFileName>.

**Parameters :**

○ distance = cutoff bond distance for h-bond

○ angle = cutoff angle for h-bond

○ outFileName = filename for output

**Output:** File with name <outfileName> with hydrogen bond

1. MTA.Dihedral.getDihedralAngle

**INPUT**: X,Y,Z Co-ordinates of 4 atoms.

**OUTPUT**: Dihedral angle between these atoms.

**DESCRIPTION**: Using Vector of Vector of X,Y,Z co-ordinates four atoms, this will compute dihedral angle.

1. MTA.Dihedral.Backbone.phiAngle

**INPUT**: PDB file.

**OUTPUT**: DATAFRAME containing the phi angles.

**DESCRIPTION**: **PHI (**φ**)** angle is between “**C-N-Cα-C**“ backbone atoms. It will extract X,Y,Z co-ordinates of these atoms inside residues or chain and pass it to **getDihedralAngle** to calculate angle.

1. MTA.Dihedral.Backbone.psiAngle

**INPUT**: PDB file

**OUTPUT**: DATAFRAME containing psi angles.

**DESCRIPTION**: **PSI (ψ)** angle is between “**N-Cα-C-N**“ backbone atoms. It will extract X,Y,Z co-ordinates of these atoms inside residues or chain and pass it to **getDihedralAngle** to calculate angle.

1. MTA.Dihedral.Backbone.omegaAngle

**INPUT**: PDB file.

**OUTPUT**: DATAFRAME containing omega angles.

**DESCRIPTION**: **OMEGA (ω)** angle is between “**Cα-C-N-Cα**“ atoms. It will extract X,Y,Z co-ordinates of these atoms inside residues or chain and pass it to **getDihedralAngle** to calculate angle.

1. **MTA.Dihedral.specifiedSet**

**INPUT**: PDB file and 4 ATOMS as specified set.

**OUTPUT**: DATAFRAME containing dihedral angle between all set of these atoms. **DESCRIPTION**: It will extract X,Y,Z co-ordinates of these specified 4 atoms inside residues or chain and pass it to **getDihedralAngle** to calculate angle.

1. MTA.CentreImage.read.input( )

**Input:** Coordinate and topology files.

**Functionality:** The function reads the coordinate & topology files. **Output**: The coordinates of the first atom in the residue.

1. MTA.CentreImage.set.boundaries( )

**Input:** Coordinate and topology files.

**Functionality:** It calculates the coordinates of the box. **Output:** It returns the coordinates of the box.

1. MTA.CentreImage.box.centre( )

**Input:** Coordinates of the box. **Functionality:** It calculates the coordinates of centre of the box. **Output:** The function returns the coordinates of the centre.

1. MTA.CentreImage.calculate.distance( )

**Input:** Coordinates of first residue atom & centre of the box.

**Functionality:** It calculates the distance between the centre of the box and the coordinates of the first residue atom (this atom has been considered as the “anchor”). **Output:** It returns the distance between the anchor and the centre.

1. MTA.CentreImage.move.fixedatoms( )

**Input:** Coordinate and topology files.

**Functionality:** This function is responsible for the updation of the coordinates of the “fixed” (non-solvent) atoms. They are moved only if they come closer to the centre after updation.

**Output:** It just updates the coordinates and returns nothing.

1. MTA.CentreImage.move.mobileatoms( )

**Input:** Coordinate and topology files.

**Functionality:** This function is responsible for the updation of the coordinates of the “mobile” (solvent) atoms. They are moved freely. **Output:** It just updates the coordinates and returns nothing.

1. MTA.RMSD.isValidInputMatrix

Check if the given matrix is a valid matrix for the Kabsch algorithm.True if and only if the number of columns is at least two and the number of rows is at least the number of columns.

**Parameter**

M: taking the coordinates dataset of atoms.

Return:

**Boolean** If the dataset satisfy the property of kabsch property then it will return true.

Else false.

1. MTA.RMSD.simpleMatrix getCovariance

Calculate the covariance matrix

**Parameters**

It is a mathematical calculation using the given dataset, so there is no need to any parameter here. Because of dataset is already there in the previous module so just use them and find the covariance.

**Return:** matrix : It returns a covariance.

1. MTA.RMSD.calculate

Compute the Singular Value Decompositon of the Covariance Matrix. Use the results to create the rotaton and translaton matrix.

1. MTA.RMSD.getCentroids
2. MTA.RMSD.getTranslatonVector

Get the translation vector.

**Parameter:** void

**Return:** Matrix It returns the translation vector.

1. MTA.RMSD. getTranslatonMatrix

Get the translation matrix.

**parameter:** void

**Return:** Matrix: It returns the translation matrix.

1. MTA.RMSD. getRotation

Get the rotation matrix

**parameter:** void

**Return:** Matrix : It returns the rotation matrix.

1. MTA.RMSD.getRmsd

**Parameter**

**U** taking the coordinates dataset of atoms.

**V**: taking the coordinates dataset of atoms.

Return**:** FloatIt’s a final function of the module and it will return RMSD which will be a real number.

**4. Software Process:**

For this development we’ll be using **RAD** (Rapid Application Development) software development life cycle. Because of shorter time and less pre-planning, this is right process model to use. The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable. As our project is based on iterative development and it requires rapid changes and we have the sort of time to complete the project so RAD is used in our project. As our project is based on iterative development and it requires rapid changes and we have the sort of time to complete the project so RAD is used in our project. Following are the various phases of the RAD Model:

* **Application Generation:** The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.
* **Testing and Turnover:** The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

**5. Design:**

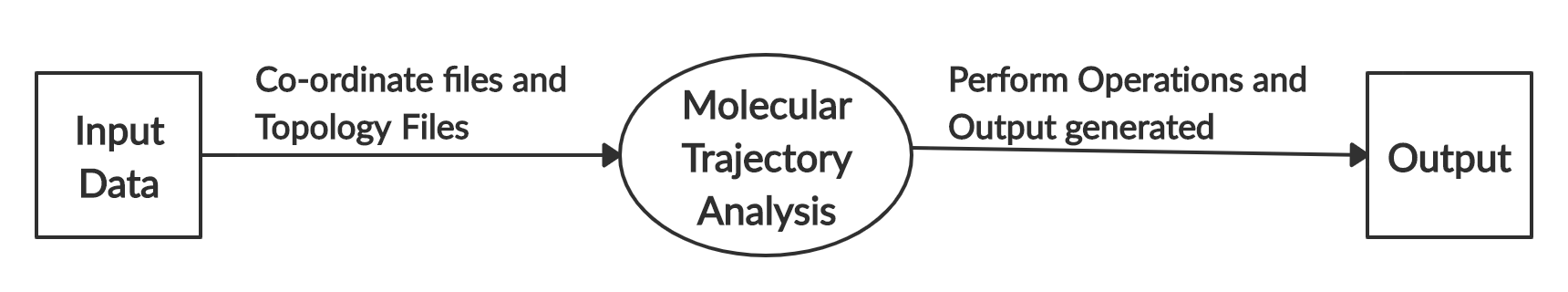
**5.1 DFD LEVEL 0 (Context level Diagram)**

There are 3 major modules to be included in this level. This is used for visualizing the data movement across the system. There are 2 external entities and a single ‘Process’.

The external entities here are:

1. User – Provides the Topology and Trajectory file to the Process.
2. Console – Prints the output.

Process – Read the Coordinates from trajectory file.

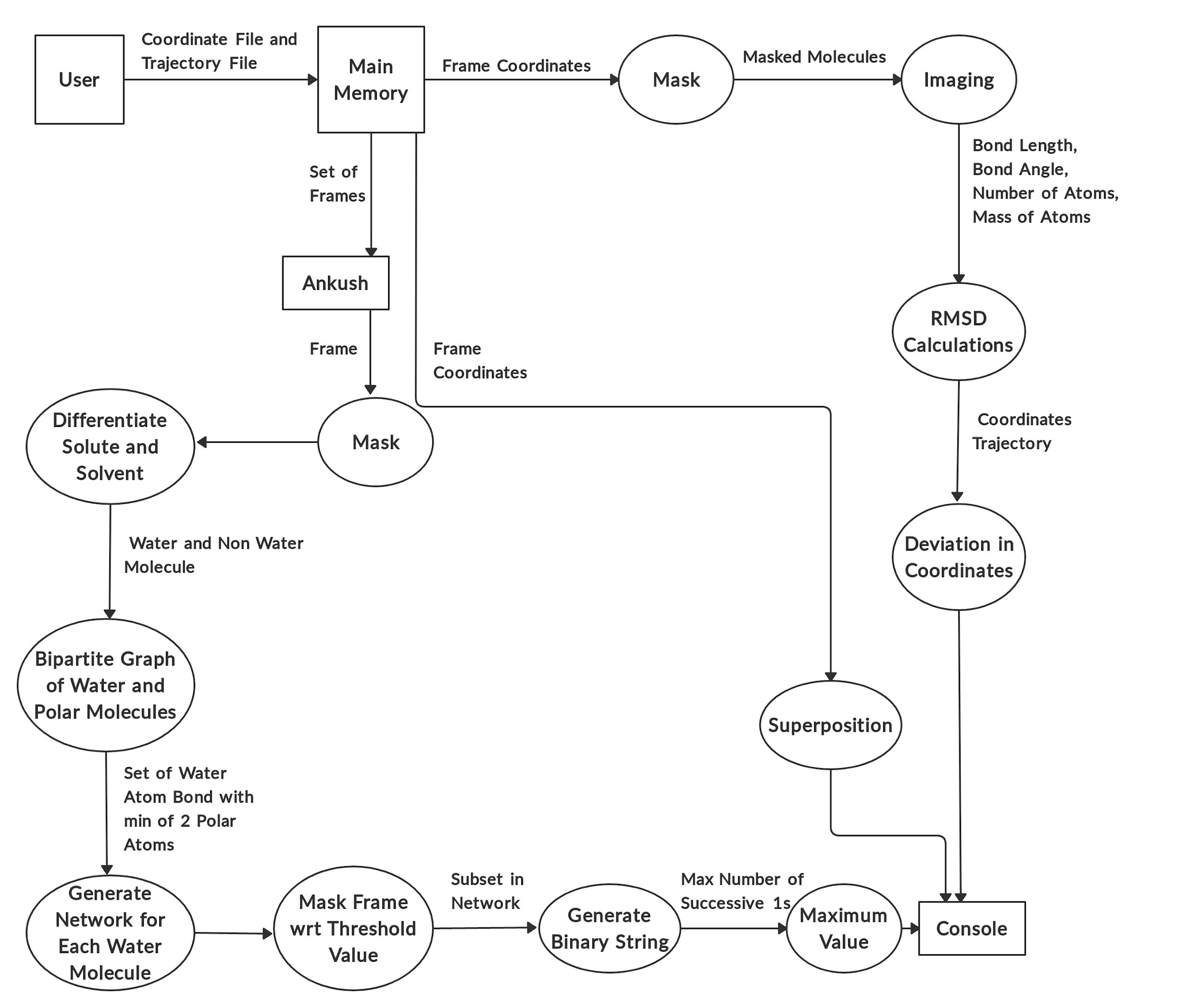
****

**5.2. DFD Level 1**

Here the ‘Process’ in level 0 is to be decomposed to get a better view of the system. In this level, there are few processes that come into consideration.

* **Masking** –Selection of atoms from residues for performing further tasks upon them.
* **Superposition** –Compares the structure and sequence of 2 related proteins / domains.
* **RMSD Calculation** –Root Mean Square Deviation analysis of the atomic positionused in MD Simulations to study the stability of our simulated system.
* **Imaging**-It’s a process of making a visual representation of somepicture ordocument.

After the RMSD is calculated, we analyse the deviation of our molecules from the reference set of molecules from the graph which will be produced as the output.



There is an extended part of the ‘Process’ here – **ANKUSH**.

We have to visualize and classify coordinates from .crd file into 2 parts - Solvent and Solute. Solvent will include all water molecules. Solute will include all non-water molecules. Generate network for each frame.

The network of each molecule will include more than or equal to 2 atoms. Now, for each network, we have to find how many such networks exists in other frames too. If the particular network crosses that threshold value then a Binary String is generated based on whether the network is present in the frame or not.From the binary string, we have to calculate the maximum length of 1’s and generate that number in the output – Maximum Residence Time (MRT). That max value is printed in the output.

**5.3. DFD Level 2**

This level describes the major modules in more detail. So this level will include :

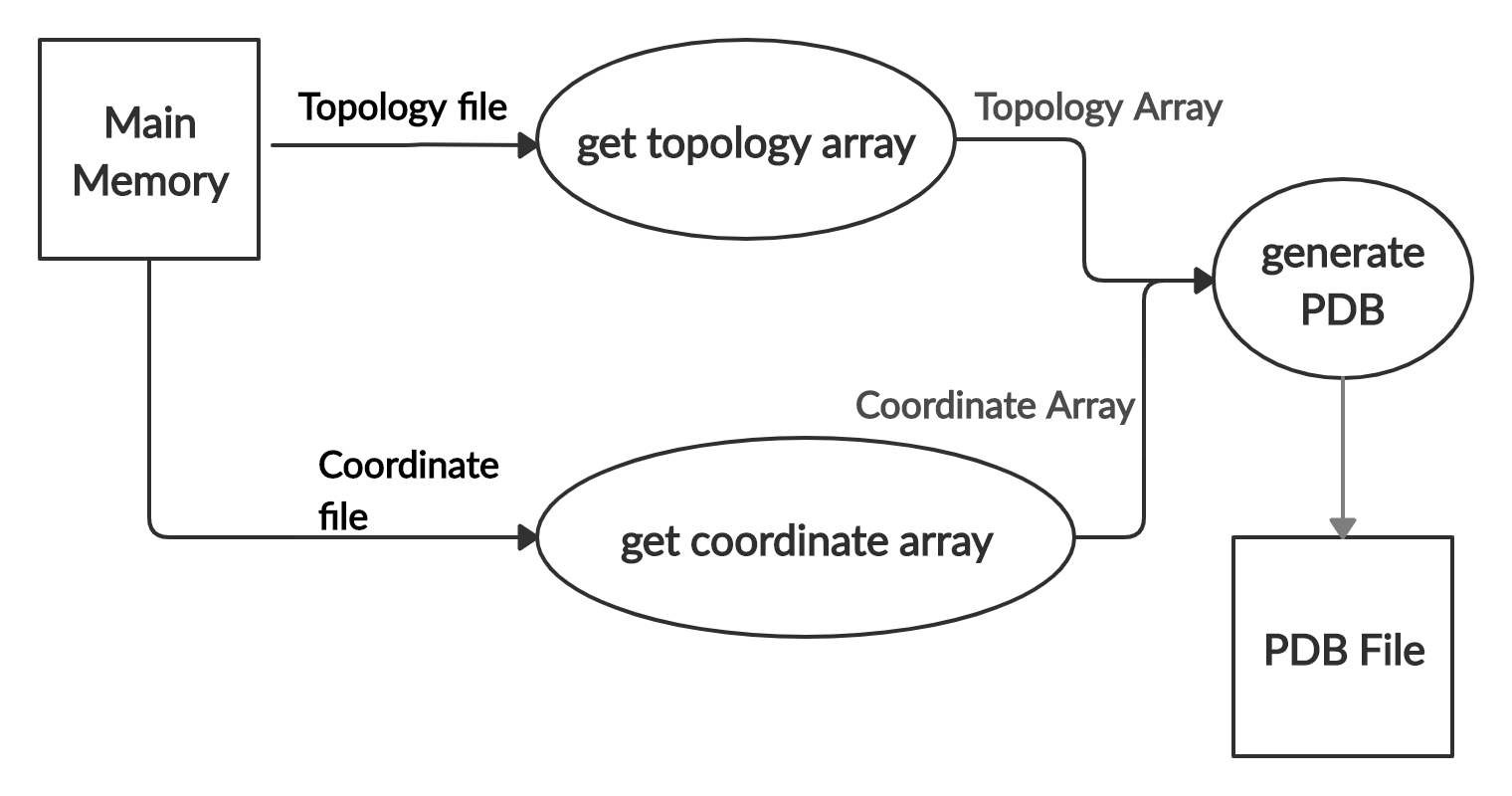
**a. PDB Generation** –

There are two input files in PDB generation process -

1. coordinate file(.crd file)

2. topology file(.top file)

From topology file, we stored usable data (atom number, atom name, atom type, residue number, residue name) into RAM and appended the coordinates from .crd file to that stored data and generated PDB file.



**b. Masking** -

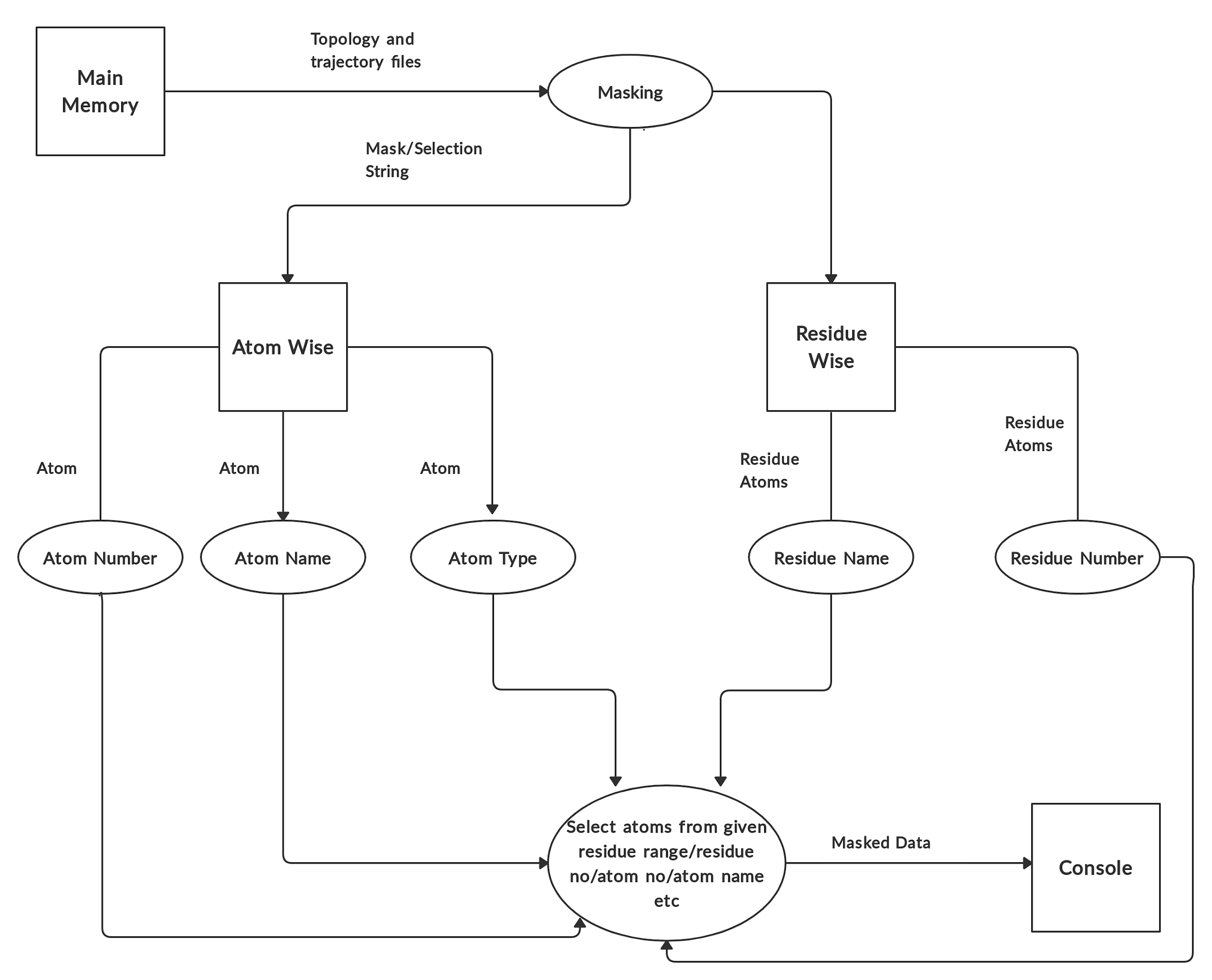
It’s a process of filtering of atoms that takes topology file and Trajectory file as input and generates the required atoms as the output , upon which the further calculations are performed on them.

Masking can be done in 2 ways –

1. By Atoms – Atom name, Atom Number, Atom Type
2. By Residue – Residue name, Residue Number

**“ @ ”** –This is used for selection of Atoms.

**“ : ”** –This is used for selection of Residues.



**c. Superposition** -

This is a process which is used to **compare structures** of 2 related proteins / domains. This can also be used to **compare sequence** of both the structures. The optimal orientation if found by minimizing the weighted sum of squared deviations of the rotated reference site positions from the observed site positions.

This includes –

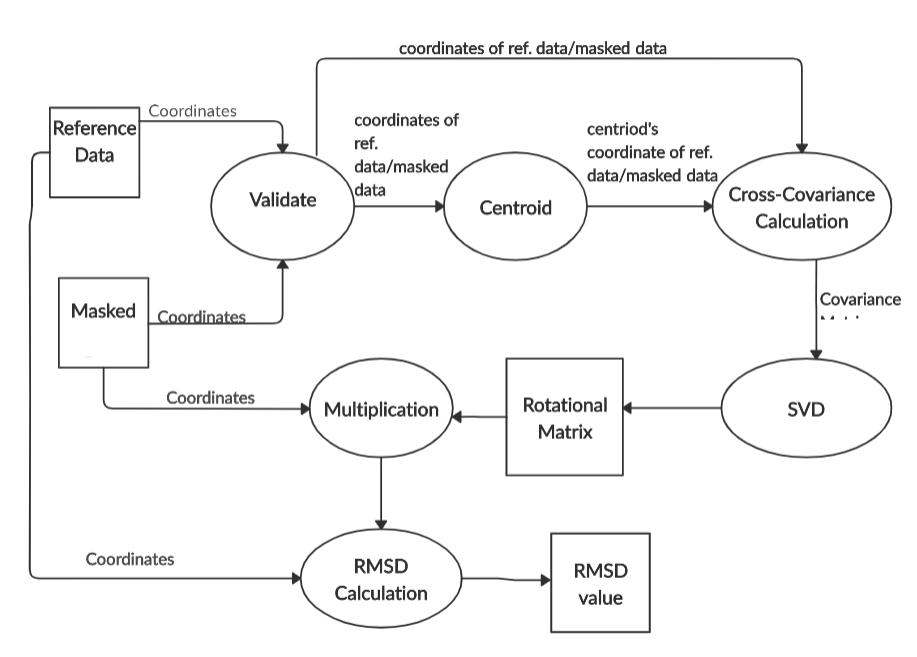
PDB file generated above, is provided as the input to find the superposition of our molecules from the reference molecules. Next, based on the coordinates in both the files, they are aligned and the superimposed visualization – Structure Comparison. Next Sequence for each chain can also be generated and can be compared – Sequence Comparison. From this we can also know about the ‘Percent Identity’.

**d. RMSD Calculation** -

It’s the **Root Mean Square Deviation Analysis** of the atomic position used in MD Simulations to study the stability of our simulated system. It’s the physical distance between structures.

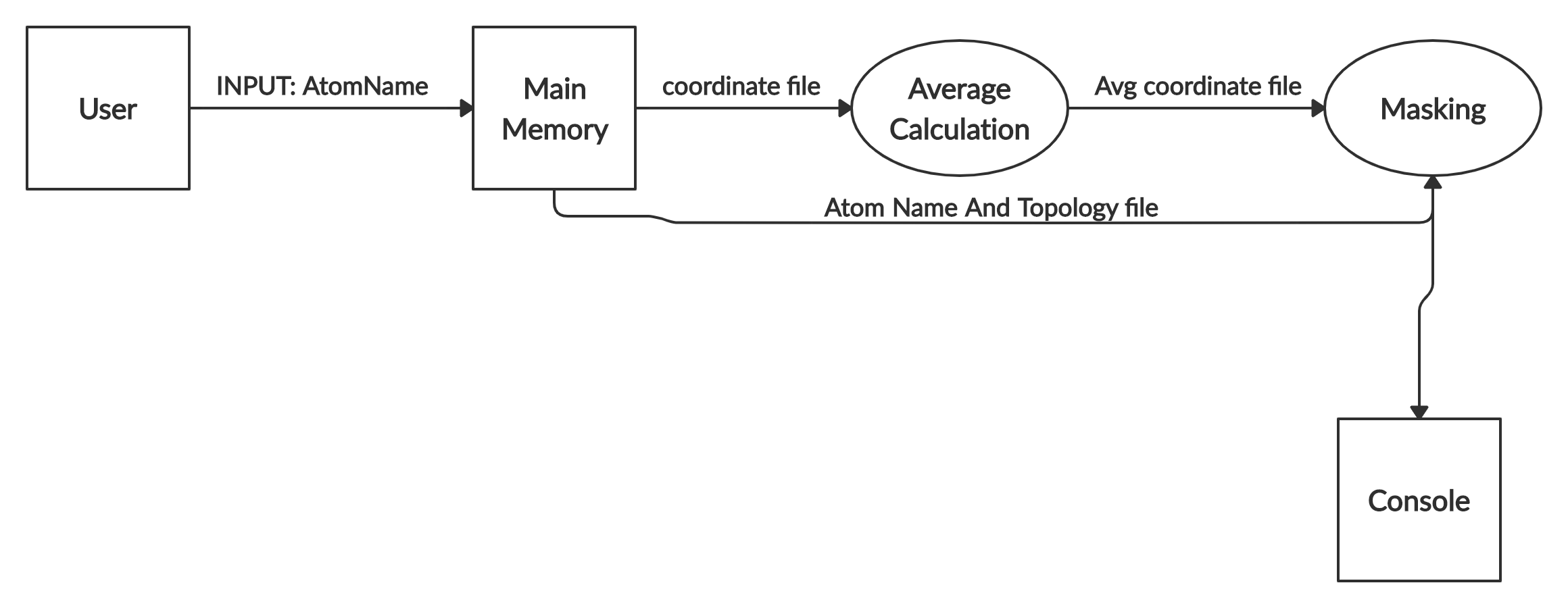
The steps involved in this –

PDB file generated above is provided as input to RMSD Calculation. After loading the file, it gives the structural information about the molecules like the number of atoms, number of frames in the provided file.We calculate RMSD for all the atoms given in the structure , or select any specific region of protein like ID- 100 to 150 , then the output will be the rmsd of this selected region only. Align our structure to the reference frame. We can show the output as a graph also. The graph depicts the relation between frame number vs rmsd values. So, the output will be an rmsd value which must be less than 2.5A.

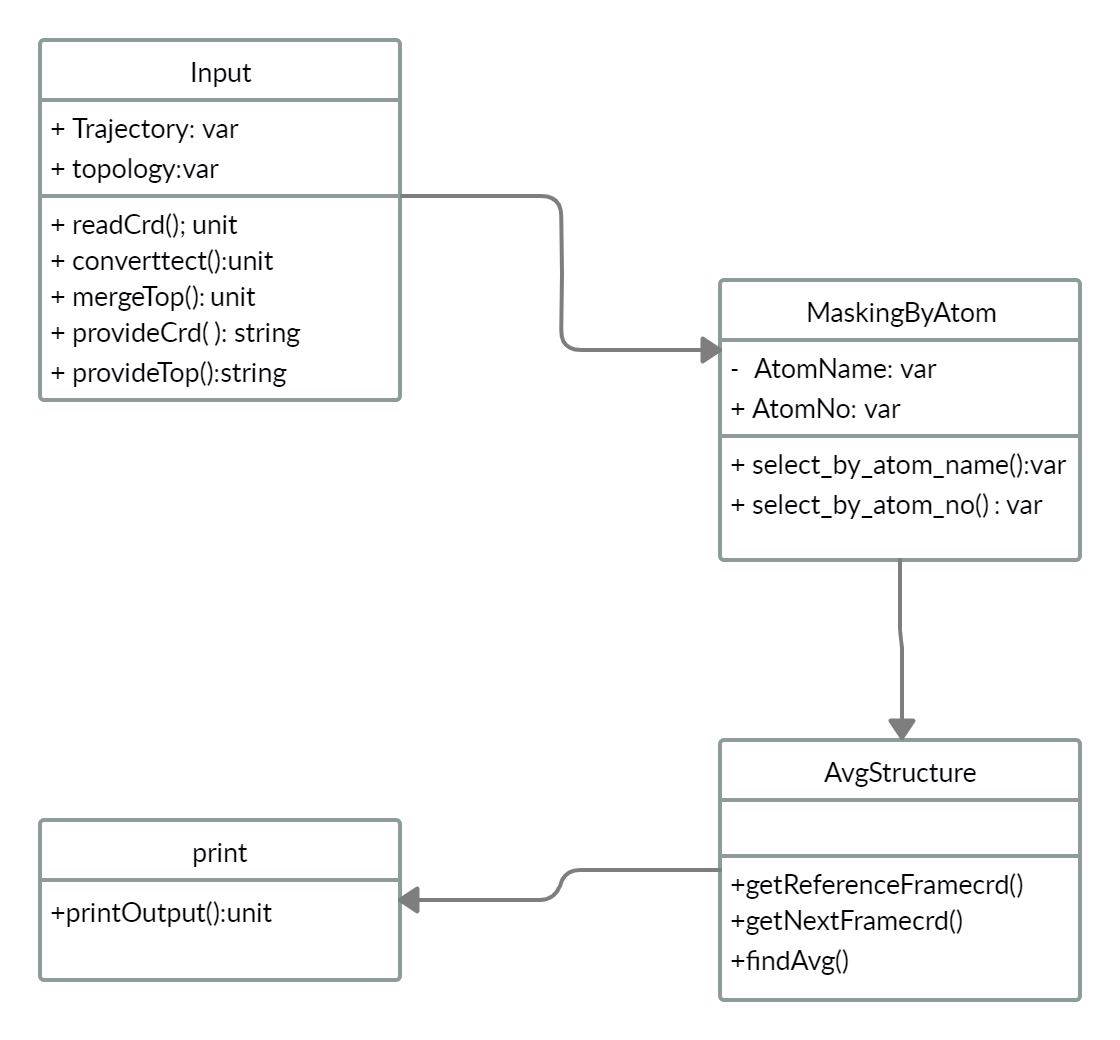
****

**e. Average Structure**

Average structure is simply the average of co-ordinates visited during the trajectory of Atoms. We calculate the average of co-ordinates by getting the co-ordinates from the crd file.

**DFD-**

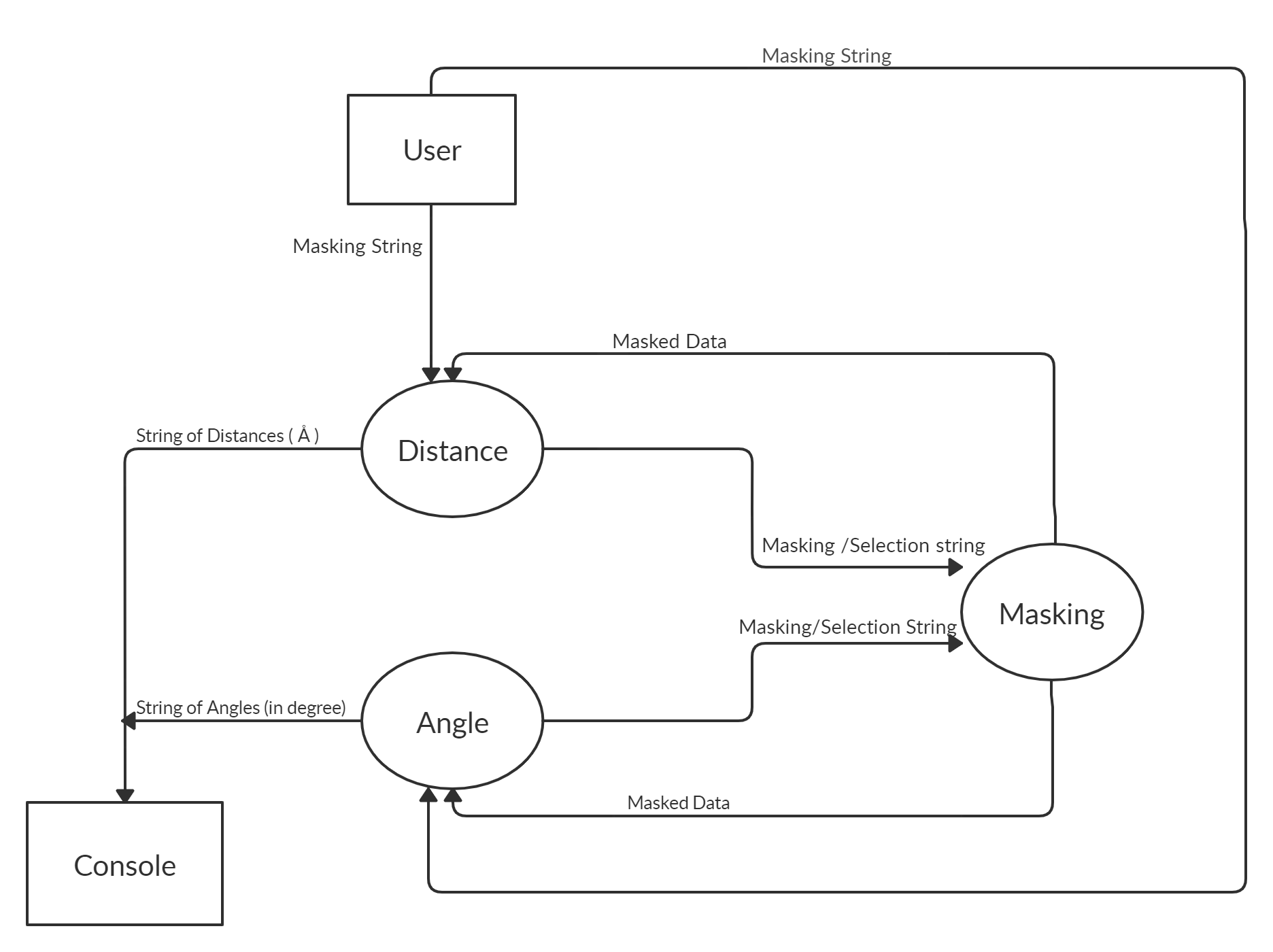
**Class Diagram-**

****

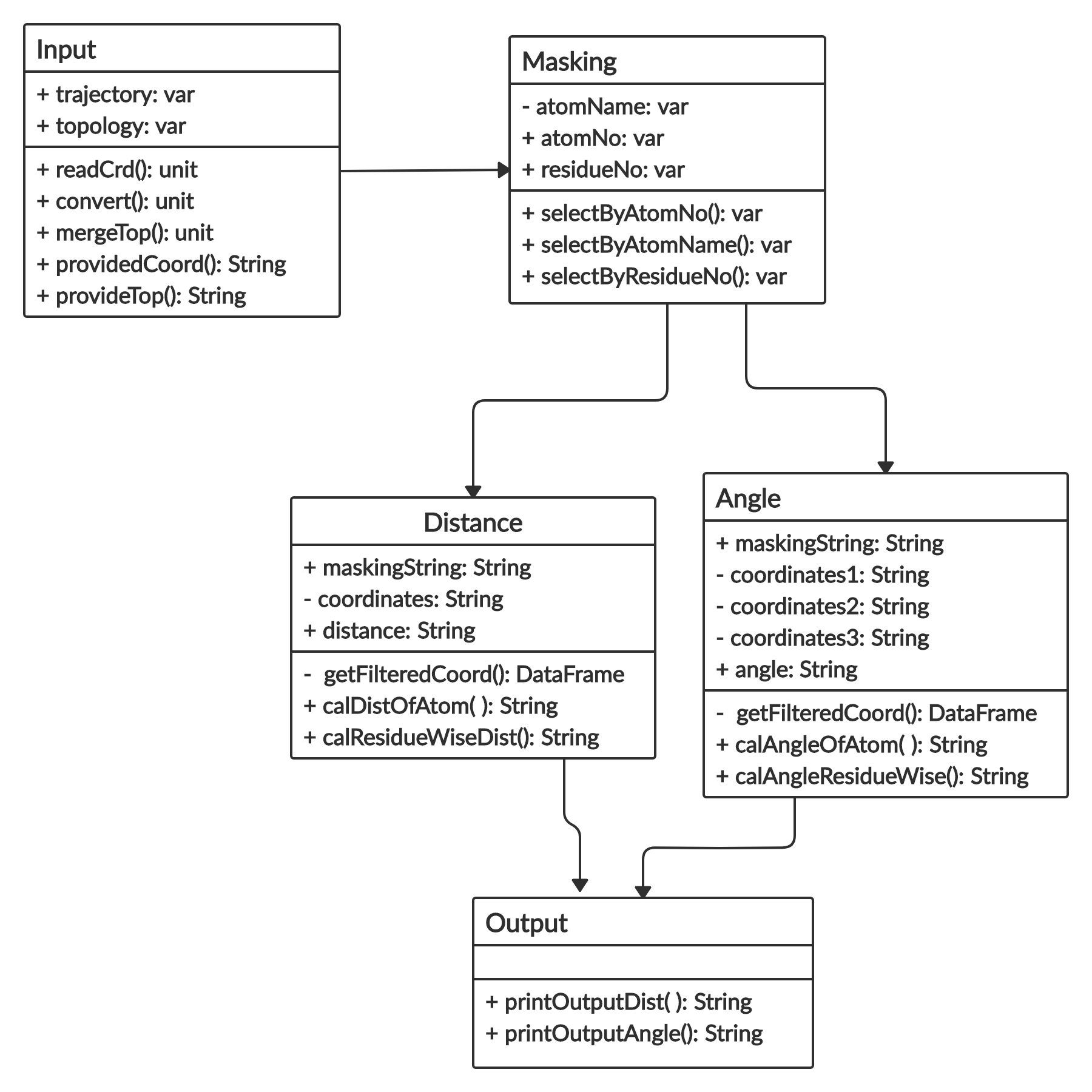
**f. Distance, Angle -**

In molecular geometry, bond length or bond distance is defined as the average distance between the nuclei of two bonded atoms in a molecule. Here we take coordinates of two atoms to calculate bond distance which further can be used by other modules. Bond angle is simply the angle between two bonds. And here we calculate the angle between the given bond which can be used by other modules.

**DFD-**

****

**Class Diagram-**

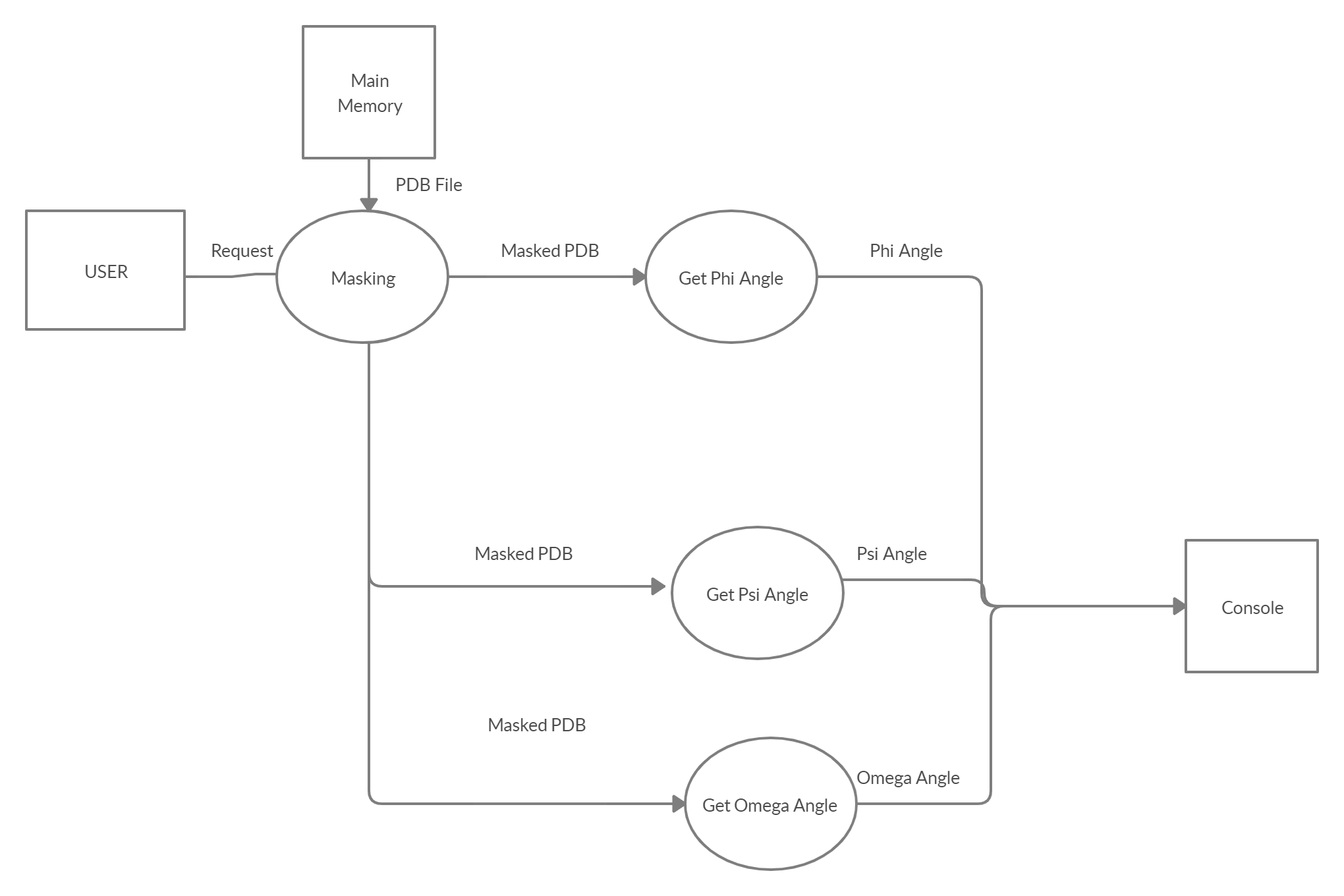
****

**g. Dihedrals -**

A dihedral angle is formed by three consecutive bonds in a molecule and defined by the angle created between the two outer bonds. The backbone of a protein has three different torsion angles.

These angles are called φ (phi) which involves the backbone atoms C-N-Cα-C, and ψ (psi) which involves the backbone atoms N-Cα-C-N, (ω) omega Cα-C-N-Cα.

**DFD-**

****

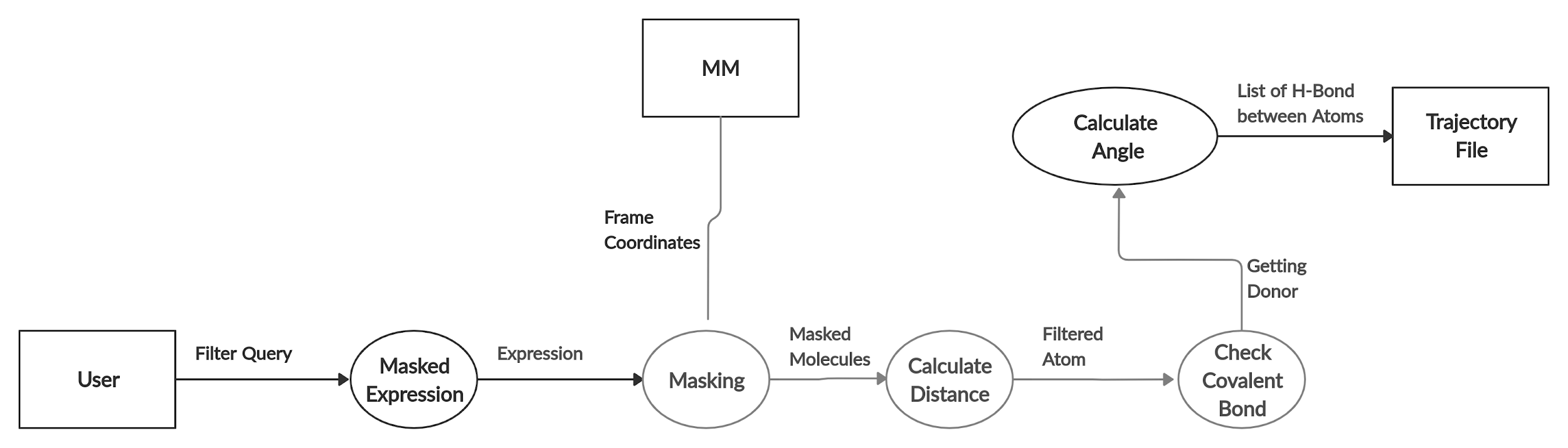
**h. Hydrogen bond-**

Determine hydrogen bonds in each frame using geometric criteria: angle and distance. A hydrogen bond is defined as being between an acceptor A, hydrogen atom H and donor atom D. If the A to D distance is less than the distance cut off 3.0 A - 3.5 A and the A-H-D angle is equal to angle cut off 120º a hydrogen bond is considered formed.

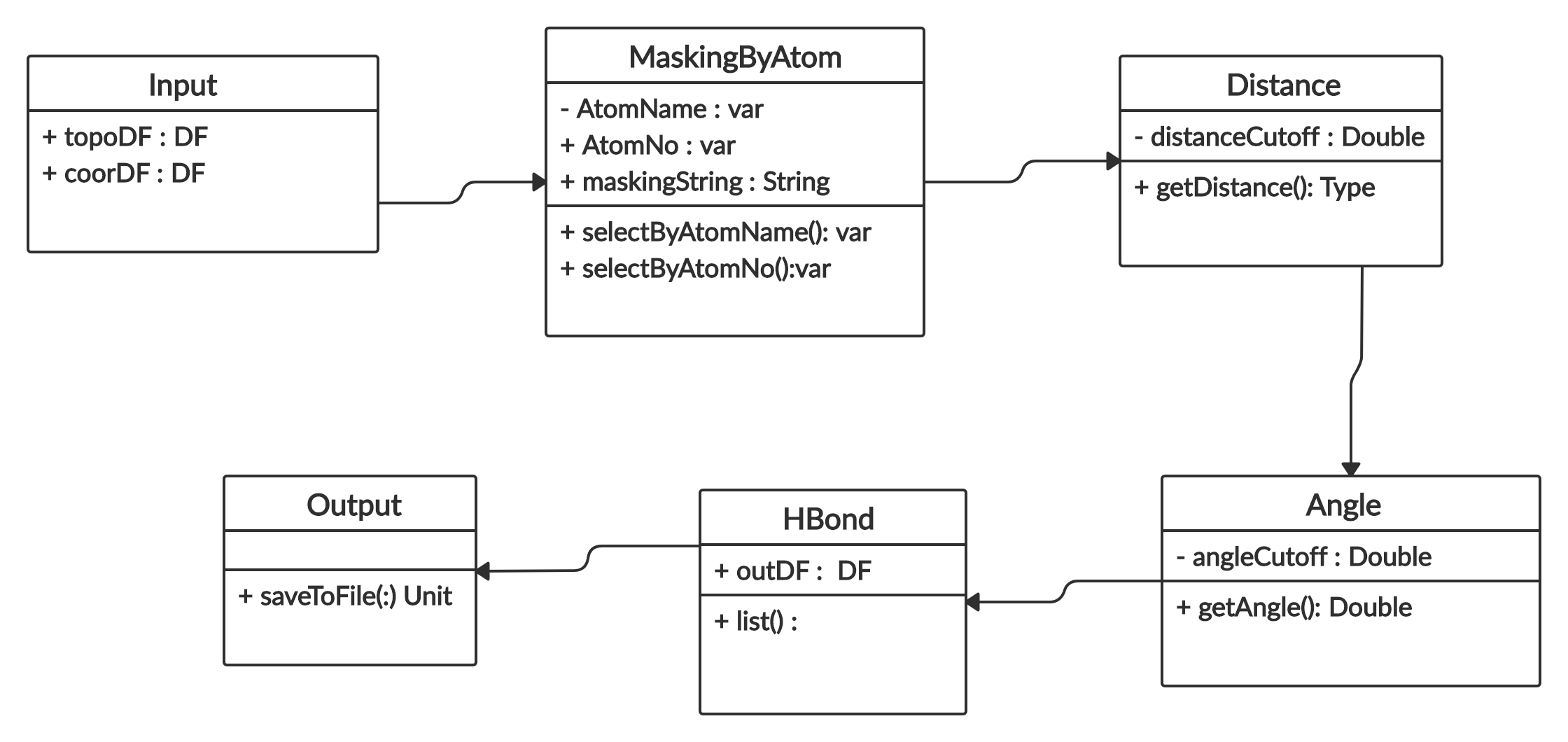
Determination of hydrogen bond donors/acceptors uses the simplistic criterion that hydrogen bonds are ON, i.e., hydrogens bonded to O, and N atoms are considered donors, and O and N atoms are considered acceptors.

The number of hydrogen bonds present at each frame will be determined and written to the file specified by user. The time series for each hydrogen bond (1 for present, 0 for not present) will also be saved for subsequent analysis. Solute-solute hydrogen bonds will be saved to file name specified by the user. The atoms are written with the residue name, residue number and atoms involved in hydrogen bonds.

**DFD-**

****

**Class Diagram-**

****

**i. Centre, Imaging-**

The first step is to find the centre of the box which can be done in different ways for example origin, reference etc. Then the second step is Imaging which is moving all the atoms’ coordinates in such a way that our anchor atom is at the centre of the box.

**7. Software Validation:**

Testing shows the presence of errors. With the help of testing we can see the difference between our outcome and desired outcome which can be seen with the help of cpptraj. With the help of validation, we can be sure if we are on the correct path.

We can do various types of testing like beta testing, component testing, system testing, acceptance testing.

**8. Software Outcomes:**

Various measurements of Molecular trajectory analysis like Select, RMSD, imaging, superimposition, Average Structure etc. will be computed with the help of Spark and Scala.

**BIBLIOGRAPHY**

1. Amber 8 Manual: <http://gohom.win/ManualHom/Amber/amber8.pdf>
2. CPPTRAJ: <https://amber-md.github.io/cpptraj/CPPTRAJ.xhtml>
3. Spark Documentation: <https://spark.apache.org/docs/latest/>
4. Scala Documentation: <https://docs.scala-lang.org>