8-2011

# An Extendable Software Architecture for Massively Parallel Biological Simulation

Karthik Iyer
*Utah State University*

# AN EXTENDABLE SOFTWARE ARCHITECTURE FOR MASSIVELY PARALLEL BIOLOGICAL SIMULATION

by

Karthik Iyer

A report submitted in partial fulfillment of
the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

| | |
|---|---|
| Dr. Nicholas Flann<br>Major Professor | Dr. Curtis Dyreson<br>Committee Member |

Dr. Daniel Bryce
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011

# ABSTRACT

AN EXTENDABLE SOFTWARE
ARCHITECTURE FOR MASSIVELY
PARALLEL BIOLOGICAL SIMULATION

by

Karthik Iyer

Utah State University, 2011

Major Professor: Dr. Nick Flann
Department: Computer Science

Information technology has aided tremendously in the growth of the medical sciences industry. Tissue simulation is one such area wherein information technology aids in the field of medical science. This project focuses on defining, designing, and developing a computational architecture that performs repetitive and rigorous tissue simulation runs under different conditions and utilizes parallel computation.

The goal of this work is to demonstrate a parallel network-based architecture involving multiple clients interacting with a server that sends and receives voluminous data. The project assumes large memory capacity for the clients and the servers, but limits bandwidth requirements. The work demonstrates a general purpose system designed to harness the power of parallel computation for use in the field of tissue simulation.

# ACKNOWLEDGMENTS

I thank Dr. Nick Flann for assisting me with this project and also in providing support during the course of my graduate program here at Utah State University.

I am grateful to my committee members, Dr. Curtis Dyreson and Dr. Dan Bryce, for their interest in this project and their valuable guidance.

I also thank Tanveer Zaman for his valuable input on Compucell which I used to test and validate various simulation models.

I also thank my wife, family, and friends whose support helped me in accomplishing this project.

Karthik Iyer

CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

As it becomes increasingly complex and diverse, medical science relies more and more on computing. The focus of this project is to define and implement a software architecture that helps biological simulations. Biological simulation depends on factors such as efficient CPU management, distributed task management, job scheduling and many more.

In order to harness all the needs of an efficient biological simulation model, this project defines decentralized, distributed software architecture. This architecture involves a single server referred to as a master and multiple clients referred to as slaves.

The master acts as a Web server[1] distributing tasks to all the clients and receives processed data from the clients in the form of images from each configuration run. The Web server is hosted using an IIS server version 5.1. The slaves receive the input configuration from the master in the form of XML data. The slaves generate images for a particular run. The actual simulation is performed by Compucell 3D[2], a 3D cell

[1] Web servers are used to deliver content that can be accessed over the Internet. Any web service must be hosted on a web server for it to be accessed by clients

[2] Compucell 3D is an open source software modeling environment used for cellular modeling.

simulation software residing in the client.

## 1.2 Project Details

Server client architecture in the form of Web services is the mode of communication for the simulation. The Server is configured as a Web server hosting the XML web service. The client consumes the XML web service by adding the web server reference.

The client server architecture mentioned above in the form of an XML webservice is used for distributed communication. The clients run Compucell3d simulation models that require python, numpy and Compucell3D software to be preinstalled on it. Each client runs a windows service that has the clients initiate as soon as Windows operating system boots up.

The approach mentioned above is similar to a multiprocessor system with subtle differences. The approach is similar in the way that multiple cores are used for computation. It differs in a way that the clients take xml as input and runs Compucell3D, a 3rd party simulation tool. The client joins the server dynamically and can run the simulation from any location as long as it has the web reference to the server added to the project.

The Web server sends the xml files and receives the files back from the client after each run. The client receives the xml file and sends them to Compucell3D for the simulation. The client sends the data obtained from the simulation back to the server. The server generates xml run files using a GUI tool. The data obtained from the simulation is compared with the configuration data in the xml to validate the images for a given xml configuration. The main project tasks involve creation of the GUI tool, web server and the client to consume the web service.

## 1.3 Compucell Overview

Compucell 3D is a three-dimensional problem solving and simulation environment for simulating disease and development processes in multi-cell systems. The input for Compucell 3D simulations is either an xml file or a Python file. The xml file identifies the type of simulation that needs to run. Compucell can be run using the GUI or the command line (by passing the name of xml file as the input).

A sample configuration input and the corresponding output using Compucell is described in brief in the following sub-section.

### 1.3.1 Compucell Input XML

A sample Compucell simulation (Ameobae 2D) is described in this section. Each simulation run requires an input configuration file in the form of an xml document. The xml contains all the parameters and configurations for a specific simulation to run. The following xml document describes various parameters for a sample run.

```
<?xml version="1.0" encoding="utf-8"?><Compucell 3D xmlns=""><Potts><Dimensions x="55"
y="55" z="1" /><Anneal>
0
</Anneal><Steps>
1000
</Steps><Temperature>
15
</Temperature><Flip2DimRatio>
1.0
</Flip2DimRatio><Boundary_y>
Periodic
</Boundary_y></Potts><Plugin Name="PlayerSettings"><Rotate3D XRot="27" YRot="-11"
/></Plugin><Plugin Name="CellType"><CellType TypeName="Medium" TypeId="0" /><CellType
TypeName="Amoeba" TypeId="1" /><CellType TypeName="Bacteria" TypeId="2"
/></Plugin><Plugin Name="Volume"><TargetVolume>
25
</TargetVolume><LambdaVolume>
```

```
15.0
</LambdaVolume></Plugin><Plugin Name="Surface"><TargetSurface>
25
</TargetSurface><LambdaSurface>
2.0
</LambdaSurface></Plugin><Plugin Name="Contact"><Energy Type1="Medium"
Type2="Medium">
0
</Energy><Energy Type1="Amoeba" Type2="Amoeba">
15
</Energy><Energy Type1="Amoeba" Type2="Medium">
8
</Energy><Energy Type1="Bacteria" Type2="Bacteria">
15
</Energy><Energy Type1="Bacteria" Type2="Amoeba">
15
</Energy><Energy Type1="Bacteria" Type2="Medium">
8
</Energy><NeighborOrder>
2
</NeighborOrder></Plugin><Plugin Name="Chemotaxis"><ChemicalField
Source="FlexibleDiffusionSolverFE" Name="FGF"><ChemotaxisByType Type="Amoeba"
Lambda="5" /><ChemotaxisByType Type="Bacteria" Lambda="5"
/></ChemicalField></Plugin><Steppable
Type="FlexibleDiffusionSolverFE"><DiffusionField><DiffusionData><FieldName>
FGF
</FieldName><DiffusionConstant>
0.000
</DiffusionConstant><DecayConstant>
0.0
</DecayConstant><ConcentrationFileName>
Demos/amoebae/amoebaConcentrationField_2D.txt
</ConcentrationFileName></DiffusionData></DiffusionField></Steppable><Steppable
Type="PIFInitializer"><PIFName>
Demos/amoebae/amoebae_2D.pif
</PIFName></Steppable></Compucell 3D>
```

The above configuration explains the lattice size for a simulation run (in this case x: 55, y: 55, z: 1). The steps indicate the number of runs in the simulation. In this case, the simulation ends in 1000 steps. The temperature induced in the system transforms the cell behavior. Other parameters like volume, area, and energy levels have a significant impact on the simulation behavior.

### 1.3.2 Compucell Output Images

The output sequence obtained from the simulation run for the above xml configuration is shown in Figure 1. This initial configuration shows two kinds of cells in different positions. The series of steps illustrated in Figure 2 shows the behavior of the cells during the course of the simulation run. The sample run sequence here shows that the green cell has more mobility and affinity in this situation. As can be seen in Figure 3, the final simulation confirms that the cells stick to each other, while at the same time trying to separate from each other. Therefore, in the example case, we can say that there is more affinity as compared to the degree of separation.

For more details on the structure of the input xml file for the Compucell simulation, refer to both Subsection 3.3 and Appendix A.

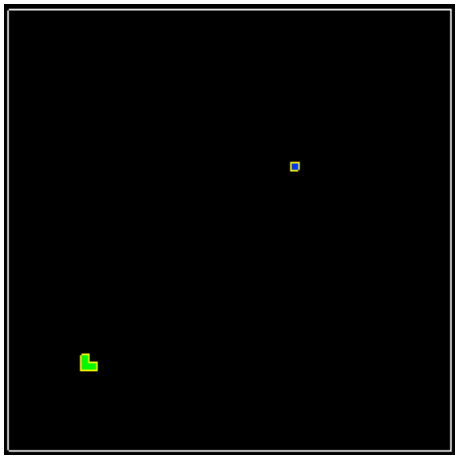


**Figure 1: Initial configuration.**

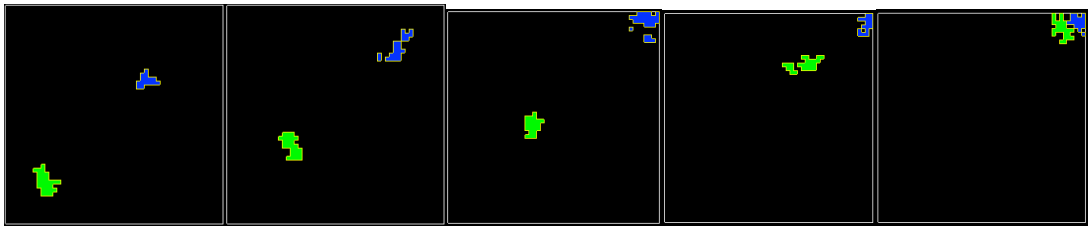


**Figure 2 Compucell simulation run.**

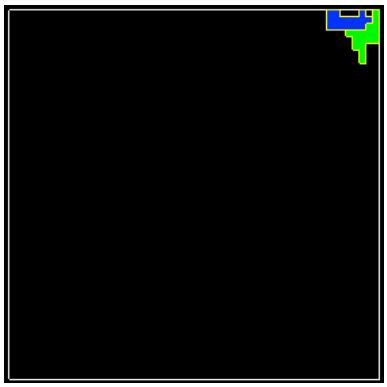


**Figure 3: Final configuration.**

## 1.4 Project Overview

In brief, the architecture designed and implemented for this project works in this way. The master sends XML run files to all the clients using web services and gets back images as the result of the simulation. The XML run files are generated on the server using an XML template and an experiment file consisting of a set of values A run file is created by substituting variables in the template file by the values in the experiment file. Run files are sent to a client for simulation run using Compucell. The images generated by Compucell are transferred back to the master for further analysis in the master.

The rest of this report is organized as follows. Chapter 2 provides a description of the requirements for this project. Chapter 3 discusses the overall design. Implementation is discussed in Chapter 4. Chapter 5 describes testing of the project. Finally, Chapter 6 provides directions for future work.

# CHAPTER 2

# REQUIREMENT ANALYSIS

## 2.1 UML Overview

In order to analyze the system, Unified Modeling Language (UML)[3] is used for identifying the actors and goals in the system. UML diagrams consist mainly of use case diagrams and class diagrams. Use case diagrams show a high level overview of the system, while class diagram shows more details of the objects, interfaces, and the associations among them.

The overall system here is visualized as a combination of two sub-systems. The server/master system resides on a central system (controller) and the slave/client resides as a part of the decentralized system. User goals defined in the following sections show the goals for the server and the client as separate entities.

The functional requirement section (2.3) below discusses constraints and system features regarding the overall system itself.

## 2.2 User Goals

The use cases define the interaction between the external actors and the system under consideration to achieve a goal. The actor could either be an external user or a given system

[3] Unified Modeling Language (UML) is a standard general purpose modeling language used in the field of object-oriented software engineering. It is used to specify, visualize, define, modify and construct artifacts for an object-oriented software system under development.

that needs to interact with another system in order to achieve a goal. In this case, we have an external user generate xml files depending on an input simulation xml and an experiment xml file.

There are three main actors for this system. The external actor or *user* generates the xml simulation files required for generating the images. The *server* which is the part of the system coordinates with the client. The *client* coordinates with the server and also performs the simulation through an external system, thus achieving the goal of the system.

The use case diagram in Figure 4 shows the main actors of the system.

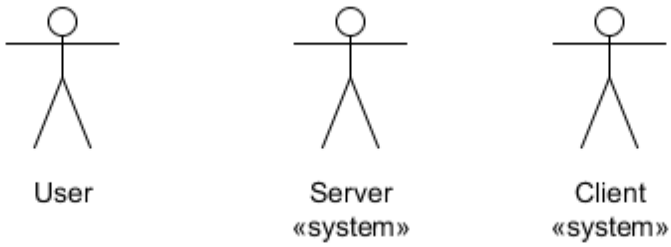



**Figure 4: Primary actors in the system.**

The use case diagram in Figure 5 below shows the high level goals of the user. The user is responsible for executing the GUI tool used to generate xml run files based on an input experiment xml file and a simulation xml file. The output generated using this tool serves as the input to the main simulation. The GUI tool receives input from the experiment file and the simulation file. It generates xml run files.

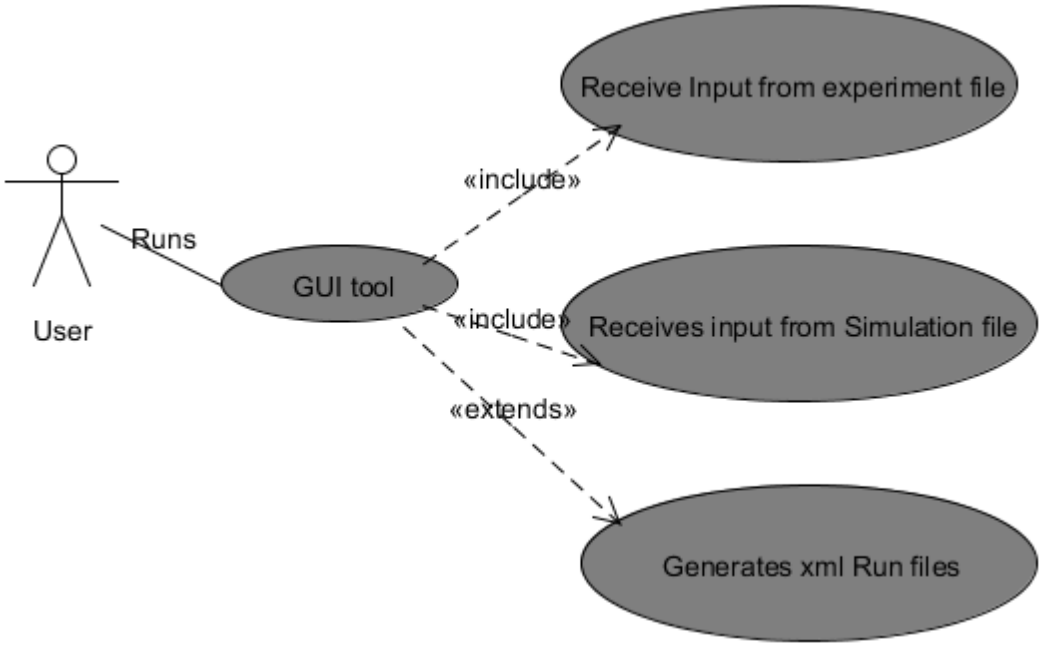



**Figure 5: Use case for user.**

The use case diagram in Figure 6 below shows the high level goals of the server. The server takes the input xml files generated by the GUI tool and sends them to the client as xml files. It receives images after the computation from the clients. The server uses the run file generated from the GUI tool, establishes connection with the client using the web service, sends the run xml file to the client and then receives images from clients (generated by Compucell).

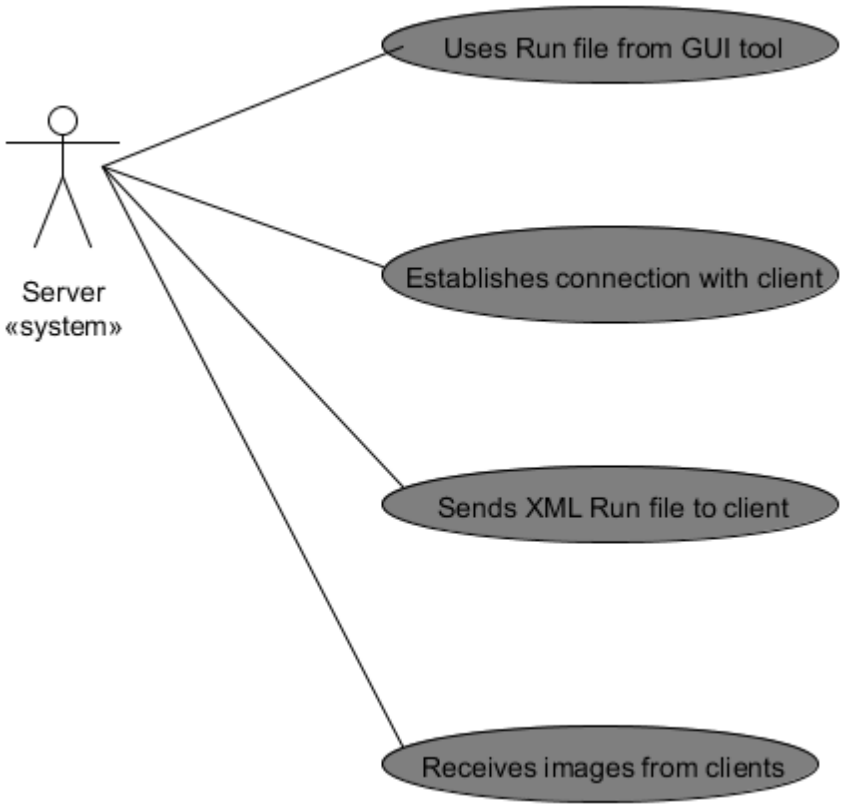



**Figure 6: Use case for server.**

The use case diagram in Figure 7 below shows the high level goals of the client. The client takes the input run xml file for a particular configuration sent by the server and runs the simulation using a third-party tool. It then sends the images generated by the simulation back to the server.

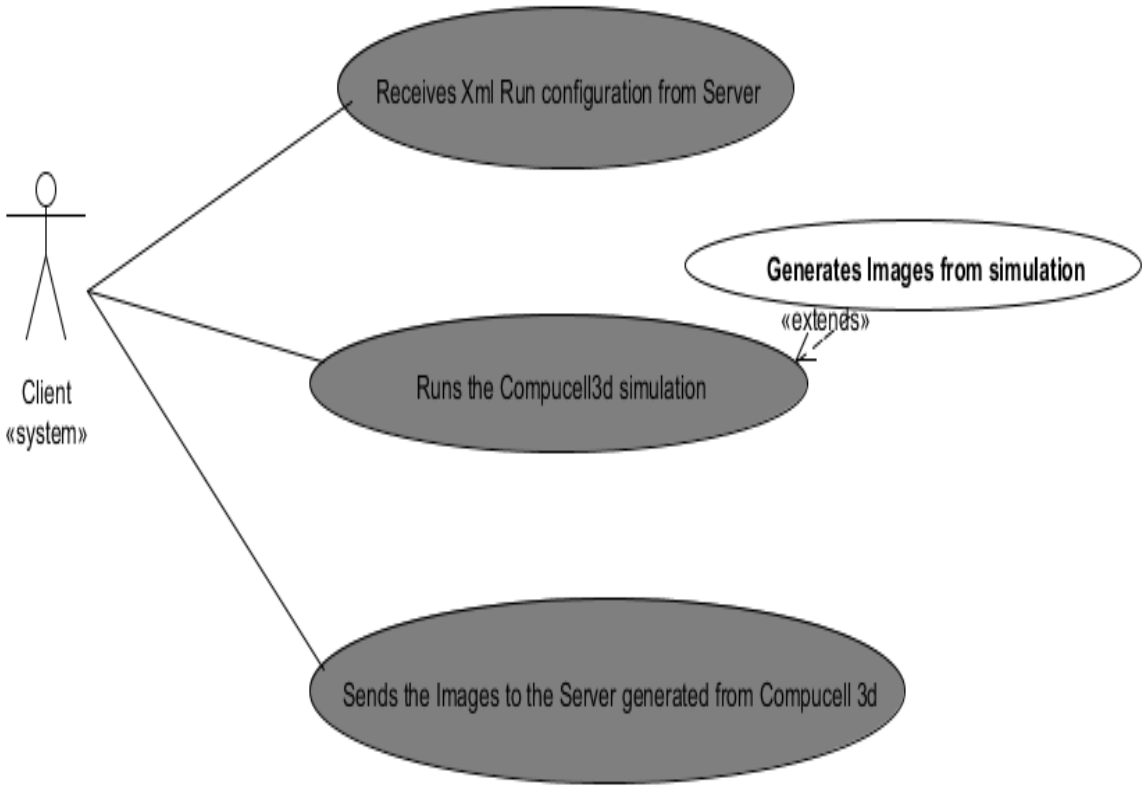



**Figure 7: Use case for client.**

The use case diagram in Figure 8 below shows the goals of the simulation. The overall server-client simulation in the use case below describes the system interactions.

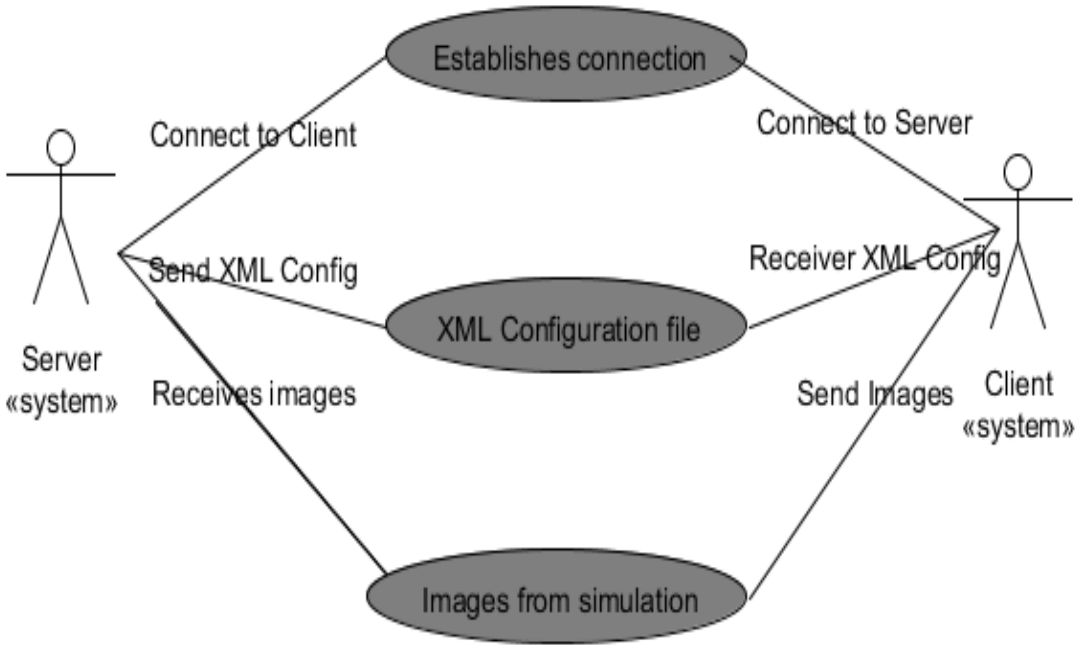



**Figure 8: Use case for system simulation.**

## 2.3 Functional Requirements

Functional requirements capture the intended behavior of the system, i.e., what the system will do. Each requirement fulfills the goal of what the system should be doing for the simulation. This is basically expressed as the tasks the system has to perform. Functional requirements are the high level details with which the system is implemented. Each requirement maps to a feature in the system that helps the developer in the implementation. This step is important for defining the exact goals of the system without which the system would not perform its intended role.

The functional requirements for the system are as follows:

**XML Generation:** The system must be able to generate xml files using a simulation file and an experiment file. This xml file is the generated run file that is used as the input for the overall simulation.

**Session Management**: The client and server must successfully establish a session to maintain proper communication.

**Robustness**: The system must be robust under all circumstances. Failure of a client causes the same configuration (run xml) to be executed in a different machine. Thus, the server must be fail-safe under all circumstances.

**Timeliness**: The simulation in the clients must respond in a timely fashion. Thus, the clients send images back to the server within a specific time upon receiving the xml configurations.

**Efficient Overhead**: The system is distributed in nature without any overhead at a particular entity, viz., server or client in the system.

**Atomic Nature:** All transactions in the system should be atomic in nature, viz., transfer of xml files or images back to the server must happen in a single operation without any interruption.

**Efficient Processing:** The system effectively utilizes CPU power by parallelizing tasks among the CPU cores, thus optimizing its overall performance.

**Load Balancing:** The tasks are distributed across the network equally among all clients. This balances the load of the entire network.

**Throughput:** The system throughput is limited by the bandwidth of the network. The throughput is tested to be efficient for data transfer across the network.

**Resource Utilization:** The system resources are utilized efficiently. The clients run simulation in all its cores efficiently.

## 2.4 System Analysis

This section focuses on creating an analysis model that is equivalent to a conceptual model in the system. Implementation details of the system are not considered here but are discussed in Chapter 4. The analysis phase acts as a prerequisite to the design phase in an object-oriented system. This phase indicates the analysis of the system based on an analysis level class diagram.

Figure 9 below shows the analysis level class diagram for the xml GUI. The xml GUI generates xml run files from an experiment file and a simulation file.

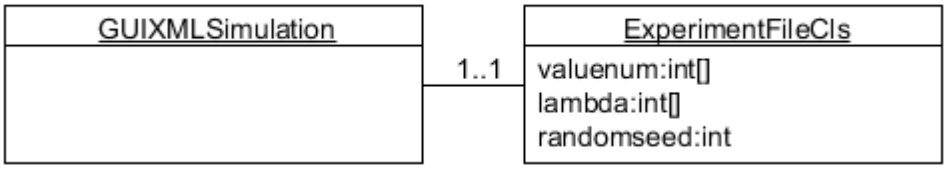



**Figure 9: Analysis level class diagram (GUI).**

The GUIXMLSimulation class is the GUI class that reads an input experiment file consisting of values and a simulation file that serves as a template for generating the run files generated by replacing the variables in the template by values from the experiment file.

Figure 10 below shows the analysis level class diagram for the server. The server sends xml files to the client and receives images from the client. MasterWebService is the main web service invoked by the web server. It associates with XMLsimulationDoc class. This class generates the xml run document. This information is passed over to the client.

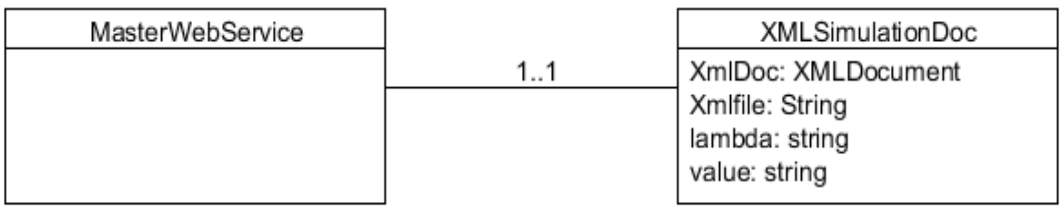



**Figure 10: Analysis level class diagram (server).**

Figure 11 below shows the analysis level class diagram for the client. The client logic is split into two parts. The first part receives the xml files from the server Web service. The second part sends the images back to the server after computation of the simulation.

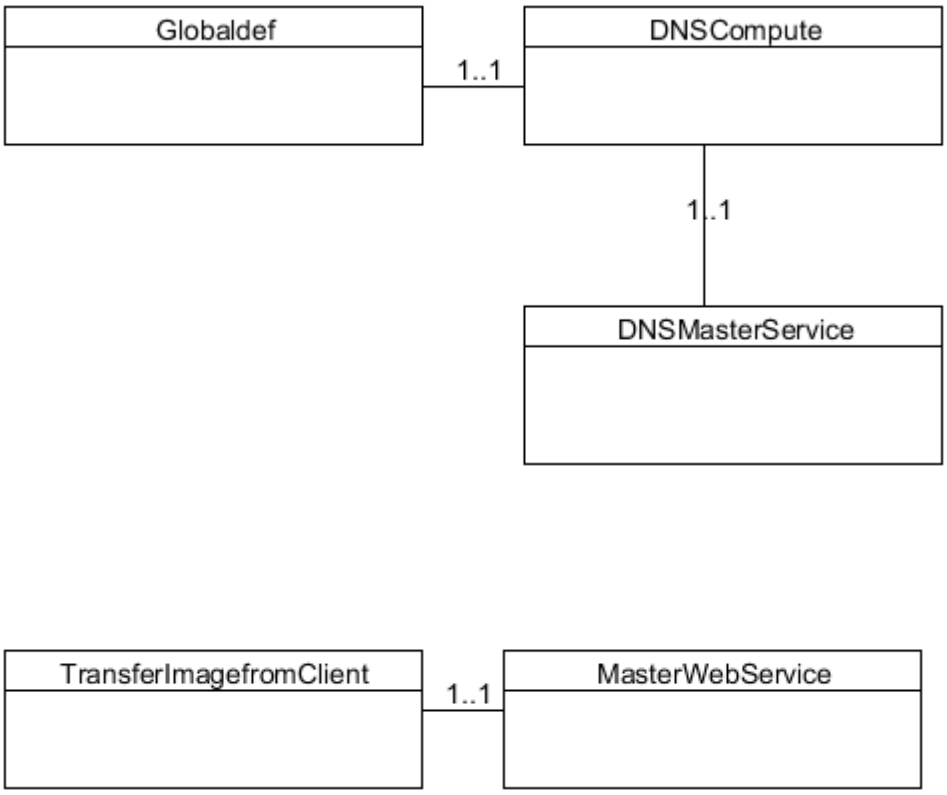



**Figure 11: Analysis level class diagram (client).**

Globaldef contains the global definitions that are used by all other classes in the client package. DNSCompute has the core logic for receiving the xml file and sending the file over to Compucell 3D for simulation. DNSMasterService is the instance of the server that is obtained by adding a web reference to the client.

The TransferImagefromClient class references the MasterWebService to send the images back to the server after computation. Splitting the client logic into two parts decouples the dependency of two different functionalities, viz., getting the xml files and sending back the images to the server.

# CHAPTER 3

# DESIGN

## 3.1 Design Constraints and Considerations

The parallel computation system poses a number of consideration and constraints in order for it to operate successfully. This is required for the optimum performance of the system. The constraints and considerations are as follows:

**Operating System**: This model is constrained to the Windows operating system. It is developed, tested and deployed for Windows only.

**Distributed:** The system is meant to work in a distributed environment in order to enhance its efficiency and throughput.

**.NET Architecture:** The system relies on the .NET architecture. The heart of the application runs a .NET web service.

**Bandwidth:** The performance of the system depends on the network bandwidth and speed. The receipt of images from various clients poses a bandwidth constraint (rates of image transfer: 12mb of images /2 minutes).

**Security:** The web service is validated for its parameters.

## 3.2 System Architecture

The system architecture describes the communication between the modules in the client,, server, and GUI. The communication is described in terms of block diagrams.

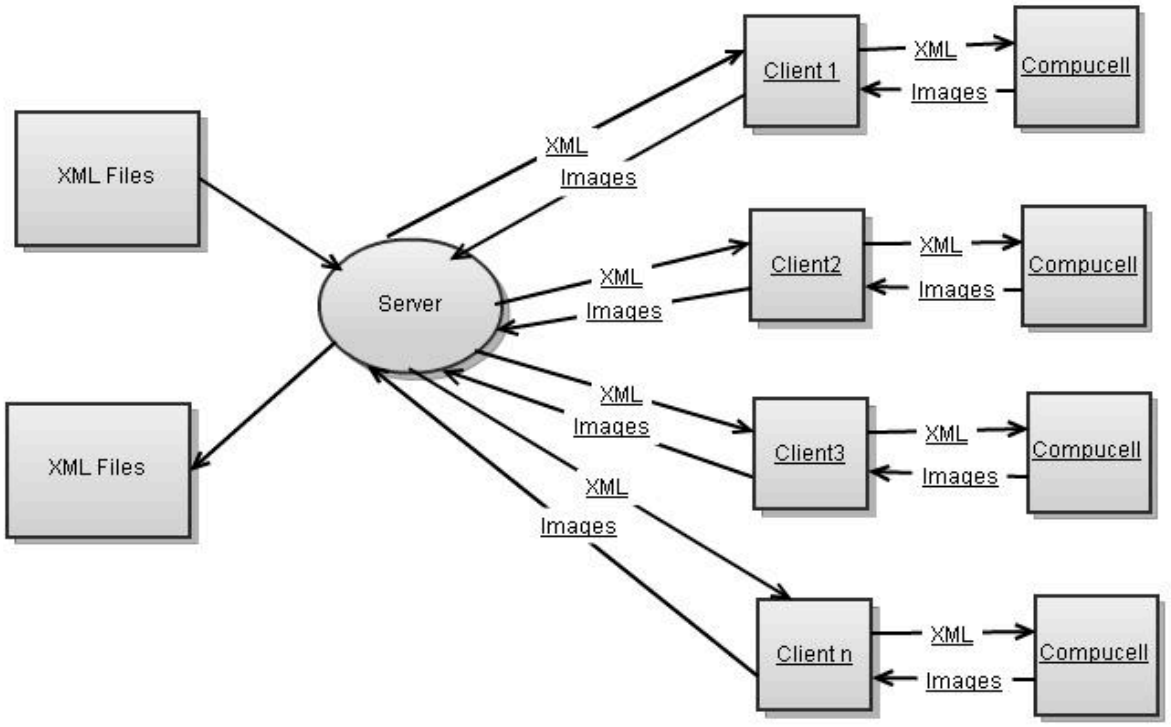



**Figure 12: System architecture.**

Figure 12 shows the architecture for the system. The server sends the generated xml files to each client and in turn receives images from the client. The above xml files in the server are generated using a GUI tool that takes an experiment xml file and a simulation file as inputs and generates xml run files. The following sections details more about the experiment file and the simulation file. Each run file is an input to the simulation system in the client that generates images after each run. These images are stored into a folder in the client from where they are sent to the server. The folder naming convention contains the lambda and the value for which the simulation was run.

## 3.3 XML Design

This section details the xml design for the experiment xml and a sample simulation xml. The content of the experiment xml file is as follows.

**<experiment>**

**<ValueNum value ="1 2 3 4 5 6">**

**</ValueNum>**

**<lamda lamda = "33 21 13">**

**</lamda>**

**<RandomSeed seed = "1225">**

**</RandomSeed>**

**</experiment>**

The values in the experiment file are multiplied out, and all possible permutations are generated by the server and executed on the client. Lambda, value, and seed are examples of some possible variables.

The simulation file is a template that corresponds to a particular simulation in Compucell 3D. The content of a sample simulation file is as follows.

```
<Compucell 3D>
 <Potts>
  <Dimensions x="55" y="55" z="1"/>
  <Anneal>0</Anneal>
  <Steps>1000</Steps>
  <Temperature>15</Temperature>
  <Flip2DimRatio>1.0</Flip2DimRatio>
  <Boundary_y>Periodic</Boundary_y>
 </Potts>
 <Plugin Name="PlayerSettings">
       <Rotate3D XRot="27" YRot="-11"/>
 </Plugin>
<Plugin Name="CellType">
    <CellType TypeName="Medium" TypeId="0"/>
    <CellType TypeName="Amoeba" TypeId="1"/>
    <CellType TypeName="Bacteria" TypeId="2"/>
 </Plugin>
 <Plugin Name="Volume">
  <TargetVolume>25</TargetVolume>
  <LambdaVolume>15.0</LambdaVolume>
```

```
 </Plugin>
 <Plugin Name="Surface">
  <TargetSurface>25</TargetSurface>
  <LambdaSurface>2.0</LambdaSurface>
 </Plugin>
 <Plugin Name="Contact">
  <Energy Type1="Medium" Type2="Medium">0</Energy>
  <Energy Type1="Amoeba" Type2="Amoeba">15</Energy>
  <Energy Type1="Amoeba" Type2="Medium">8</Energy>
  <Energy Type1="Bacteria" Type2="Bacteria">15</Energy>
  <Energy Type1="Bacteria" Type2="Amoeba">15</Energy>
  <Energy Type1="Bacteria" Type2="Medium">8</Energy>
  <NeighborOrder>2</NeighborOrder>
 </Plugin>
<Plugin Name="Chemotaxis">
  <ChemicalField Source="FlexibleDiffusionSolverFE" Name="FGF">
          <ChemotaxisByType Type="Amoeba" Lambda="$Lambda"/>   <!-- 3 !-->
          <ChemotaxisByType Type="Bacteria" Lambda="$Lambda"/>   <!-- 2 -->
  </ChemicalField>
 </Plugin>
 <Steppable Type="FlexibleDiffusionSolverFE">
    <DiffusionField>
      <DiffusionData>
        <FieldName>FGF</FieldName>
        <DiffusionConstant>0.000</DiffusionConstant>

        <DecayConstant>0.0</DecayConstant>

<ConcentrationFileName>Demos/amoebae/amoebaConcentrationField_2D.txt</ConcentrationFileName>
      </DiffusionData>
   </DiffusionField>
 </Steppable>
 <Steppable Type="PIFInitializer">
   <PIFName>Demos/amoebae/amoebae_2D.piff</PIFName>
 </Steppable>
</Compucell 3D>
```

The variables in the above file is denoted by $variable. If the variable is named $ Lambda, it gets the value of Lambda from the experiment file. Otherwise, the $variable is replaced by the value from the experiment file. The $seed or $randomseed corresponds to the random seed variable in the experiment file. Each value in the experiment file corresponds to a new file with that value. The Lambda= “$Lambda” in the above file is the variable. This is replaced by the Lambda value. The file that results after replacing these values is shown below. The file obtained after the replacement of the variables with the values in the experiment file is the run file.

```
<Compucell 3D>
<Potts>
<Dimensions x = "55"  y = "55"  z = "1" />
<Anneal>0</Anneal>
<Steps>1000</Steps>
<Temperature>15</Temperature>
<Flip2DimRatio>1.0</Flip2DimRatio>
<Boundary_y>Periodic</Boundary_y>
</Potts>
<Plugin Name = "PlayerSettings" >
<Rotate3D XRot = "27"  YRot = "-11" />
</Plugin>
<Plugin Name = "CellType" >
<CellType TypeName = "Medium"  TypeId = "0" />
<CellType TypeName = "Amoeba"  TypeId = "1" />
<CellType TypeName = "Bacteria"  TypeId = "2" />
</Plugin>
<Plugin Name = "Volume" >
<TargetVolume>25</TargetVolume>
<LambdaVolume>15.0</LambdaVolume>
</Plugin>
<Plugin Name = "Surface" >
<TargetSurface>25</TargetSurface>
<LambdaSurface>2.0</LambdaSurface>
</Plugin>
<Plugin Name = "Contact" >
<Energy Type1 = "Medium"  Type2 = "Medium" >0</Energy>
<Energy Type1 = "Amoeba"  Type2 = "Amoeba" >15</Energy>
<Energy Type1 = "Amoeba"  Type2 = "Medium" >8</Energy>
<Energy Type1 = "Bacteria"  Type2 = "Bacteria" >15</Energy>
<Energy Type1 = "Bacteria"  Type2 = "Amoeba" >15</Energy>
<Energy Type1 = "Bacteria"  Type2 = "Medium" >8</Energy>
<NeighborOrder>2</NeighborOrder>
</Plugin>

<Plugin Name = "Chemotaxis" >
<ChemicalField Source = "FlexibleDiffusionSolverFE"  Name = "FGF" >
<ChemotaxisByType Type = "Amoeba"  Lambda = "1" />
<ChemotaxisByType Type = "Bacteria"  Lambda = "1" />
</ChemicalField>
</Plugin>
<Steppable Type = "FlexibleDiffusionSolverFE" >
<DiffusionField>
<DiffusionData>
<FieldName>FGF</FieldName>
<DiffusionConstant>0.000</DiffusionConstant>
<DecayConstant>0.0</DecayConstant>
<ConcentrationFileName>
Demos/amoebae/amoebaConcentrationField_2D.txt
</ConcentrationFileName>
</DiffusionData>
</DiffusionField>
</Steppable>
<Steppable Type = "PIFInitializer" >
<PIFName>Demos/amoebae/amoebae_2D.piff</PIFName>
</Steppable>
</Compucell 3D>
```

## 3.4 Server Design

The server consists of two main parts, viz., the GUI tool and the web server. The GUI tool is responsible for generating the run xml file. The web server sends the run file to the client and receives the images after the computation at the client.

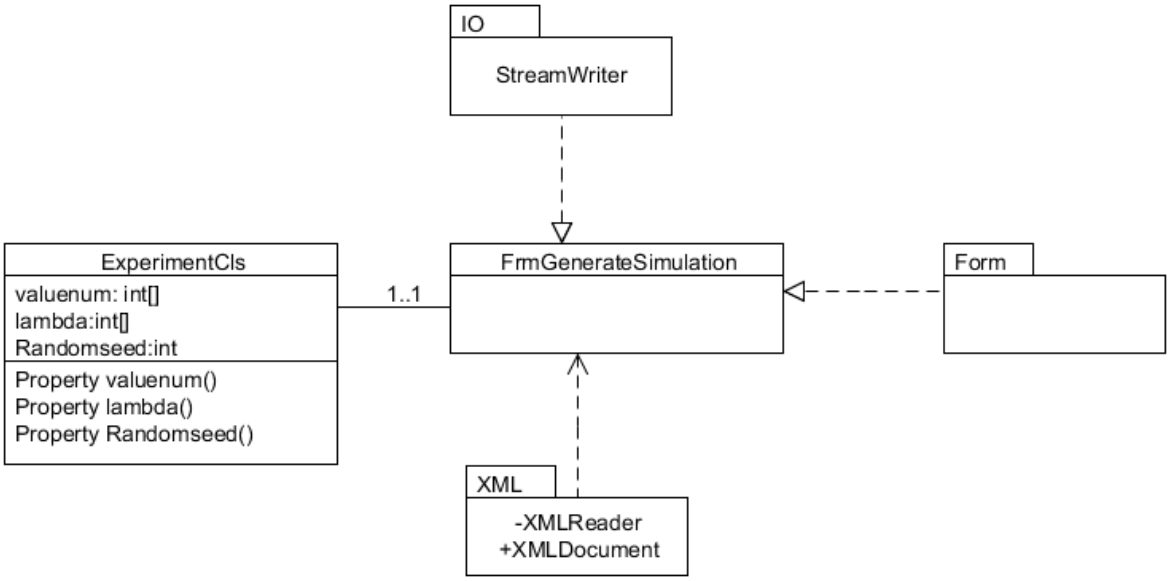



**Figure 13: GUI class design.**

Figure 13 shows the detailed GUI class design. FrmGenerateSimulation is the main GUI class. It uses the packages/namespaces IO for writing an xml file, form for displaying the GUI, and XML for reading the xml files. ExperimentCls declares variables like value, lambda and randomseed. This is instantiated in the main class where it reads the experiment xml and stores the variable values in the corresponding instance variable. The class is later used for generating run xml files.

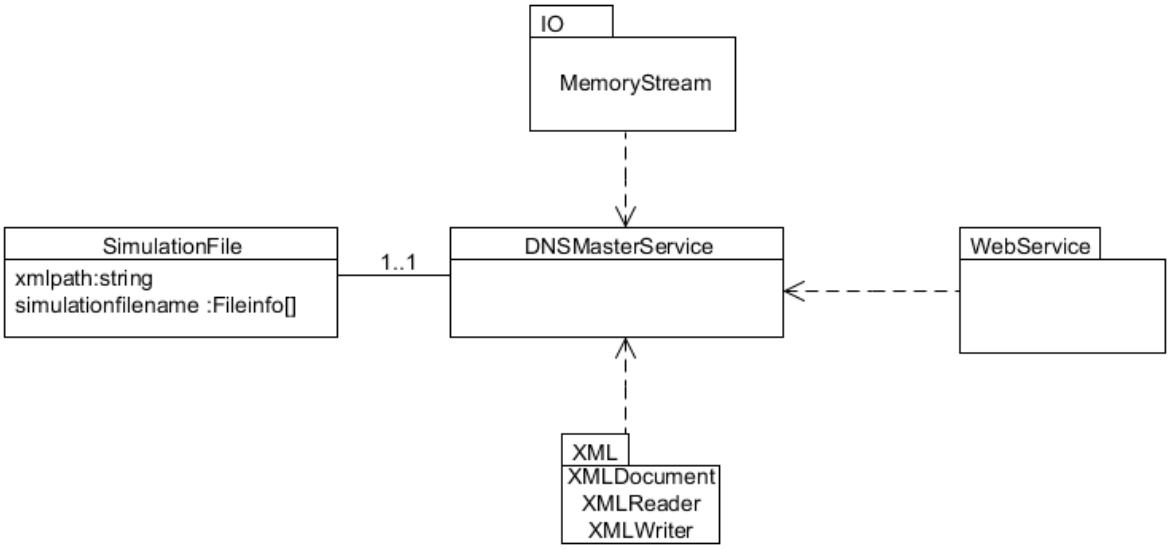



**Figure 14: Web server class design.**

Figure 14 shows the web server class design. DNSMasterService is the main class wherein all the web methods are defined. The main class references the IO, XML, and Webservice namespaces. The IO namespace contains memorystream class which is used to transfer images across the network. The main class implements the Webservice package. The XML namespace has the XMLDocument for creating xml, XMLreader for reading xml documents, and XMLwriter to write an xml file.

## 3.5 Client Design

The client interfaces with the web server for getting the xml file and also acts as an input module to the simulation program, in other words CompuCell3D generates images after each simulation run. The client logic is split into two modules. The buildXml module gets the

xml file from the server. The TxfrImage module sends the images back to the server after computation.

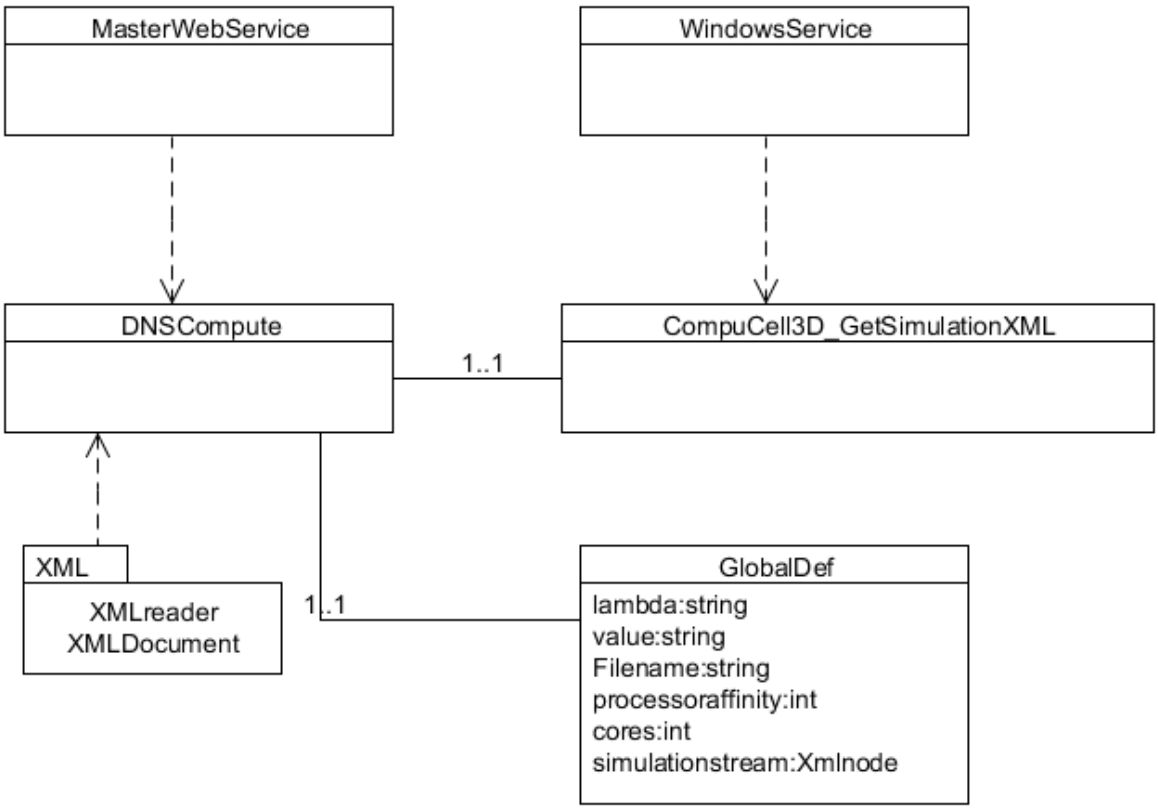



**Figure 15: Get simulation client class design.**

Figure 15 shows the class design for the get simulation file module. The core logic is implemented in the DNSCompute Class. The class references the xml namespace to read documents using the XMLreader and create xml documents using the XMLDocument class. Globaldef is the global definition file that is instantiated by the DNSCompute class. The core class also references the web service hosted by the server to access its web methods. The

main interface calls the DNS compute function as many times as the number of cores in the client system. This parallelizes the computation to get more throughputs. The client is set to automatically start and connect to the server using a Windows service. The Windows service calls the GetSimulationXML class which in turn calls the DNS compute. The xml files are saved to a location in the client. The client then calls the Compucell3D program with the generated xml file as the input parameter. This final command results in images being generated by Compucell3D during its simulation of the tissue. These images are handled using the TxfrImage module back to the server.

Figure 16 shows the class design of the Txfr image class design. The core logic is performed by the ImageTxfr_fromClient class. The IO namespace is referenced in the main class for accessing the directory class. The directory class is used to save images in specific folders. The folders are named using the lambda and value used for the particular simulation. The client starts with a Windows boot using the Windows service. The Windows service invokes the ImageTxfr_fromClient class which contains the core logic.

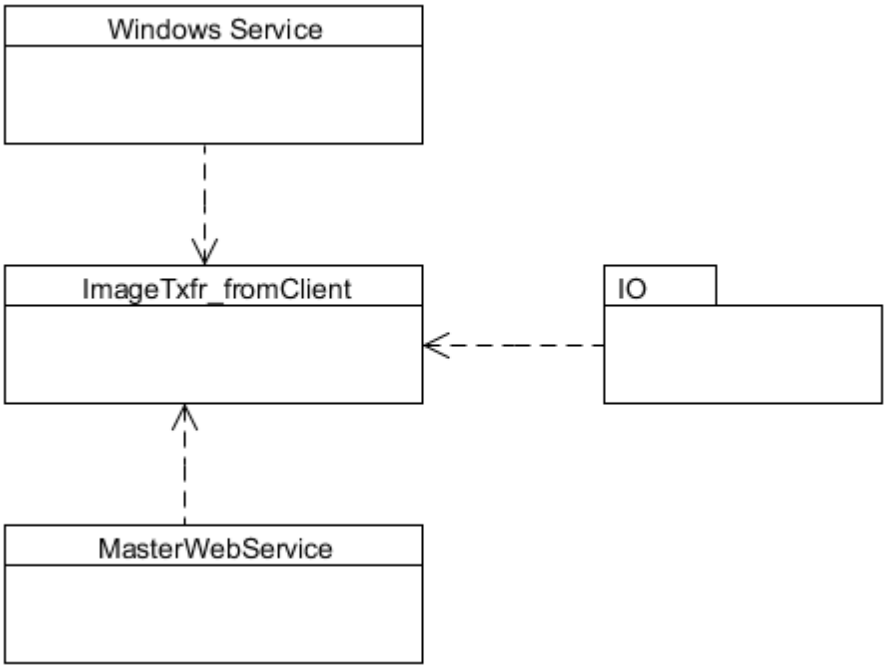



**Figure 16: Txfr image class design.**

# CHAPTER 4

# IMPLEMENTATION

## 4.1 Implementation Overview

The development environment used for the project is Visual Studio 2008. The web server used for the project is IIS 5.1. The implementation language is C#. The main namespaces used are xml, IO, and web services.

## 4.2 Server Implementation

The server implementation is divided into two parts, viz., the GUI implementation and the web server implementation. The GUI tool generates the run file from the experiment file and the simulation file.

The GUI tool uses the xml namespace to access the xml reader for reading xml documents. The xml document class is used for creating xml documents. The GUI module uses the following methods. XML RunFileDialog is an OpenFileDialog type that allows selection of xml files, and it is used to select the experiment file, as shown in the example below.

```
XMLRunFileDialog.ShowDialog()
XMLRunFileDialog.InitialDirectory = @"c:\"
XMLRunFileDialog.Filter = "XML|*.xml"
```

XmlTextReader is used to read an xml document. The xmltxtreader.read() executes until the control reaches end of the file. The Nodetype property checks if each line in the xml is an element or text or attribute. See the example below.

```
XmlTextReader xmltxtreader = new
XmlTextReader(tbSearchEngineTemplate.Text)

xmltxtreader.Read()

xmltxtreader.NodeType == XmlNodeType.Element or XmlNodeType.Text

xmltxtreader.MoveToNextAttribute()
```

Streamwriter creates a file stream to which it writes and saves the contents of an xml. The file is named after the value and lambda value and is saved as an xml file. See the example below.

```
StreamWriter writer = new StreamWriter( GeneratedSimulationfilepath +
"//tempdata" + "value_" + j + "_lamda_" + k + ".xml");

Writer.write(contents)
```

The webserver has the following methods:

```
[WebMethod(EnableSession = true)]

public XMLSimulationClass GetXMLDocumentSimulationFile()
```

The above declaration exposes the GetXMLDocumentSimulationFile as a webmethod. This method is invoked by the client to get the xml document from the server. The XMLSimulationClass contains the XMLfile, value of lambda, and variable value.

The PutImage method is accessed from the client for transferring images from the client to the server. See the example below.

```
[WebMethod(EnableSession = true)]
PutImage(byte[] ImgIn, string Filename, string DirectoryName)
```

## 4.3 Client Implementation

The client logic is split into two modules. The first module receives the xml file from the server. The second module sends the images back to the server. The Compucell BuildXML module has the following two methods. The first method gets the number of cores in the system. This is used to invoke the server method to get the xml file as many times as number of cores. The second method creates an instance of the web service. The web service is referenced in the client by adding a web reference to it.

```
New System.Management.ManagementObjectSearcher("Select * from
Win32_Processor").Get()
```

and

```
DNSMasterService.DNSMasterService dns1 = new
DNSMasterService.DNSMasterService();
```

The XmlWriter settings and the conformance level are set before creating an xmlwriter.

```
XmlWriterSettings xmlsettings = new XmlWriterSettings();
xmlsettings.ConformanceLevel = ConformanceLevel.Document;
```

```
XmlWriter xmlwritersimulationfile =

xmlWriter.Create(globaldef.SimulationFileName);
```

Then, the XmlWriter is saved into an xml document using the following command

```
globaldef.SimulationFileStream.WriteTo(xmlwritersimulationfile);

xmlwritersimulationfile.Close();
```

The Imagetxfr module has the following methods:

```
string homedir =
Environment.GetFolderPath(System.Environment.SpecialFolder.ApplicationData)

string directory = Directory.GetParent(homedir).FullName
Directory.SetCurrentDirectory(directory + "\\CC3DWorkspace")
```

The images are saved in the user's folder after the computation in Compucell 3D. Thus, the homedir gets the user\CC3Dworkspace directory.

```
if(f.Name.Contains(".png"))
                {
                FileStream fs = new
FileStream(f.FullName,FileMode.OpenOrCreate, FileAccess.Read);
                Byte[] img = new Byte[fs.Length];
                fs.Read(img, 0, Convert.ToInt32(fs.Length));

                if (!dns1.PutImage(img, f.Name, di.Name))
                {

                   continue;
                }
```

There is a check done for png files. The image is read using a file stream which is then transferred into a byte. The image is transferred as a byte to the server. This is done

using the putImage web service method. The continue statement above continues to transfer the next image if the previous call to putImage fails.

```
foreach (string d in Directory.GetDirectories(Directory.GetCurrentDirectory()))
        {

           DirectoryInfo di = new DirectoryInfo(d);
           Thread.Sleep(3000);
           di.Delete(true);

        }
```

After the image is transferred to the server, the images are deleted from the client. This avoids overhead on the network while processing new requests from the client.

**C:\Program Files\Compucell 3D>Compucell 3D.bat -i**

**C:\Client\XMLFiles\xmlsimulation**

**file_lamda_1.xml_value_1_KARTHIKIYER.xml -o C:/XMLFiles**

where

**Compucell 3D.bat: Batch file to run Compucell 3D**

**i: Input xml file**

**o:Output directory for the files.**

Compucell 3D simulation is processed by passing command line calls to invoke the simulation with the generated xml file as the argument.

# CHAPTER 5

# TESTING

## 5.1 Testing Overview

Testing is second only to the design phase in its importance to the project. It is done to test and validate proper functionality of the system. It also validates each requirement the system is to fulfill. The testing for the system is setup in two ways. The first method is localized testing wherein the server and client exist in the same system. This tests the functionality of the system and eliminates the networking factor in the system. The second method for testing involves a test bench setup wherein an actual web server is setup and multiple clients add references to the web server in order to receive and transmit data.

## 5.2 Local Testing

In the case of local testing, the server and client exist in the same system. This validates the system's goals. See Figure 17. The network factor is eliminated in this case. The server is hosted on localhost. The files are present under the C:/inetpub/wwwroot folder. That is the default path for the web server.

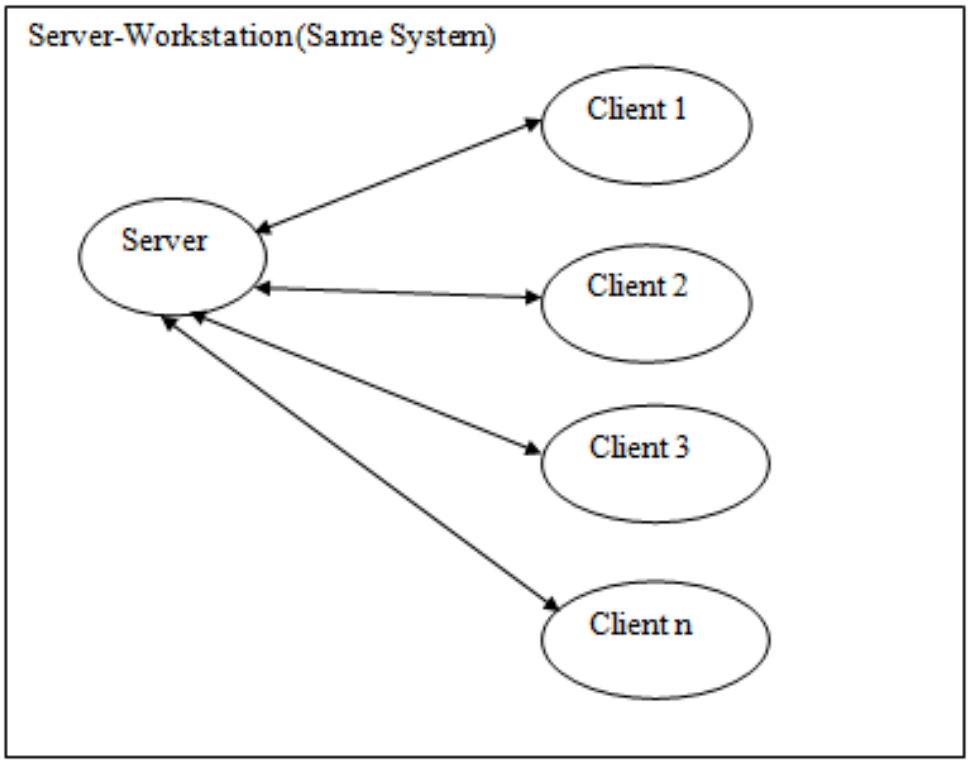



**Figure 17: Local test setup.**

## 5.3 Test Bench Setup

The test bench setup involves setting up an actual server and a DNS record (domain name). Figure 18 provides an illustration. The domain name helps in easy identification on the Web.

The server is hosted using a third-party vendor. The files are present under the C:/inetpub/wwwroot folder. That is the default path for the web server.

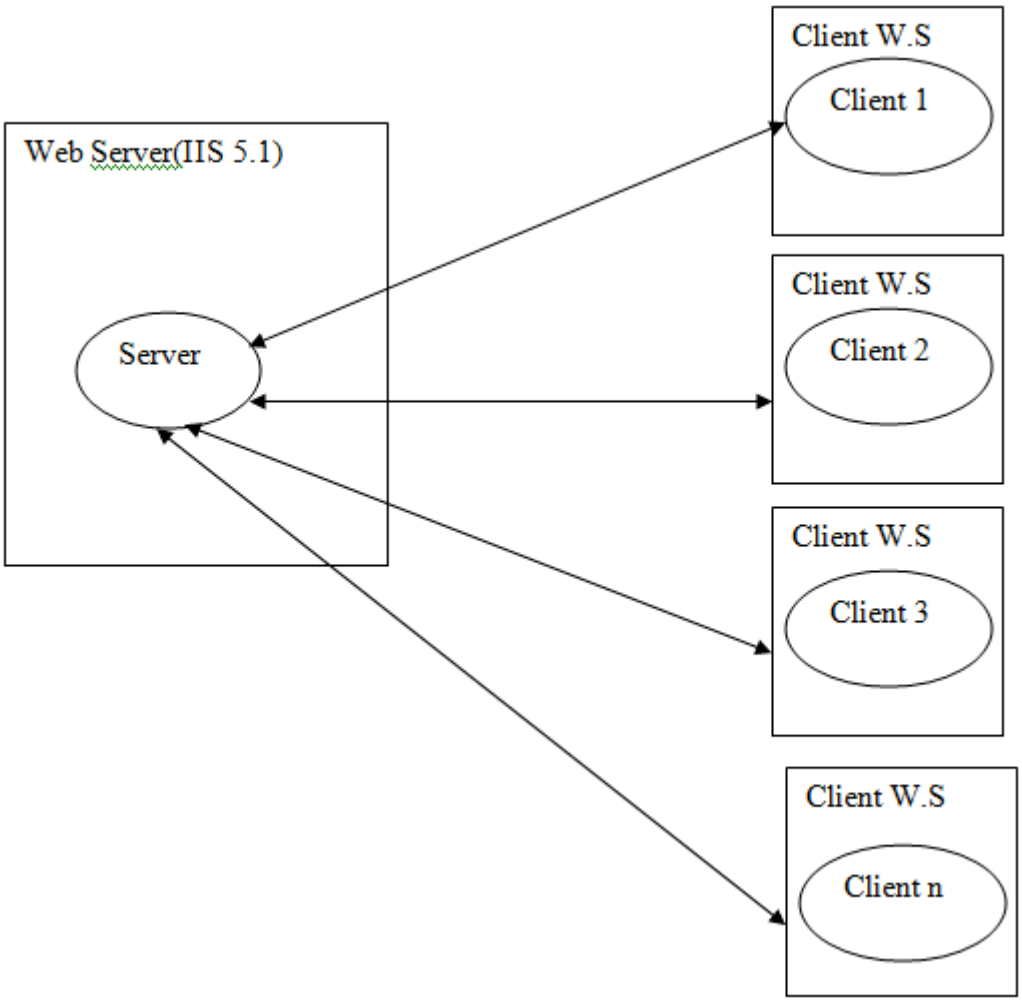



**Figure 18: Test bench setup.**

## 5.4 Test Cases

The experiment was performed for multiple scenarios. Some of the scenarios were tested for correctness on Localhost. The test setup was then extended to include external clients. In each of the following situations, the data size transferred to the server increased gradually. The following section explains each test case in more detail.

### 5.4.1 Test Case 1: Localhost

In this particular test case, the client and server exist in the same system. This is a validation test case in order to validate the functionality of the system.

**Configuration 1:** This configuration involves 2 simulation xml files generated using the GUI tool. The simulation chosen in this case was amoebae 2D. The experiment file had the enumeration details for the simulation. They were as follows:

**<ValueNum>**
**ValueNum: 1,2,3,4,5,6**
**</ValueNum>**
**<lamda>**
**lamda:1,2,3**
**</lamda>**
**<RandomSeed>**
**Seed:1225**
**</RandomSeed>**

The simulation files generated using the GUI tool had a lambda of 1 and value of 1, lambda of 1 and value of 3. It took 13 seconds to transfer 856kb of image and text data.

**Configuration 2:** For the 2nd configuration, the configuration for the lambda and values were as follows:

Lambda: 0 , Value :0

Lambda: 1, Value: 2

Lambda: 2, Value: 2

Cellsort 3D simulation

The time taken for transferring 1.73Mb ofi and text data was 21 seconds.

**Configuration 3:** For the third configuration, the configuration for lambda and values were the same. More simulations like Cellsort 2D, Cellsort 3D, multiscale simulation, and diffusion were added to the existing configurations. The time taken for transferring 2.81 Mb of image and text data was 26 seconds.

Figure 19 shows a graphical plot for the three configurations on Localhost.

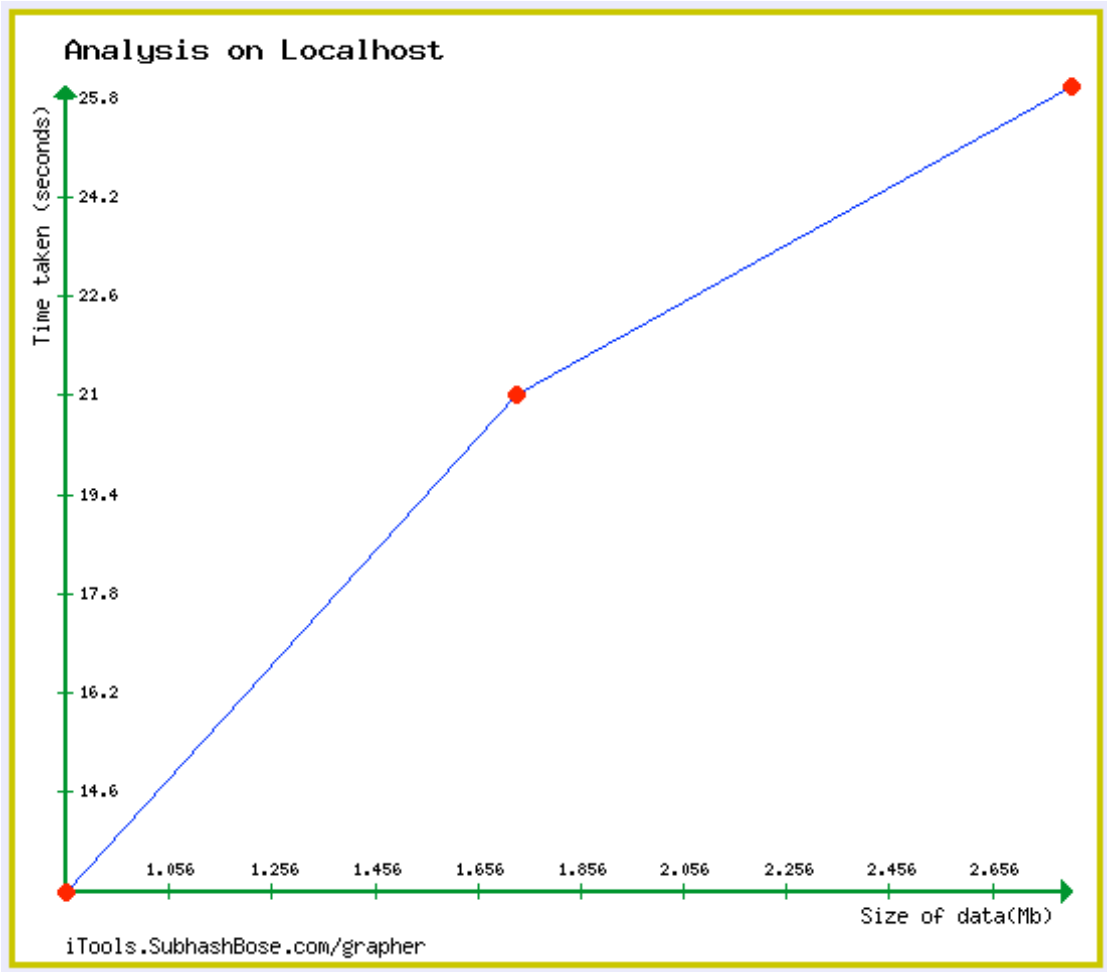



**Figure 19: Graphical analysis on Localhost.**

### 5.4.2 Test Case 2: One Client

In this test case, the analysis is done for the same configurations as that on Localhost but with one external client accessing the server through a web service.

**Configuration 1:** This configuration was the same as Configuration 1 for Localhost. For transferring 856kb of image and text data, the time recorded was 46 seconds (given network speed and latency).

**Configuration 2:** This configuration was the same as Configuration 2 for Localhost. For transferring 1.73Mb of image and text data, the time recorded was 69 seconds.

**Configuration 3:** As in Configurations 1 and 2 above, this Configuration was the same as Configuration 3 for Localhost. For transferring 2.81Mb of image and text data,the time recorded was 108 seconds.

Figure 20 shows a graphical plot for the three configurations for a single client.

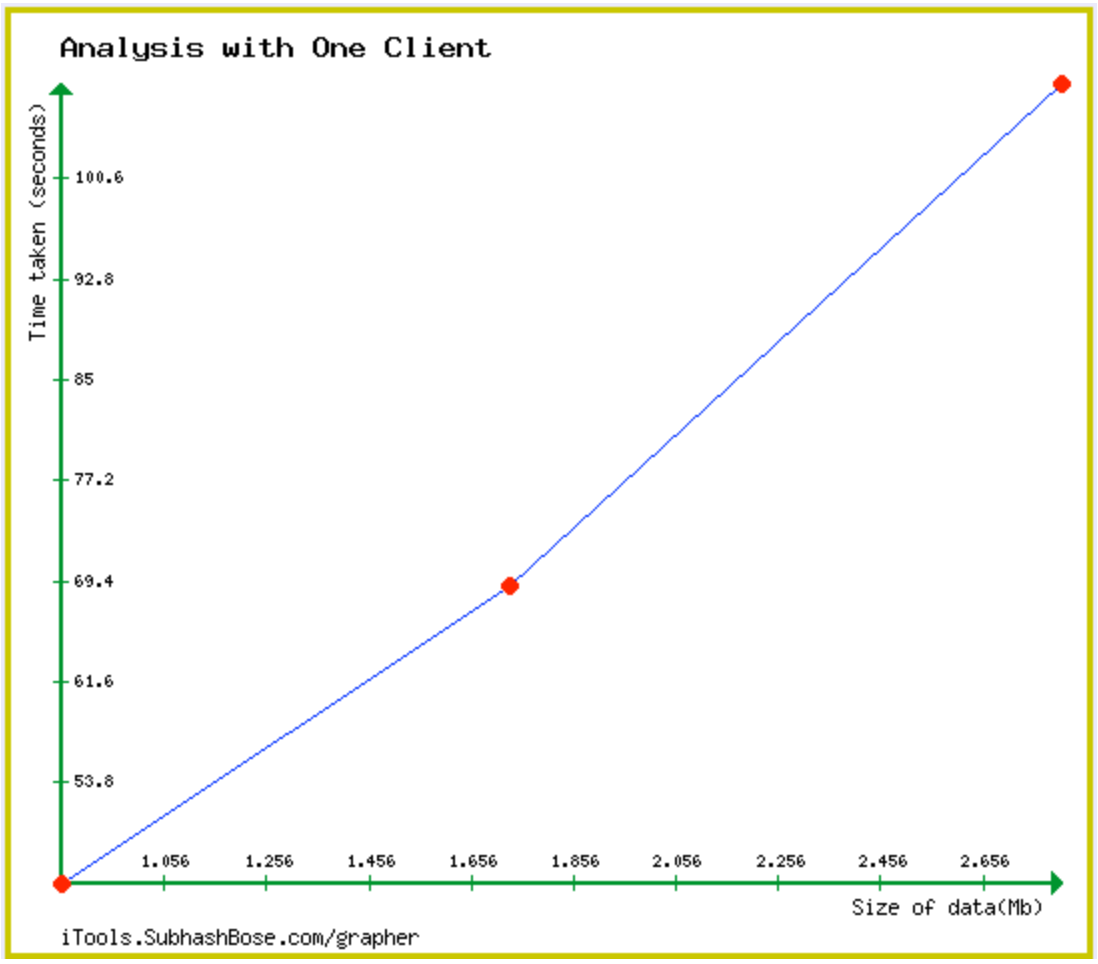



**Figure 20: Graphical analysis with one client.**

The above analysis shows that Localhost is the fastest since it is a loopback mode. With one client, the latency increases based on network bandwidth, network speed, and the distance in terms of number of hops between the server and client. For the above test case the machine had the following configuration

RAM : 2.5GB  Processing Speed: 1.6 Ghz

Operating System : MS  Windows XP SP3.

Speed test:  10.21 Mbps Download speed / 4.03 Mbps Upload speed

The below graphs were generated on computers with the following configurations

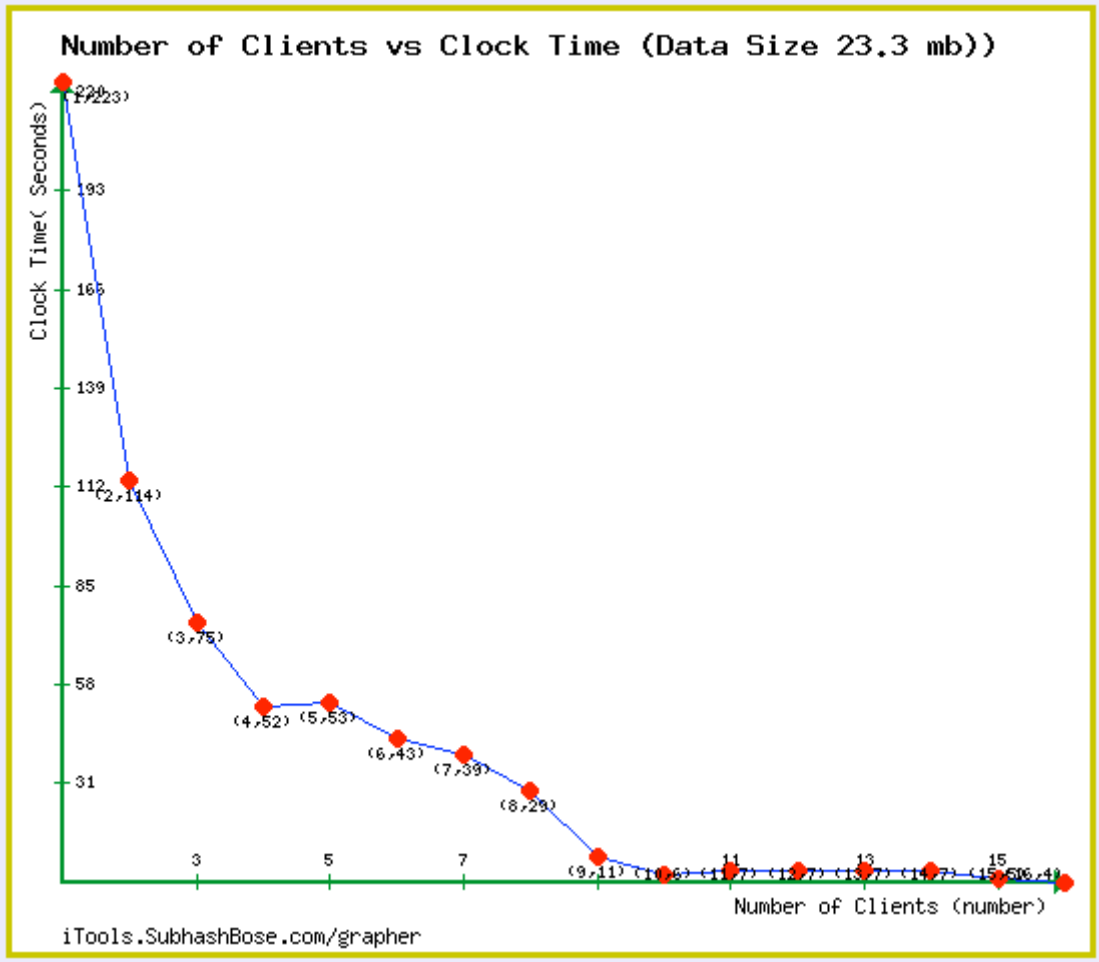
RAM : 4 GB

Processing Speed: 3.2 Ghz

Windows 7 64 bit OS

Speed test:

93.4 Mbps Download speed / 66.23 Mbps Upload speed




The above graph indicates the time taken to compute the model up to 16 clients. The clock time taken decreases with the increase in the number of clients used for the simulation.

# CHAPTER 6

# FUTURE WORK

Biological computation is very complex and requires huge computation power. In order to address this problem, in this project, we have successfully achieved performing repetitive tasks in a parallel fashion.

In the next iteration of this project, I perceive the need for a network activity monitoring tool that would monitor the live status of the tasks in each of the clients. In the current version, Compucell 3D is not entirely integrated with the application. With the next phase, it would be useful to integrate it with the .NET framework for a seamless performance. In the current work, all the cores in each client are used for computation purposes. However, optimized performance is still yet to be achieved. This could be addressed in the next phase. It may be useful to offload heavy computations on to the GPU.

# REFERENCES


[1] Wikipedia. *UML*. http://en.wikipedia.org/wiki/Unified_Modeling_Language. Acccessed July 2011.

[2] UmlLet. *Open Source UML Design Tool*. www.umlet.com. Accessed July 2011.

[3] Wikipedia. *Compucell 3D: Three-Dimensional Simulation Program*. http://en.wikipedia.org/wiki/Compucell 3D. Accessed July 2011.

[4] Wikipedia. Microsoft .NET Architecture. http://en.wikipedia.org/wiki/.NET_Framework. Accessed July 2011.

[5] Wikipedia. .NET Windows Services. http://en.wikipedia.org/wiki/Windows_service. Accessed July 2011.

[6] Wikipedia. .NET Web Services. http://en.wikipedia.org/wiki/Web_service. Accessed July 2011

[7] Wikipedia. Testing Methodologies. http://en.wikipedia.org/wiki/Software_testing. Accessed July 2011.

# APPENDICES

# Appendix A. Compucell 3D : A 3D Cell Simulation Program

Compucell 3D is a three-dimensional problem solving and simulation environment for simulating disease and development processes in multi-cell systems. Cellular Potts Model$(CPM)^2$, a lattice-based computational modeling method for simulating collective behavior (collective behavior includes cell growth, division, death , adhesion, volume and surface area) of cellular structure, is simulated by Compucell 3D. It also includes equation solvers for modeling reaction – diffusion of external chemical fields. Further, Compucell 3D integrates all the cellular models with external chemical reactions, such as secretion, and resorption, and responses, such as, chemotaxis and haptotaxis. It can even model different phenomena such as avian wing development.

Compucell 3D is basically built using C++ (Core Engine component), QT(Compucell Player) and Python. The input for Compucell 3D is usually an xml configuration file or a Python file. Compucell 3D reads from an xml configuration file and then generates a simulation by running the Compucell GUI.

The software was tested for sample simulations like Ameoba 2D simulation and secretion. Figure 21 shows the output generated from the simulation.

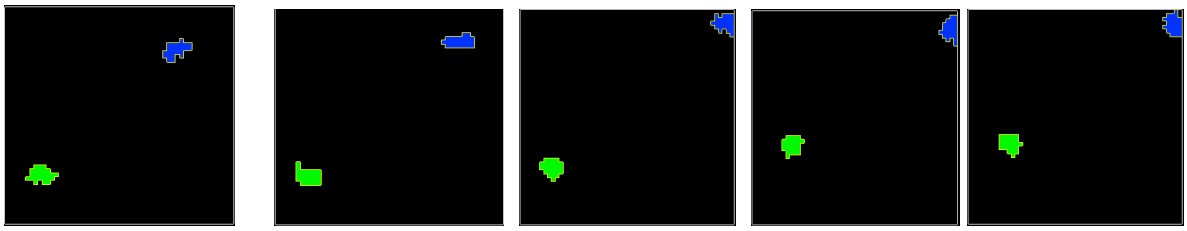


**Figure 21: Ameoba 2D secretion simulation.**

Figure 22 shows the output generated from the simulation for Cellsort 2D boundary simulation.

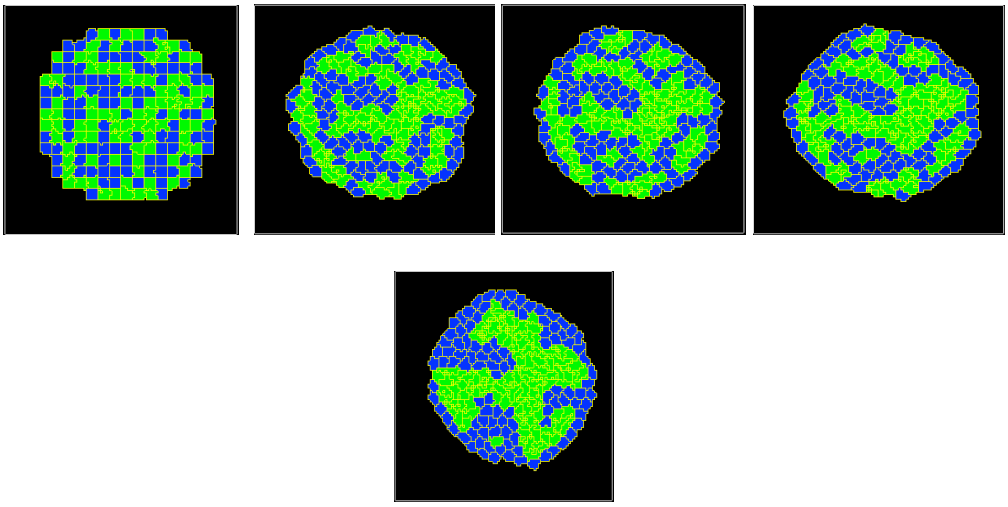


**Figure 22: Cell 2D boundary simulation.**

Figure 23 shows the simulation generated for multiscale simulation.

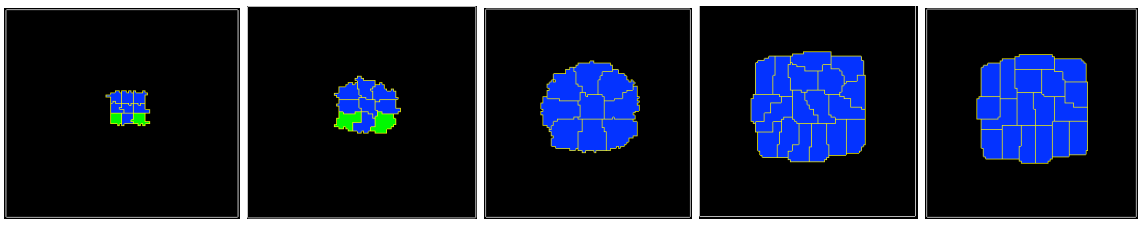


**Figure 23: Multiscale simulation.**

# Appendix B. .NET Windows Services

.NET windows services enable executables to perform functions requiring no user intervention. Windows services can be used to start an application on a Windows boot up or run a task in the background when Windows OS is running. A Windows service can be seen under services in the control panel administrative tools section or by typing services.msc in the command line.

.NET Windows service can be developed using Microsoft Visual Studio Windows service project. In such a case, the service control manager interface controls the behavior of the Windows service on start, stop, and other events in the service.

Installation of a Windows service application requires a project installer and a service installer. The service installer has a service property pointing to the service itself and a parent property that references the project installer. Also, the start type is set for the service installer property. The process installer component's account property sets the service type of the service, viz., local service, network service, etc.

The following commands are used from the command line for installing and uninstalling a windows service.

**Installutil.exe /I C:/MyService.exe**

**Installutil.exe /u C:/MyService.exe**