Computational Modeling and Simulation of the Immune System

J.K. Kalita[†], K. Chandrashekar[†], R. Hans[†], P. Selvam[†] and M.K.Newell[‡]

[†]Department of Computer Science, University of Colorado,
1420 Austin Bluffs Parkway
Colorado Springs, Colorado 80917 USA
E-mail: {jkalita, kchandra,rhans,pselvam}@uccs.edu
*Corresponding author
[‡]Department of Biology, University of Colorado, 1420 Austin Bluffs Parkway
Colorado Springs, Colorado 80917 USA
E-mail: mnewell@uccs.edu

Abstract: We have developed a software system called SIMISYS that models and simulates aspects of the human immune system based on the computational framework of cellular automata. We are motivated by the goal of modeling participants in the immune system at the cell level, simulate their interactions and infer overall system behavior. We model tens of thousands of cells as exemplars of the significant players in the functioning of the immune system, and simulate normal and simple disease situations. We present the simulation while in progress with graphical illustration of the participating cells and appropriate graphs. SIMISYS 0.3, the current version of the software, is able to model and simulate the innate and adaptive components of the human immune system. The specific players of the immune system we model are the macrophages, dendritic cells, neutrophils, natural killer cells, B cells, T helper cells, complement proteins and pathogenic bacteria

Keywords: Modeling, Simulation, Immune System, Anthrax

Reference to this paper should be made as follows: Kalita, J.K., K. Chandrashekar, R. Hans, P. Selvam and M.K. Newell (2005) 'Computational Modeling and Simulation of the Immune System With Modeling and Simulation of *Bacillus anthrax*', Int. J. Bioinformatics Research and Applications, Vol. 1, Nos. 1/2/3, pp.xx–yy.

Biographical notes: Jugal Kalita is an Associate Professor at the University of Colorado at Colorado Springs. K. Chandrashekar graduated with an MS in Computer Science from the University of Colorado at Colorado Springs in the Spring of 2005. R. Hans graduated with an MS in Computer Science from the University of Colorado at Colorado Springs in the Spring of 2004. P. Selvam is an MS student in the Department of Computer Science at the University of Colorado at Colorado Springs. M.K. Newell is an associate professor of Biology at the University of Colorado at Colorado Springs.

1 INTRODUCTION

The immune system is a collection of molecules, cells and organs whose complex interactions form an efficient system that protects the individual from potential harm and outside invaders (11). There are up to 10^{12} cells that participate in the immune system. A traditional view is that the immune system can be divided into two functionally distinct categories: *innate* (non-adaptive), and *acquired* (adaptive). These two arms of the immune response may not perform their duties independently. Innate immunity is characterized by its non-specificity, and is achieved by the actions of physical or chemical barriers, phagocytes, neutrophils, natural killer cells and complement proteins. Adaptive (or, acquired) immunity is specific, has a diversity of responsiveness and appears to maintain memory. This type of immunity is usually found only in vertebrates and is mediated by B- and T-lymphocytes, clonally distributed and characterized by specificity and memory.

The main function of the immune system is to provide specific protection from harm. Different types of immune cells play different roles in the overall immune response. Chemical signals provide communication among these cells. In this study we model features of innate and adaptive immune responses. All immune cells are modeled as classes using object-oriented technology (3). Cellular interactions are modeled based on the computational paradigm of cellular automata (6; 31; 33; 35; 36). A graphical user interface is provided so that the user can vary parameters of the simulation. A graphical display, created using the SDL library (24), provides visual images during the simulation.

The organization of this paper is as follows. Section 2 provides an overview of related research. Section 3 describes the architecture of the software system designed and implemented. Section 4 provides details of the object-oriented class structure. Section 5 discusses an XML language that we use in a prototype of SIMISYS. In Section 6 we analyze the simulation results. Section 7 discusses an extension to SIMISYS for a simple simulation of Anthrax infection. Finally, in Section 8, we discuss future directions for research.

2 RELATED RESEARCH

In spite of the enormous complexity of the immune system, several computational studies present a global and thus, necessarily simplified understanding of the system. Three main approaches have been adopted by researchers:

- Ordinary differential equations (as summarized in (25)) ,
- Qualitative, i.e., non-numeric, information for modeling (e.g.,(32)), and
- Distributed computation using cellular automata (e.g.,(16)).

Ordinary differential equations (ODE) have been traditionally used to model complex systems. Perelson and Weisbuch(25) use physical concepts and differential-equations based mathematical methods for modeling immunological problems. For example, they present models for clonal selection and affinity maturation, network models for antibody and B-cell interactions, and autoimmune diseases. Differential equations also have been used in other efforts such as for modeling virus-neutralizing immunoglobulin response(8), dynamics of co-infection of *M. tuberculosis* and HIV-1(13), the dynamics of *Plasmodium falciparum* blood-stage infection(21), change in CD4 lymphocyte counts in patients before and after administration of HIV protease inhibitor indianvir(30), and the differentiation of B lymphocytes under control of antigen(15).

However, researchers have also enumerated problems with pure ODE approaches (16). Some of the problems with ODEs are a) The ODE approach assumes large populations of essentially identical entities, which is not the case with biological cells as each cell has a unique life history that defines its interaction with the environment, b) The ODE approach gives only average behavior of the system, and c) It is difficult to model non-linear behavior.

Cellular automata(6; 31; 33; 35; 36) are discrete dynamical systems whose behavior is completely specified in local terms. They have been widely studied as examples of complex dynamical systems(10; 35; 22), originally as examples of components in a self-reproducing machine(4; 9; 10) and then within the area of *artificial life*(18). In a cellular automata model, a uniform grid represents space, with each cell containing a small amount of data. Time advances in discrete steps and simple laws of behavior are used at each step for each cell to compute its new state from that of its neighbors. The behavior of a complex system emerges from simple interactions of simple individuals following simple rules. Cellular automata are sometimes described as counterpart to ordinary or partial differential equations for describing continuous dynamical systems. There have been attempts to simulate aspects of the immune system using cellular automata(2; 5; 17; 19; 29). For example, Bezzi(1) discusses models for evolution of the immune system. In particular, he introduces a cellular automata model for studying an evolving set of individuals as well as the effects of co-evolution. Schadschneider et al.(28) simulate pedestrian dynamics with friction to validate models of emergency egress in aircraft. Weimar(34) models and simulates enzymatic reaction networks. He uses each lattice site as a container for one enzyme molecule and multiple metabolite molecules.

3 IMPLEMENTATION OVERVIEW

While developing SIMISYS, we have made simplifying assumptions about the innate and adaptive immune systems, and the communications between the two. We follow a systems biology approach(14) that requires us to model the system by understanding:

- the structure in terms of components and interfaces among the components,
- the dynamics of the system and
- the control structure.

Figure 1 gives the high level view of the immune system according to our model. Many details are purposely left out in this figure to keep it simple, and also to emphasize that ours is a proof-of-concept model and is being continuously improved. Immune cells such as macrophages, dendritic cells, neutrophils, phagocytes, and natural killer (NK) cells are created in the bone marrow. Neutrophils, the most abundant of all white cells, are recruited to the region of a pathogen's attack in an

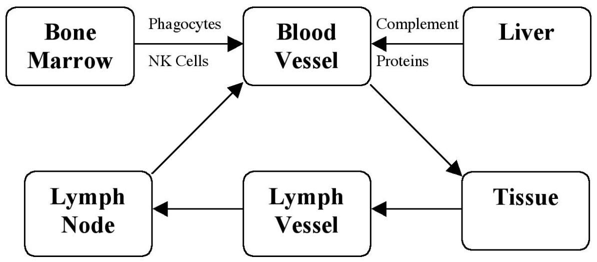

Figure 1 System Biology View of the Immune System

infected tissue based on the concentration of chemo-attractants. The requirements for activation of antigen-specific lymphocytes, either T or B, are recognition of antigen and other co-stimuli, including cytokines. NK cells need certain cytokines to be activated and kill target cells. They have been implemented as being attracted to the region of infection as a function of the amount of lipopolysaccharide chemical (LPS) produced by the antigen.

The blood vessel is the main port of entry of immune cells into the tissue. Both innate and adaptive players look for a suitable place to exit the blood vessels so that they can enter the lymph node where the lymphocytes become focused to respond to the potential invaders. There is a flow from any location in the tissue to the lymph vessel so that the immune cells responding to the invaders may move to a lymph node. To maintain a continuous movement of immune cells in the tissue (also called the *grid structure* or *grid* in our model), the blood vessels translocate the immune cells from the lymph node back to the tissue. Coordination between the blood vessels and the lymph vessel set up in the simulation has a major role in maintaining flow. For example, antibodies, secreted by the primed B cells, are created in the lymph node and translocated to the tissue. There they opsonize the bacteria. The complement proteins puncture these tagged bacteria in the tissue.

Movement of chemicals such as cytokines in the blood and tissues are modeled using diffusion. A chemical is loaded onto the grid cells at the location of the cell that secretes it. They are diffused through the whole grid depending on their respective breakdown rate and their diffusion constants. This sets up a gradient of chemicals in the tissue allowing for the movement of the immune cells in the tissue based on chemo attractants. This also allows the activation of the appropriate immune cells. based on cytokine stimulation. The behavior of certain immune cells, especially macrophages, depends on the state of their activation. In summary, the number of immune cells of various types and their movement are managed by the combined action of the blood and lymph systems and chemical gradients. Figure 7 shows the flow of control between the entities and the interactions.

3.1 Software Architecture

The interaction between immune cells and pathogens has been modeled using cellular automata. SIMISYS 0.3 is a complex software system with many interacting components. It has been developed keeping in mind that the system will evolve over time as additional complexities are added. Figure 2 provides an overview of

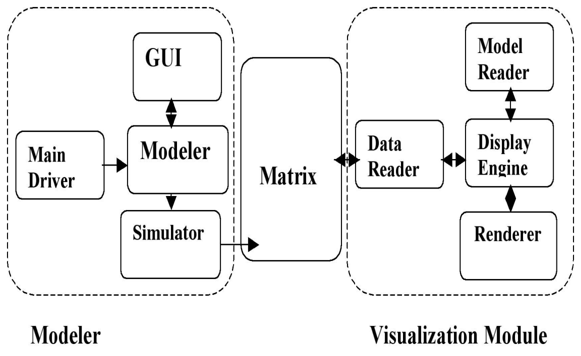

Figure 2 Software Architecture for SIMISYS

the software components that constitute SIMISYS.

There are three main components:

- The Modeler
- The Matrix
- The Visualization Engine

The main driver program brings up a graphical user interface (GUI) through which the user inputs parameters to model pathogens and immune cells. The user can vary the parameters describing the cellular players such as their initial number, life span and maximum count. The user can also provide data to set up the size of the grid and study the impact of changes on the immune response through the GUI. The emphasis of the design has been to create a highly configurable system so that specific scenarios can be modeled with ease. Some of the parameters that can be input are given in Table 3.1. Once the parameters have been specified, control of the system passes to the Modeler. The Modeler reads the values entered and starts the Simulator. The Simulator creates the Matrix, all immune cells, pathogens, blood and lymph vessels, and controls interactions among them. The Matrix models the physical space that the cells occupy. It consists of a 3D grid of cells, where the simulator places all cells and pathogens. Each entity occupies one cell. The matrix also holds chemo-attractants and diffuses them.

The Visualization Module is responsible for display and its simulation. The Data Reader reads the information from the Matrix and provides the information to be displayed to the Display Engine. The Display Engine presents this information to the graphical interface built using SDL which presents a view of the infected tissue or the lymph Matrix depending on the users interest. A separate panel for the display of the statistical results of the system is also provided.

4 Software Details

The SIMISYS Immune System simulation is implemented in C and C++. It has a multithreaded architecture based on *pthreads*(23). The images are displayed using SDL(24), a graphics library.

ENTITY NAME	PARAMETERS
Neutrophil	Number of Neutrophils, Life Span
Natural Killer	Number of Neutrophils, Life Span
Macrophage	Number of Macrophages, Life Span
Bacteria	Initial Count, Maximum Count, Life Span, Mature Age
B cells	Initial Count, Maximum count, Life Span, Plasma B Life Span, AB_BreakDown
T cells	Initial Count, Maximum Count, Life Span
Display	Display Concentration, Display Gradient
Grid	Rows, Columns, Depth, Minimum Concentration, Maximum Concentration, Diffusion Constant, Breakdown Constant

Table 1 Parameters that can be input through GUI

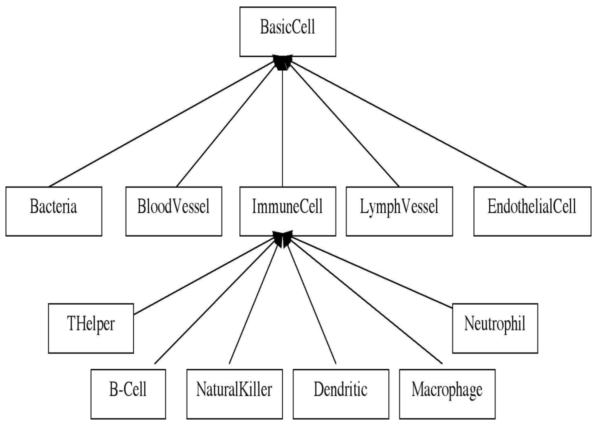

Figure 3 Main C++ Classes Implemented

4.1 The Modeler Entities

A C++ object represents each entity in the model. Some of the C++ objects we use are given in Figure 3. Each classs behavior is described in terms of a deterministic finite automaton or DFA (20). The DFA is expressed in terms of an XML language discussed in Section 5. In the discussions below, we show the DFA for a few of the classes

4.1.1 Class `BasicCell`

Class `BasicCell` is at the top level of the hierarchy tree. All cells inherit its characteristics. Common methods such as `setType()`, `setStatus()`, `setState()` are defined in this class. Methods like `move()`, `setGridWrapper()` and `setLifeSpan()` are also coded here. A few classes at lower levels of hierarchy, modify the method `move()`depending on the specific manners in which they move.

4.1.2 Class `ImmuneCell`

This class at the second level of hierarchy is the parent class of all immune cells. The main methods of interest are `hasBumped()`, `selfNonself()` and `die()`. An immune cell calls the method `hasBumped()` to check whether it has bumped into another cell and it calls `selfNonself()` to check whether another cell is an invader. The method `die()` is called when a cell attains its mature age.

4.1.3 Class `Bacteria`

Currently we have only one type of pathogen: a generic bacterium. The class `Bacteria` exhibits the behavior of bacteria once it enters the body. A bacterium reproduces at a specified age and moves through the tissue, travels with the flow maintained in the grid and finally reaches the lymph node by using the method `moveBacteriaLymph()`. It carries its bit signature (expressed in its epitope), which allows T cells with the complimentary signature to be activated. They also secrete LPS, a chemical which affects the tissue and its cells.

4.1.4 Class `Macrophage` and Class `DendriticCell`

The class `Macrophage` models macrophages that are prevalent in the tissue in the beginning of the simulation. A fixed number of these move around at random in the tissue and exhibit their garbage collector behavior of eating any dead or foreign entity. A macrophage moving around in the tissue checks if it has found an entity in its vicinity by the method `hasBumped()`. It further checks if the entity is foreign by the method `selfNonself()` inherited from the `ImmuneCell` class. The DFA given in Figure 4 depicts its behavior.

Before presenting the antigen to T cells, a macrophage processes the ingested antigen using the `chopAntigen()` method implemented in the class `Cytoplasm`. Simply encountering an antigen does not make a macrophage an APC. A cytokine signal such as IFNγ secreted mainly by T-Helper Typ1 (or Th1) NK cells may be needed to become an APC. This is incorporated by loading this chemical onto the grid using the method `loadChem()` in the class `Grid`. The hyper-activated state of a macrophage is implemented by loading the LPS secreted by bacteria and checking its concentration in the vicinity of an activated macrophage using the method `getConcentration()` of `Grid` class. Finally, the method `move()` lets macrophages move around in the grid unless they bump into another entity and undergo processing.

Another category of phagocytes is the dendritic cells implemented as class `Dendritic`. Dendritic cells have been implemented to move to the site of infection and eat the bacteria. Normally the macrophages present in the grid perform

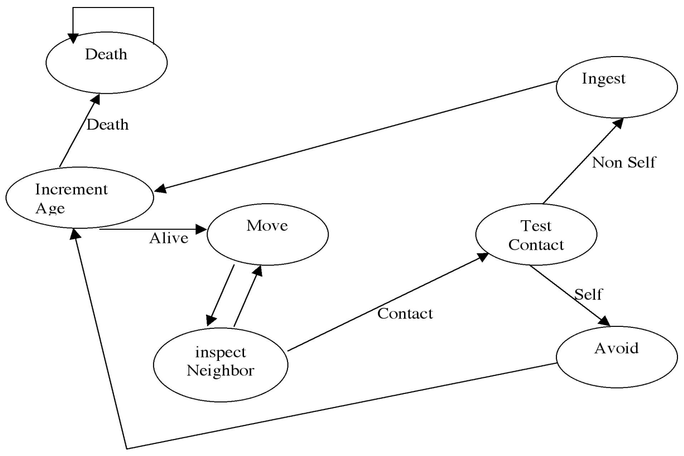

Figure 4 DFA for Macrophage

this function but in a more serious situation, the dendritics are called to the site of infection. When activated macrophages secrete TNF, dendritic cells are activated and they return to the lymph node. On the way they increase MHC expression and the quantity of B7 family of co-stimulators.

4.1.5 Class `Neutrophil`

Neutrophils have been implemented as immune fighters with a very low life span but which arrive in large quantity. Instances of the class `Neutrophil` move out of the damaged area on sensing the concentration of the inflammatory cytokines. The software simulates a scenario where neutrophils are not called unless the battle is intense. This is followed by the macrophages and dendritic cells removing all the bacteria. It is only when the macrophages are activated by the IFNγ secreted by a large number of cells that they in turn secrete cytokines to signal to the resting state neutrophils in the blood vessels to go to the site. This is an important detail because in normal circumstances a small number of antigens are easily cleared off by the phagocytic cells.

4.1.6 Class `NaturalKiller`

Before the adaptive immune fighters are called to the battle site, there is another category of immune cells that play an effective role in killing antigens. These are natural killer cells implemented as the class `NaturalKiller`. The natural killers are implemented as coming out of blood vessels on sensing LPS secreted by the bacteria. When the bacteria are killed, the IFNγ released by them are loaded onto the grid. This activates the macrophages and as a result more secretion of TNF and IL1 occurs. These two chemicals are diffused through the grid by `loadChem()`. This in turn accelerates NK cells to produce even more IFNγ; this is how the innate fighters accelerate one another's actions. The methods of interest are also mainly those that it inherits from `BasicCell`. It uses the method `hasBumped()` to check the type of the cell into which it has bumped, discriminates it on the basis of `selfNonself()` method of `ImmuneCell` class and kills it using `kill()`. They move in the tissue in a random fashion since they are long lived and have a large capacity to kill the invaders. The method `moveNKrandom()` guides their movement in the tissue

4.1.7 Class `ComplementSystem`

The complement proteins are in the tissue and blood as the simulation starts. They are spread all over the grid in a random manner with no gradient basis. Complement proteins act as a part of innate immunity during the initial phase of an infection. Their main function is to punch holes into the cell walls of antigens. But before they can make holes using the Membrane Attack Complex (MAC), they need to be activated either by the *alternative pathway* or by the *lectin activation pathway*. Whatever pathway activates them, the initial spark comes from the C3 convertase formed by the cleaving of the protein C3. This results in further cleaving of some more neighboring C3 proteins and ultimately the chain of C5, C6, C7 and C8 is produced. The end product of this chain is the formation of MAC. In our simulation, opsonization is implemented so that complement proteins coat the pathogens; this eases their phagocytosis. Once a bacterium tagged by antibodies produced by the activated B cells finds the complement proteins in its vicinity, it succumbs to the MAC created by these proteins. This completes the link from the adaptive fighters back to the innate fighters in assisting them in the killing of pathogens.

One method of interest in the `ComplementSystem` class is `inspectForNeighbors()` whereby once the bacteria find complement proteins in their vicinity, the C3 protein is cleaved to form C3a and C3b. The C3b attaches itself to the bacteria to further cleave more C3 and C5 so that they can proceed to form the MAC. In our simulation, the antibody IgM is able to activate five C1 protein complexes at a time. These activated complexes can initiate a cascade of events that produce a C3 convertase. On the other hand, in our software, antibody IgG can activate only one C3 convertase at a time. At times other than when they are not in the vicinity of any bacteria, the complement proteins move in the grid by `moveComplement()`. They are very unstable so their number is maintained constant by continuously creating more of them.

4.1.8 Class `THelper`

T cells are responsible for activating B cells after they themselves get activated on bumping into an APC. The `complement()` method finds the complement string of the signature of the antigen. String matching is used to activate T cells if a right match of the epitope of bacteria (a string of characters) and the signature of the T cell is found. The activated T cells carry the signature of the processed antigen from the membrane of the APC. This signature is needed to prime B cells which can create antibodies specific for an antigen. Cytokine signal IL1 released by activated phagocytes is needed for T cells to be activated. Chemicals loaded at one position in the grid are spread throughout the grid by the `diffuseChemicals()` method. A concentration gradient is maintained in the grid depending on the flow of body fluids in the grid. T cells follow this gradient. Once activated, T cells follow the flow of body fluids and travel to the nearest lymph vessel. Currently, our software

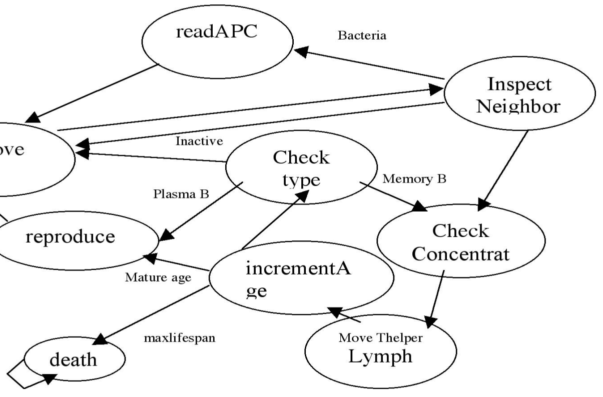

Figure 5 DFA for T Cell

supports T cell generation and activation, but assumes a generic T cell whose role is integral to the generation of antibodies. T cells are activated by APCs. But it must meet its cognate antigen peptide for activation. This feature is implemented by allocating a unique signature to every T cell. This is stored in the T cell receptor (`tcr`, also called the signature) field of the T cell. When a T helper cell bumps into a bacteria, it uses the `find()` method to find the `tcr` complement in the bacteria membrane. If the match is found, the T cell is activated and it can activate the matching B cell to produce the antibodies.

4.1.9 Class `Bcell`

B cells created in the bone marrow are released into the grid through the blood vessels simulated in our model. Our software model has antigen specific B Cells generated by a random bit generator. The process of binding to specific parts of the antigen presented by the APCs is simulated in the `readBact()` method. Only B cells that find the complement of their receptors (`bcr` or signature) in the signature of the bacterial membrane are primed by setting the `prime` flag to be true. When a B cell meets an activated T cell, `readTcell()` compares its `bcr` with that of the T cell using the `find()` method. This renders B cells activated and the `active` flag is set to true. The DFA for Bcells is given in Figure 6.

The primed B cells follow the flow set by the blood vessel and travel to the lymph node where they perform *clonal selection.* Only the B cells that are primed and active form clones by using the method `reproduce()`. In the human immune system, each B cell can produce only one kind of antibody. Once the primed B cell knows the signature of the bacteria, the activated T cell provides it the required growth factor IL2. The primed and activated B cell proliferates to form a clone of B cells with the same `bcr`. Finally, we have enough B cells to produce antibodies against the bacteria with a particular signature.

The next stage is the career decision, whether to be a memory B cell or plasma B cell. A few of the reproduced B cells are created to be memory B cells to fight against the invasion by the same bacteria at later stages in the life cycle of the host; others take the function of plasma B cells. In the current simulation, this categorization is on the basis of a randomly generated bit. The plasma B cell takes

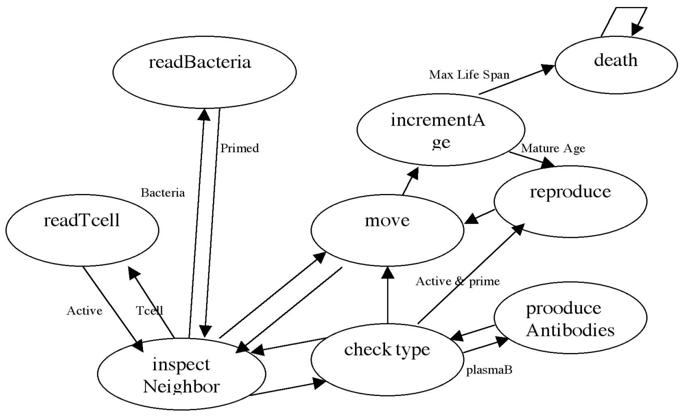

Figure 6 DFA for B Cell

the job of producing antibodies. Going through the list of B cells, a plasma B cell places an antibody in its grid position. Currently the model assumes bacteria of only one type, so only one type of antibody is released. We also have implemented the process of repeated mitosis and generation of plasma cells secreting a specific receptor or antibody into the tissue.

4.1.10 Class `Antibody`

Two kinds of antibodies are created in the present simulation in SIMISYS: IgM and IgG. IgM is created in the initial phase of the simulation and IgG toward the end. An antibody of the type IgM is capable of activating five of the complement convertase complexes. This in turn can deactivate five bacteria at a time so that in the beginning of the simulation antibodies of only the type IgM are created. Towards the end when usually only a few bacteria are left in the system, IgG can very well do the job by activating just one convertase molecule at a time. In the current simulation, this is implemented by creating IgM or IgG based on the total count of bacteria in the simulation. Antibodies are produced in the lymph node by the method `produceAntibodies()` and are translocated to the tissue by the blood vessels by the method `searchForAntibody()`. We have only one lymph node in our current simulation and it is situated behind the tissue in the grid and in the display. There is a lymph vessel inside the lymph node. The antibodies find bacteria by using the method `inspectForNeighbor()` where they look for entities in their current position by `isOccupied()` method, check its type by `getType()` and if it is a bacteria, the antibody status is set to `dissolve` and it attaches itself to this bacteria rendering its status to `disabled`.

4.1.11 Class `Cytoplasm`

Every cell, whether basic or immune, has cytoplasm in it where all metabolic activities vital for the survival of the cell take place. The class `Cytoplasm` has been introduced mainly to process the ingested antigen and present it to the T cells. In addition, every Antigen Presenting Cell (APC) has MHC class II (Major Histocompatibility Complex) molecules in its cytoplasm. This molecule has a groove on its surface where it holds the chopped peptides of the ingested antigen. A variable `mhc` in the cytoplasm holds the groove in our simulated cells. The groove has markings in the form of a string of 0s and 1s expressed in one byte. When the macrophage or the dendritic cell ingests the bacteria, it calls the method `loadMHC()` wherein the `chopAntigen()` method creates strings of peptides, again 0s and 1s expressed in one byte.

4.1.12 Classes `LymphVessel`, `LymphNode` and `BloodVessel`

There are two different "worlds" in our simulation: one is the infected tissue and the other is the lymph node. The immune cells and bacteria move around in the grid whether in the tissue or in the lymph at random. But when the number of bacteria is large, the bacteria as well as the APCs and immune fighters, specifically the adaptive fighters move to the lymph node where they can easily get hold of the bacteria and create antibodies to kill them with specificity. To achieve this, the movement of the cells from one grid world to another (tissue to node and vice versa) has been accomplished through blood vessels and lymph vessels.

There are two active `BloodVesssel`s in the grid which send new entities to the grid after they sense the chemicals around them by the `inspectForChemicals()`. This method checks for the concentration of the chemicals LPS, INF and IFNy around its walls and on detecting their presence allows the entry of instances of class `Neutrophil`, `NaturalKiller` and `Thelper`. The blood vessels also suck in the activated immune fighters: `Bcells`, `Tcells` and `Bacteria` and send them to `LymphVessel` after they sense their presence using `inspectForNeighbors()`. This method checks for the type of these entities and places these entities in the grid inside the `LymphNode` created at the back of the main screen of the simulation using the `placeEntity()` method. The `LymphVessel`, located in the lymph node, looks for entities like antibodies and translocates them back to the tissue so that they can pinpoint the intruders and make the job of phagocytes and complement proteins easy. This is how a continuous flow of entities is maintained.

4.2 The Matrix

The Matrix represents the physical space we simulate. It is implemented using the `Grid` and `Gridwrapper` classes. The implementation of cellular automata approach to modeling and simulation requires that we have a physical space composed of *physical cells*. The `Grid` implements the physical cells. Sometimes we just call it the *grid*. Inside each physical cell in the grid, we can place one *biological cell* and one or more identified molecules.

4.2.1 Class `Grid`

The basis of the simulation rests with the `Grid` structure. The `Grid` is composed of an array defined in 3D making up the world of simulation. Each grid cell or box can contain a pointer to an entity and also store information about the current conditions in that cell. Currently we have implemented a grid with 100 rows, and 100 columns. The depth is 20. This allows us to simulate up to 100 x 100 x 20 = 200,000 cells. This is a small number compared to the many billions of players in the immune system, but this is still a very high number for computer simulation. The `Grid` forms the main section of the tissue where all immune cells move around and interact with each other. Each grid cell maintains a concentration list of all chemicals in it. Currently we take into account six chemicals that stimulate the cells and maintain the concentration gradients required for the movement of the cells. These are:

- LPS (lipopolysaccharide)
- IFN (Interferon)
- IL-1 (Interlukin-1)
- IL-2 (Interlukin-2)
- TNF (Tumor Necrosis Factor)
- IFNγ (Interferon-gamma)

There are other chemicals that we do not model; these include IL-4 for example. Three blood vessels and one lymph vessel are stationed in the main grid as well. The lymph node present inside the lymph vessel encloses another small grid in itself. The smaller grid has a size of 30 x 30 x 20 or 18,000 physical cells. A *grid pointer* is used in each entity to point to the grid position in its world. Using the grid pointer, a cell can check its neighboring grid positions for other entities, or inquire about the chemicals, complement proteins and antibodies present within its own position. A *back end pointer*, present in each grid cell points to the cell that is in it currently. The back end pointer reveals the identity of the immune cells or the bacteria present in the neighboring positions. Further, it also helps the visualization module to get the identity of cells at the positions that it is displaying.

There are several methods of this class. For example, `isOccupied()` lets an entity to know whether the neighboring position is occupied. When we say the neighboring position, we mean anyone of the neighboring 27 positions since it is a 3-dimensional grid. The methods `setOccupied()` updates the status of each of the grid cells as it is occupied or emptied out. The method `loadChem()` allows for the loading of a specific concentration of a chemical and the method `getConcentration()` allows the access to the concentration of an already loaded chemical onto a grid location.

4.2.2 Class `GridWrapper`

The class `GridWrapper`, as the name suggests, encapsulates a 3D rectangle of grid cells. It allows us to contain information about a grid along with the memory allocated to it. The `GridWrapper` class is essential in cases where more than one grid or world is simulated such as a section of infected tissue and a lymph node as we do currently. Each of this is implemented as a separate grid. We have to inform each entity that we create about the world that it belongs to. Hence each entity structure contains a pointer to a `GridWrapper` class object through which it can access the appropriate data in the world to which it belongs.

4.2.3 The Visualization Engine

We have developed a tightly integrated visualization engine that is easy to set up and can be adapted to handle introduction of entities in future releases. We also use a graphing package to illustrate the results of the simulation. The engine is based on the use of the SDL library(24), an open source package suitable for direct screen manipulation. The advantages of using the SDL package are the ability to *blit*, or paste an image on the screen at a specified location. The SDL library can be used for normal 3D operations, but the direct blitting of images facilitates rapid prototyping. The engine operates described below.

1. The user specifies the images that are to be used for each type of entity present in the simulation. The engine ignores display of any entity that has not been specified.
2. The engine formats the image loaded for transparent background color and performs scaling of each of the entities to have a series of increasingly larger images.
3. Based on the user's key presses, the engine decides the area of the simulation to be displayed. Currently six possible directions of movement, two on each of the three dimensional axes have been implemented. This allows the user to zoom in and out of the screen as well as to move up and down the screen.
4. The engine scans the specified section of the grid.
5. For each of the entities recognized in the grid, the engine computes the distance of the entity from the front of the screen. Based on this distance, the image is blitted on screen such that a smaller image is pasted for an entity further than an entity closer to the user. This gives the impression of a 3D engine, without the computational expense.

4.3 Simulation

The objective of SIMISYS is to simulate various normal and infection scenarios. Simulating scenarios helps us understand how the players in the immune system interact. Our long term goal is to develop a platform that allows simulation of many different infection and disease scenarios in great detail. At this time, the simulation is fairly simple. The algorithm used for the simulation is given below.

```
Read the input file and get counts for each entity
Create linked lists of each entity

For each timestamp do
         Select one of the linked lists randomly
         For each entity in the linked list do
                   Run the live method of the entity
                   If the entity is dead, remove it
                                 from the linked list
         End do
         If there are chemicals to be
```

```
                                spread in the grid,
                 spread them to neighboring cells
        Gather information about
                 the currently viewed portion of the grid
        Pass the currently viewed
                 portion to the display module
        Provide visual display
End do
```

The main driver reads an input file and creates a list of the entities in the simulation. The multithreaded architecture lets one of the linked lists to be selected at random and for each entity of the system, the `live()` method from its corresponding class is executed. The `live()` method of a class decides the status of the entity at the end of a simulation cycle. The entities which are dead at the end of a simulation cycle are removed from the list and new entities are added if they are created. The dynamics of the system are controlled partly by chemicals and partly by the changes in the life cycle of the entities. One after the other, depending upon which thread of computation gets the control, each linked list of entities goes through the simulation cycle. The display engine also gets the control on a regular basis. It gathers information about the currently processed portion of the grid, updates the data and passes the information to the display module which provides the visual display. Figure 7 gives a flow chart depicting the processes that are modeled and simulated.

5 Use of XML in SIMISYS

It was recognized during the creation of SIMISYS that developing a robust way of obtaining, storing and using information is an essential requirement. XML is a standard format for handling such information(26). The hierarchical document format is intuitive and easy to handle. Open-source parsers are available to create and extract the information in the XML format. We use the Iksemel parser from the Jabber project available from `http://ikesemel.jabberstudio.org`.

In SIMISYS, the configuration files for setting up the parameters that control the simulation are written in XML. In addition, the attributes and behavior of each entity are encoded in the XML format. For example, Figure 11 shows the behavior details of `macrophage.xml`. The contents of this file is based on the DFA for class Macrophage given in Figure 5.

One of the design goals while incorporating XML into SIMISYS is that we want to describe an entity's behavior, parse it, interpret it, and create code from it. We have developed an XML language that can facilitate this process. The features of the language are given below.

- Supports the following data types: int, double, string and boolean.
- Supports mathematical evaluations such as $+$, $-$, $*$, $/$ and ^ in expressions.
- Supports strict data type checking.
- Supports the common conditional statements.

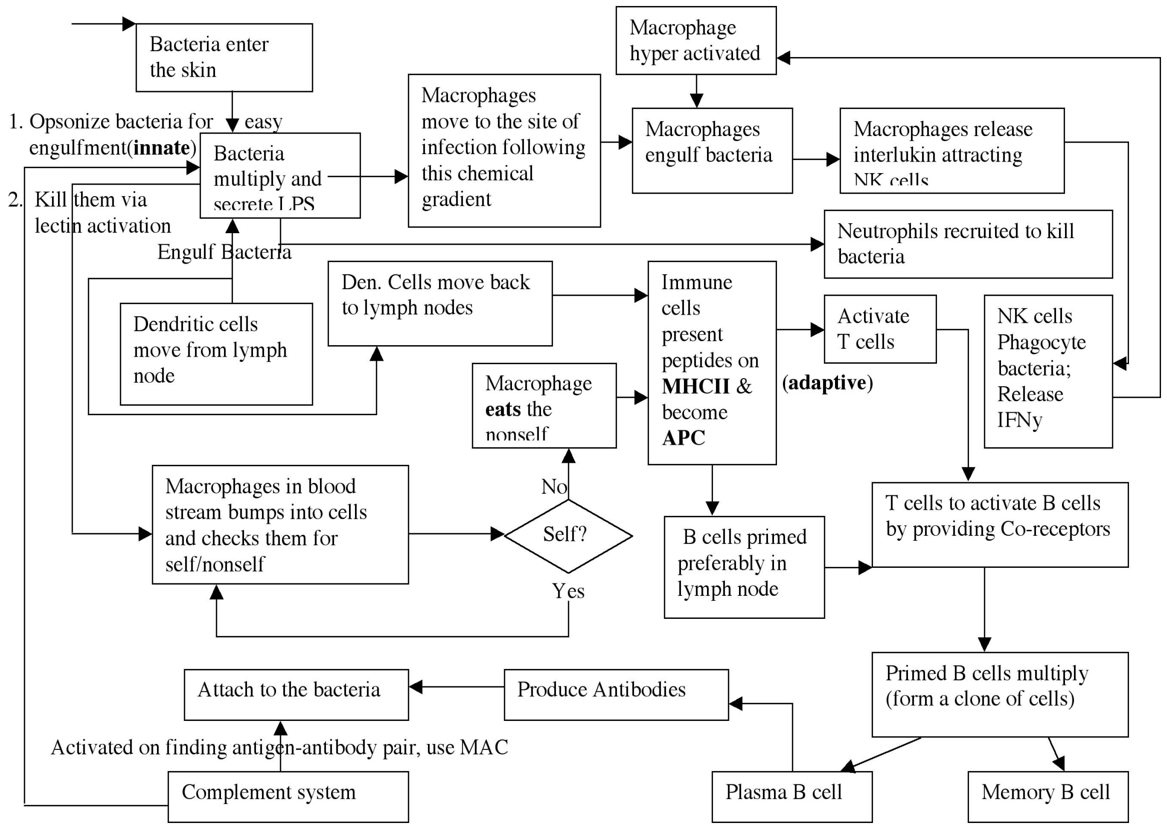
Bacteria enter the skin
1. Opsonize bacteria for easy engulfment(**innate**)
2. Kill them via lectin activation
Bacteria multiply and secrete LPS
Engulf Bacteria
Macrophages move to the site of infection following this chemical gradient
Macrophage hyper activated
Macrophages engulf bacteria
Macrophages release interlukin attracting NK cells
Neutrophils recruited to kill bacteria
Dendritic cells move from lymph node
Den. Cells move back to lymph nodes
Immune cells present peptides on **MHCII** & become **APC**
Activate T cells
(**adaptive**)
NK cells Phagocyte bacteria; Release IFNy
Macrophage **eats** the nonself
No
Self?
Yes
Macrophages in blood stream bumps into cells and checks them for self/nonself
B cells primed preferably in lymph node
T cells to activate B cells by providing Co-receptors
Primed B cells multiply (form a clone of cells)
Attach to the bacteria
Produce Antibodies
Activated on finding antigen-antibody pair, use MAC
Complement system
Plasma B cell
Memory B cell

Figure 7 Flow Diagram showing the role of innate and adaptive immune fighters. The arrows show the flow of control and the boxes highlight actions.

```
<entity type="macrophage">
   <properties>
         <life-span>1000</life-span>
         <start-age>0</start-age>
         <membrane>MHCMACROPHAGE</membrane>
  </properties>
  <behavior start-BU="incrementAge">
         <BU name="incrementAge">
     <event name="alive" reaction="move"/>
     <event name="dead" reaction="death"/>
         </BU>
         <BU name="move">
      <event name="" reaction="inspectNeighborhood"/>
         </BU>
         <BU name="death">
      <event name="" reaction="death"/>
         </BU>
         <BU name="inspectNeighborhood">
     <event name="contact" reaction="testContact"/>
     <event name="" reaction="incrementAge"/>
         </BU>
         <BU name="testContact">
      <event name="self" reaction="incrementAge"/>
      <event name="non-self" reaction="ingest"/>
          </BU>
          <BU name="ingest">
       <event name="" reaction="incrementAge"/>
          </BU>
  </behavior>
</entity>
```

Figure 8 XML File for Macrophages

- Supports indefinitely nested conditionals.
- Supports the ability to call C functions from XML.
- Supports the writing of DFAs in XML.

An interpreter has been written in order to handle this specification. The system works as given below.

- Each entity is represented by the same class called `Entity`.
- From the configuration file, each instance of the class reads the location of the XML file representing its behavior and loads it.
- Based on the DFA present in each XML file, the behavior of the entity is determined. The XML file for macrophages is shown in Figure 8. The interpreter determines the state of each entity depending upon the state in which the entity is currently in.
- For each state in the DFA, a code block is written that determines the behavior of the entity while in that state.

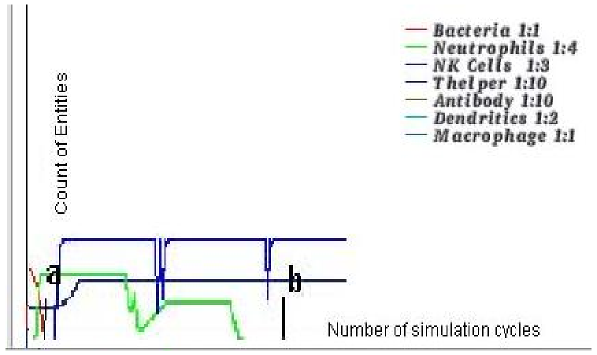

Figure 9 Result Graphs for Scenario 1

- Transitions from one state to another are made while interpreting the code block.
- There is a facility to call C++ functions in order to explore the environment of the entity such as other cells in the neighborhood or the presence of chemicals. Such stimuli can be used in conditionals in order to modify the flow of the code.

6 Results and Analysis

We discuss four scenarios modeled and simulated using SIMISYS 0.3. They explain the immune response to a simple infection. The numbers displayed in the result graphs of the run of the simulation are the ratios of the number of the entities. On the X-axis, we indicate the number of iterations of the simulation. The change in the number of the associated entities is on the Y axis.

Situation 1 assumes that a person is infected with a generic bacterium. For example, the individual may have been exposed to bacteria from some other person sneezing. Here we assume a minor infection. The bacteria, in such a case, travel through the nasal path and activate the nearest lymph node situated in the throat. We see the bacteria number quite high as the simulation starts at the origin. As simulation progresses, we see the number of neutrophils growing in number between **0** and the point **a**. NK cells also participate in the fight but may be due to the scale used in the graph, they do not show up earlier. Between points **a** and **b**, we see that the bacteria number does not rise and also the number of neutrophils fades away. This agrees with real life where a minor infection is usually taken care by the neutrophils.

In Scenario 2, we assume that the person is infected with bacteria and the infection remains local. The result in this case is different from the previous case at point **b**. Here also the neutrophils and the NK cells are the first ones to reach the battle site. Between points **0** and **a**, the bacteria number stays constant for some time and then it starts increasing. It seems that at point **a** that the bacteria are all destroyed. But at point **b**, the appearance of macrophages indicates that the bacteria are still there and are being eaten by macrophages. The macrophage

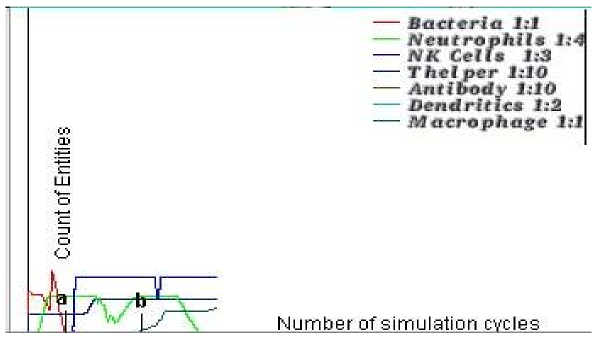

Figure 10 Result Graphs for Scenario 2

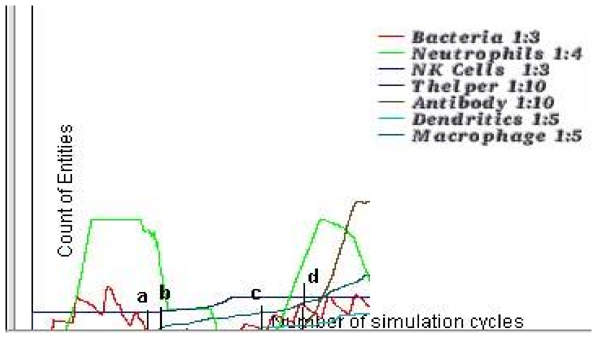

Figure 11 Result Graphs for Scenario 3

graph here represents the number of macrophages that have engulfed the bacteria. Finally after a few more runs of the simulation we see that the bacteria are totally gone. This result also matches favorably with a real life situation whereby the macrophages become activated by the secretions of an invader and help the innate fighters clean up the infection by eating them.

Scenario 3 is a case of real bad infection where the neutrophils alone are not able to kill the bacteria.. The macrophages enter the battle site at point **b**. At point **c**, even the dendritics, which join the innate fighters only when the infection is really out of control, also appear at the battle scene. But we still see the number of bacteria rising up between points **c** and **d**. At point **d** we see the activated B cells also are releasing the antibodies. Even if their number is so high, the bacteria number is still uncontrolled. It takes some time for the antibodies to tag the bacteria and make their killing by phagocytes easier. The results of the simulation comply very well with the real life situations.

Situation 4 is a case of bad infection where the neutrophils alone are not able to kill the bacteria. The macrophages enter the battle site at point **b**. At point **c**, the dendritics, which join the innate fighters only when the infection is out of control, also appear at the battle scene. But we still see the number of bacteria rise between point **c** and **d**. At point **d** we see the activated B cells also release antibodies. Even if their number is high, the bacteria number is still uncontrolled.

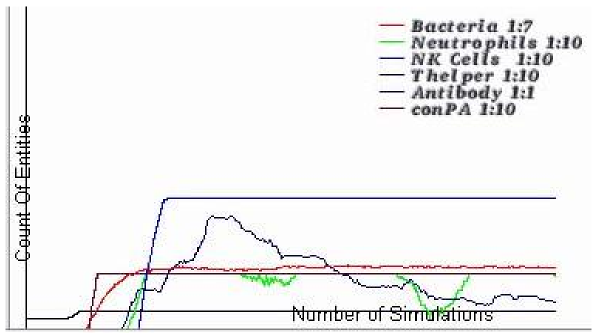

Figure 12 Result Graphs for Scenario 4

It takes some time for the antibodies to tag the bacteria and make their killing by phagocytes easier. The results of the simulation comply very well with real life even if the classes implement very simple methods as of now.

7 Simple Extensions for ANTHRAX Simulation

We have extended SIMISYS 0.3 to illustrate infection by the anthrax bacteria *Bacillus anthracis* (7; 12; 27). We achieve this by simulating the following.

- The behavior of spore form of bacteria in the body,
- The release of toxins by the bacteria and the damage caused, and
- The action of the known antibiotics for anthrax.

Figure 13 describes the flow of events during the anthrax simulation. The anthrax bacteria invade the human body in the form of spore. When a spore is eaten by an immune cell such as macrophage or neutrophil, the spore finds suitable conditions to germinate and multiplies. The resulting bacteria are carried to the lymph node and spread into the blood stream. The bacteria continue to release the toxins: Protective antigen (PA), Edema factor (EF) and the Lethal factor (LF). Edema toxin increases cAMP that upsets water homeostasis and causes edema. It also impairs neutrophil function. Lethal toxin stimulates the macrophages to release TNFa and IL1b that are responsible for shock and death.

The main methods which have been modified for the simplistic modeling and simulation of anthrax are those of the classes: `BasicCell`, `Bacteria` and `Macrophage`. They are briefly explained below:

7.1 Extensions to the `Bacteria` *Class*

For this simulation, the bacteria are initially placed at the site of infection in *spore* form. As long as these bacteria are in spore form their age is not incremented; they cannot release toxins or reproduce but only move. When a bacterium is eaten by an immune cell such as a macrophage or a neutrophil, its state is changed to *vegetative*. In this state the bacteria release toxins, reproduce and age in every

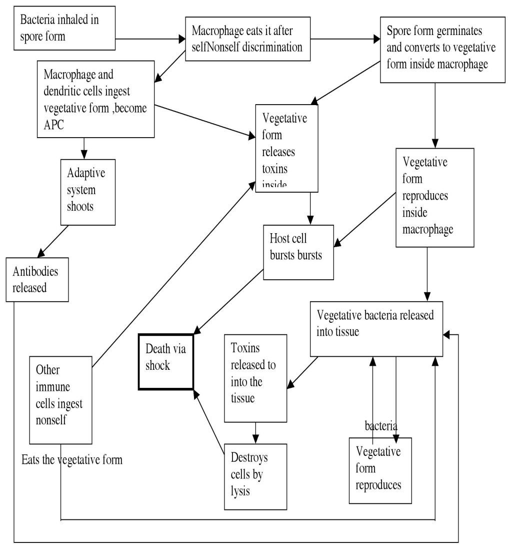
Bacteria inhaled in spore form
Macrophage eats it after selfNonself discrimination
Spore form germinates and converts to vegetative form inside macrophage
Macrophage and dendritic cells ingest vegetative form ,become APC
Vegetative form releases toxins inside
Vegetative form reproduces inside macrophage
Adaptive system shoots
Host cell bursts bursts
Antibodies released
Vegetative bacteria released into tissue
Death via shock
Toxins released to into the tissue
Other immune cells ingest nonself
bacteria
Eats the vegetative form
Destroys cells by lysis
Vegetative form reproduces

Figure 13 High-level Flow Diagram for anthrax simulation

simulation cycle. Based on the bacteria's position (inside an immune cell or in the grid), the reproduced bacteria and the toxins are released inside the immune cell or the grid. The bacteria stop reproducing and releasing toxins after reaching a certain age and eventually die.

The `eat()` method no longer kills the bacteria when a macrophage ingests the spore form of anthrax. Instead the `releaseToxins()` method of the `Bacteria` class allows the release of toxins into the cytoplasm of the macrophage. When the maximum fluid parameter associated with the macrophage reaches its upper limit, the macrophage bursts, i.e., dies and the chemicals EF, LF and PA and also the vegetative form of bacteria are released into the grid. The `diffuseChemical()` method (that runs as a thread in `main`) diffuses these chemicals into the grid and on reaching a threshold value, any other cell that comes in contact with these toxins dies due to cell lysis. A method `findToxins()` checks for the presence of toxins within a cell and also in its grid position. If edema toxin is found, the macrophages are set to produce increased amounts of cAMP and IL6. If lethal toxin is present the macrophages produce more IL1 and releases less TNFa. The IL1 accumulates inside the macrophage using `addChem()`.

7.2 Extensions to the `BasicCell` *Class*

Since all cells inherit from the `BasicCell` class, the effect of the bacterial toxins was added to this class. If edema toxin is found to be present, the Macrophages are set to produce increased amounts of cAMP and IL6. If lethal toxin is present the macrophages produce more IL1 and releases less TNFa. The presence of these toxins also sets the neutrophils to become impaired and hence the neutrophils' ability to phagocytose is disabled.

7.3 Extensions to the `Macrophage` *and* `Neutrophil` *Classes*

Since the Anthrax bacteria primarily affects macrophages, harmful effects are added to the `Macrophage` class. A macrophage moves around and eats a bacterium whether spore or vegetative. If the bacterium multiplies within the macrophage, the reproduced bacteria are maintained in a vector with the macrophage. The toxins released by the bacteria are also held within the macrophage. When the macrophages capacity exceeds either due to the multiplication of bacteria or the toxins released, the macrophage bursts and the bacteria and the toxins within the macrophage are released to the grid cell in which the macrophage is currently present. The macrophage status is set to `dead`. The released toxins add to the total toxin levels shown in the graph and contribute to the change in the behavior of the macrophages and other cells. The toxins released when the macrophage bursts are diffused to the surrounding locations.

A normal neutrophil eats bacteria, whether spore or vegetative. The bacteria multiplies and releases toxins within the neutrophil until its bursts. When a neutrophil moves to a grid cell that has EF and PA, the edema toxins get into the neutrophil. This also increases the cAMP within the neutrophil and causes it to become impaired. This impaired neutrophil does not function normally.

7.4 Antibiotic

We model the effect of a generic antibiotic in a very simplistic manner. The antibiotic `cipro` is injected at a location within a blood vessel. `Cipro` like all the other chemicals in the simulation is diffused through the tissue using the method `diffuseChemicals()`. The `live()` method of the `Bacteria` class has been modified to look for the `cipro` chemical in its vicinity and the bacteria are rendered dead if they sense `cipro` in their surroundings. The bacteria check the concentration of `cipro` in its vicinity using `getConcentration()`. Antibiotics have been found to be effective if administered during the initial stages of an infection. If the bacteria have already started germinating and releasing their harmful toxins, the antibiotic only kills the bacteria but the toxins continue to affect the cells. The antibiotic coats the bacterial cell wall, impairs it and eventually kills the bacteria. Also if before contact with `cipro`, the toxin level reaches a threshold, the body succumbs to it. This is what our graphs show. By the time the bacteria come in contact with the `cipro` chemical diffused through the tissue, a good amount of toxins have already been released. Some bacteria are rendered dead due to `cipro`, but the toxins show their harmful effects.

7.5 Observed Results

The behavior of different cells discussed above have been incorporated to the model. An example simulation is shown in Figure 14.

The graph shows increase in the levels of cAMP, IL1 and TNF due to the toxins released by the Anthrax bacteria. For simplicity, only the level of toxin PA (Protective Antigen) has been shown but the edema and lethal toxins are also released in proportional amounts. The edema toxin causes an increase in the level

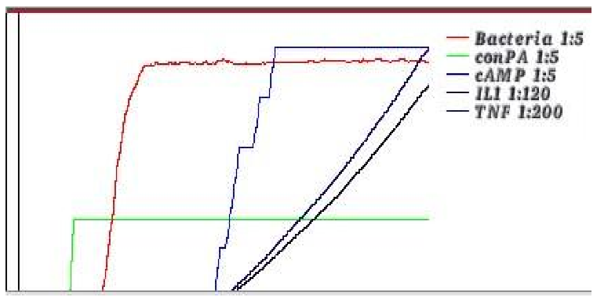

Figure 14 Result Graphs for Simple Anthrax Situation

of cAMP and the lethal factor stimulates the macrophages to release TNF and IL1. Due to these factors the macrophages and the neutrophils are impaired and the host defense is affected.

8 CONCLUSIONS

In summary, SIMISYS 0.3, the latest version of our software has well-integrated modules simulating the innate and adaptive systems. The salient features are:

- A stable grid structure called the Matrix to facilitate positioning of large number of cells in space.
- A hierarchical class structure of cells and other players that closely resembles Nature implemented in C++.
- Each of the large number of cells is implemented as an object with its own characteristics and identity
- A 3-D visualization module that offers scrolling in 6 directions and statistical graphing capabilities using the SDL libraries.
- Models the working of the simulation based on a section of generic tissue connected to a lymph node through lymph vessels.

The current model of the innate immune system simulates the self non-self recognition, garbage collection by macrophages, and the role of complement proteins, and the attraction of neutrophils and NK cells to the region of attack. The adaptive part stimulates the activation of T cells, B cells, production of antibodies, and the final action of complement proteins by MAC to kill bacteria. The model also simulates the diffusion of six chemicals in the grid and their effect on the functioning of the immune cells. We make many simplifying assumptions to obtain a fully-functioning software system. In future versions, we intend to implement further details into all these already implemented classes. We intend to add other immune cells and "organs" such as bone marrow and the liver. We intend to improve the GUI to make it more user friendly. Currently we are working on parallelizing the implementation of SIMISYS on a 32-machine Beowulf cluster so that we can dramatically increase the numbers and details of the cells and chemicals we model and simulate.

The anthrax version of the software currently exhibits the basic features of the infection. We can further explore the details of the infection and the possible ways in which the body can fight against it by adjusting the parameters responsible for damage to the body and the parameters which can help stop the release and the spread of these toxins and study their effect in-silica.

Even in its current form, SIMISYS can be useful in many ways. It can be used as an educational aid. We can modify the parameters and see how the response of the immune system varies in a specific scenario. In its current form, it can also be used as a tool to simulate many of the known diseases including autoimmune diseases. Also it can be modified to model and simulate the effect of known antibiotics for some diseases and trying new ones.

References and Notes

1 M. Bezzi. Modeling evolution and immune system by cellular automata. http://citeseer.nj.nec.com/429312.html, 2000.

2 M. Bezzi, F. Celada, S. Ruffo, and P.E. Seiden. The transition between immune and disease states in a cellular automaton model of clonal immune response. *Physica A*, pages 145–163, 1997.

3 Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Publishing Company, 1998.

4 A.W. Burks. *Cellular Automata*, chapter Von Neumann's Self-Reproducing Automata, pages 3–64. University of Illinois Press, 1970.

5 F. Celada and P. E. Seiden. A computer model of cellular interactions in the immune system. *Immunology Today*, 13:56, 1992.

6 A.K. Dewdney. A cellular universe of debris, droplets, defects and demons. *Scientific American*, 261(2):102–105, August 1989.

7 Terry C. Dixon, Matthew Meselson, Jeanne Guillemin, and Philip C. Hanna. Anthrax. *The New England Journal of Medicine*, 341:815–826, September 1999.

8 Funk G.A., A.D. Barbour, H. Hengartner, and U. Kailinke. Mathematica model of a virus-neutralizing immunoglobulin response. *Journal of Theoretical Biology*, 195(1):41–52, 1998.

9 M. Gardner. Mathematical games on cellular automata, self-reproduction, the garden of eden and the game of life. *Scientific American*, 223(4):112–117, 1970.

10 M. Gardner. Mathematical games: The fantastic combination of john conway's new solitaire game of life. *Scientific American*, 224(2):120–123, 1971.

11 Richard A. Goldsby. *Immunology*. W. H. Freeman & Company, fifth edition, January 2003.

12 C. Guidi-Rontani, M. Weber-Levy, E. Labruyere, and M. Mock. Germination of bacillus anthracis spores within alveolar macrophages. *Molecular Biology*, 31(1):9–17, 1999.

13 D. Kirschner. Dynamics of co-infection with m. tuberculosis and hiv-1. *Theoretical Population Biology*, 55(1):94–109, 1999.

14 Hiroaki Kitano. *Foundations of System Biology*. MIT Press, 2001.

15 P. Klein, J. Sterzl, and J. Dolezal. A mathematical model of b lymphocyte differentiation: control by antigen. *Journal of Mathematical Biology*, 13(1):67–86, 1981.

16 Steven H. Klienstein and Philip E. Seiden. Simulating the immune system. *Computer Simulation*, pages 69–77, July-August 2000.

17 B. Kohler, R. Puzon, P. E. Seiden, and F. Celada. A systematic approach to vaccine complexity using an automaton model of the cellular and humoral immune system. *Vaccine*, 19:862–876, 2000.

18 C.G. Langton. Self-reproduction in cellular automata. *Physica 10D*, pages 135–144, 1984.

19 O. Lefevre, P. E. Seiden, and F. Celada. Insights into rheumatoid factor production using a cellular automaton model of the immune system. *International Journal of Applied Science and Computation*, 3:32–47, 1996.

20 John C. Martin. *Automata: Introduction to languages and the theory of Computation*. McGraw-Hill, third edition, 2002.

21 F.E. McKenzie and W.H. Bossert. The dynamics of plasmodium falciparum blood-stage infection. *Journal of Theoretical Biology*, 188(1):127–1440, September 1997.

22 Melanie Mitchell. *Nonstandard Computation*, chapter Computation in Cellular Automata: A Selected Review, pages 95–140. Weinheim, VCH Vetagsgesellschaft, 1998.

23 Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads Programming: A POSIX Standard for Better Multiprocessing*. O'Reilly, Sebastopol, Caliofnria, 1996.

24 Ernest Pazera and Andre LaMothe. *Focus on SDL*. Premier Press, Portland, Oregon, 2002.

25 Alan S. Perelson and Gerald Weisbuch. Immunology for physicists. *Reviews of Modern Physcis*, 69(4):1219–1267, October 1997.

26 Anderson Richard, Birbeck Mark, and Kay Michael. *Professional XML*. Wrox Press Ltd., Birmingham B276BH, UK, 2000.

27 Abigail A. Salyers and Dixie D. Whitt. *Bacterial Pathogenesis: A Molecular Approach*. American Society for Microbiology, second edition, December 2001.

28 Andreas Schadschneider, Ansgar Kirchner, and Katsuhiro Nishinari. *Lecture Notes in Computer Science 2493, Cellular Automata, 5th International Conference on Cellular Automata for Research and Industry, ACRI 2002, Geneva, Switzerland, October 9-11*, chapter CA Approach to Collective Phenomena in Pedestrian Dynamics, pages 239–248. Springer, 2002.

29 P.E. Seiden and F. Celada. *Some New Directions in Science on Computers*, chapter A Simulation of the Immune System, Experiments "in machina". World Scientific Press, Singapore, 1997.

30 D.S. Stein and G.L. Drusano. Modeling of the change in cd4 lymphocyte counts in patients before and after administration of the human immunodeficiency virus protease inhibitor indinavir. *Antimicrobial Agents and Chemotherapy*, 41(2):449–453, 1997.

31 T. Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modelling physics. *Physica 10D*, pages 117–127, 1984.

32 R.B. Trelease and J. Park. Qualitative process modeling of cell-cell-pathogen interactions in the immune system. *Computer Methods and Programs in Biomedicine*, 51:171–181, 1996.

33 John von Neumann. *Theory of Self-Reproducing Automata*. University of Illionois Press, Champain, Illinois, 1966.

34 Jorg R. Weimar. *Cellular Automata (Fifth International Conference on Cellular Automata for Research and Industry, ACRI), 2002, Lecture Notes in Computer Science 2493*, chapter Cellular Automata Approaches to Enzymatic Reaction Networks, pages 294–303. Springer-Verlag, 2002. http://citeseer.nj.nec.com/537405.html.

35 S. Wolfram. Universality and complexity in cellular automata. *Physica 10D*, 1-35 1984.

36 S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific Press, Singapore, 1986.