ABSTRACT

Title of Thesis: LARGE SCALE AGENT-BASED MODELING: SIMULATING TWITTER USERS

Arjuna Ariyaratne, Master of Science, 2016

Thesis directed by: Professor William Rand
Institute for Systems Research

This thesis details an attempt to conduct a large-scale Agent-based modeling simulation where simulating Twitter users are used as an example. In this thesis, Computational Mechanics is used for developing rules that govern each Agent. This thesis also details the development of an entirely new simulation software capable of simulating a large number of Agents by taking advantage of the parallelism offered by latest computing platforms. Development details of this simulation software, named elixrABM, is available from the concept phase to the testing phase of the simulator.

MODELING AGENT BEHAVIOR THROUGH PAST ACTIONS : SIMULATING TWITTER USERS

by

Arjuna Ariyaratne

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2016

Dedication

To my parents, who always stood behind me.

Acknowledgments

This thesis would not have been possible if not for the guidance and mentoring of my advisor, Bill Rand. His unique insights molded me into the multidisciplinary researcher that I am today.

To my dear friend, David Darmon. Dave, this thesis owes much of its existence to your research. Not only do I appreciate your brilliant and ingenious mind, I truly appreciate the guidance you gave me during this work.

For my family, who raised me, who nurtured me, and who was my bedrock during my life: this work owes much of its existence to you.

Table of Contents

List of Figures

List of Abbreviations

ABM	Agent-Based Modeling
EBM	Equation-Based Modeling
CM	Computational Mechanics
CSSR	Causal State Splitting Reconstruction
PRNG	Pseudo-random number generator

Chapter 1: Introduction

1.1 Overview

Humanity has always tried to explain the behavior of fellow humans. We came up with philosophies and religion to explain and guide our behaviors. In recent years, humanity has tried to explain human behavior through the quantification of human actions. Furthermore, quantification of behavior patterns has long being identified as an enabler of emergency operations during a crisis event [1].

The Information technology revolution has crept into human life in ever increasing amounts. We now carry and utilize a large array of sensors that track our behaviors as we go with our day-to-day life. We express our thoughts, emotions, and fantasies through the ever-present Internet. The nature of the Internet ensures that these expressions are stored for a long time.

We aim to retrieve these expressions, now stored as data, extract them, quantify them, find patterns of behaviors and ultimately build rules that guide us in explaining human behavior.

The Internet, combined with low cost of data storage and processing has ensured data about human behavior is constantly recorded. In fact, an entire science called Data Science has been borne out of the necessity to process this gargantuan amount of data. Data Science deals with understanding and predicting the use of data. To achieve this, data science utilizes knowledge bases from computer science, statistics, linguistics, and physics to efficiently process a large amount of data.

Web sites these days routinely track users every click and selection. Every action performed by the users are tracked and stored for further analysis. This data is commonly termed "Big Data" due to the gargantuan size of the resulting data set. Some websites even allow 3rd party developers access to some of the data collected by the website. This data tend to reflect the human behavior of the underlying user.

For example, take a social media website. A Social media website is a specific kind of website where users interact with another users for information dissemination. People connect, advertise and ultimately behave in social media, prompting similar behaviors to as in a social gathering. The resulting data set is perfectly recorded expressions of human behavior. This data tend to be in the forms of actions, text, pictures, and videos. The digital nature of the data also perfectly time stamps the data.

The large scale of the Big Data requires system level development to obtain any meaning out the data. Collecting data from thousands to billions of users produces its own unique set of problems. Collected data then must be stored in an easily and speedily accessible manner. Stored data must also be retrieved with the end system goals in mind. Then this large set of data must be processed in a computationally efficient manner. After processing, the results must be shown to be stored in a useful and understandable manner.

One who considers the building of a system that processes thousands to millions of users first must define the need to process all of them. In this instant, the simulating system must approximate the real world as close as possible. In the real world, a user is influenced by many other users who each behave in unique ways. As one can understand that no human is alike, no human user behavior is alike as well. This means that any simulation that attempts to replicate the behavior of community of digital users, must consider all behavior shown by all users. This is commonly called the property of heterogeneity, where all users have uniform characteristics [2].

One of the most significant architectural decisions taken during the development of the system is the selection of the appropriate user characterizing algorithm, the algorithm that would describe each user into a set of rules. The chosen algorithm not only affects the results of the system, but it also determines the components of the system.

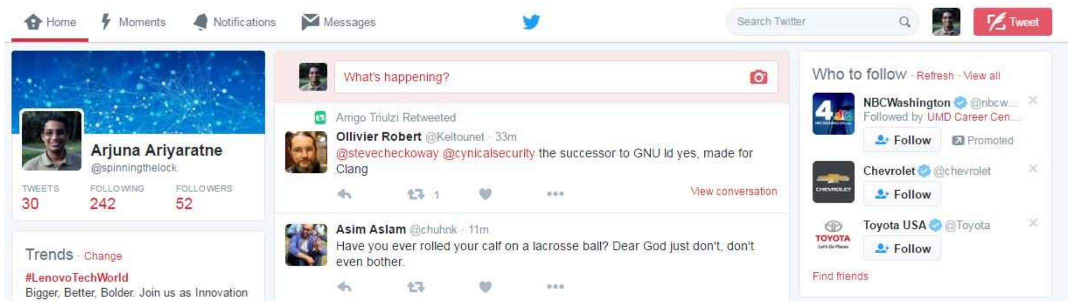
This thesis develops a system that considers all these factors into account. This thesis is a case study of developing an application that can ultimately predict the behavior of people as they behave in social media as a group.

1.2 Problem

Humans take their behavior patterns to any social engagement they perform. These patterns determine some of our behaviors subconsciously. If researchers have a platform to place these behaviors under a microscope, they can exploit these behaviors for the betterment of humanity. As a stepping stone for this grand challenge, this thesis puts the users of Twitter social media platform under the microscope.

1.2.1 Twitter

Twitter is a social media website/platform founded in 2006. The primary concept of operation behind Twitter is to provide the user with a constant stream of messages, limited to 140 characters, produced by other Twitter users who have been chosen by the user to follow. As such, the Twitter platform is a variant of broadcast platforms. This broadcast nature of Twitter has propelled the platform to be one of the primary modes of announcing and communicating through the internet. From celebrities to politicians to family friends, people from many arenas use Twitter. This larger user-base, combined with the relative ease of obtaining data has made the Twitter platform attractive proposition for research.



Arrigo Triulzi Retweeted

Ollivier Robert @Keltounet · 33m
@stevecheckoway @cynicalsecurity the successor to GNU ld yes, made for Clang

Asim Aslam @chuhnk · 11m
Have you ever rolled your calf on a lacrosse ball? Dear God just don't, don't even bother.

Figure 1.1: Twitter Home Page

The Twitter universe has its unique terminology associated with it. In Twitter, subscribing to an another user is called *following*. Once a user has followed they become a *follower* of that user. In Figure 1.1, under the name of the user, Twitter lists the number of followers the user has, in simple terms this is the number of users the message sent by this author would be received.

In Twitter, the message sent is termed as a *tweet*. This message has a maximum limit of 140 characters imposed by the platform. A Message can include images and moving images (GIF formatted files). Many users use a URL (Uniform Resource Locator) shortener to link websites or YouTube videos and stay within the character limit. As a social media platform, tweets have the capability to identify other users within the tweet. This act is called *mentions*. Mentions take the form the "@" sign followed by the username of the user that tweet is mentioning. These mentions can be seen on Figure 1.1.

The Twitter follower-followed paradigm results in a network flow relationship. We will utilize these network flow characteristics in this research.

1.2.2 Human Behavior in Social Media

In preparation for this analysis of the human behavior in social media, we have accumulated a large data set of human behaviors. The data is from the social media website Twitter and records the actions of 15,000 selected Twitter users over a period of one year. Data was collected through TwEater [3].

TwEater is an application created by Center for Complexity In Business at University of Maryland, for utilizing the Twitter Application Programming Interface (API) for the purpose of recording Twitter user behavior. It collects actions performed by the user and stores them in a database.

The problem that this thesis attempts to solve is to develop a system that can take this data and convert them into meaningful results where the system predicts the actions of the 15,000 users as a group in the future. To achieve this, the system development is carried out in several stages that address the three main problems faced by this kind of systems.

The first problem was the selection of the machine learning algorithm that would characterize each user. The system is then developed around the selected algorithm so that the full system works efficiently together. Selected algorithm must be able to take processed data and produce predictions. Input and Output characteristics of the algorithm ultimately determine the rest of the system.

Another problem is the pre-processing of the data. As with most data originated from social media websites, original raw data are in arbitrary formats. In this instant the data format is determined by the Twitter API. This format has to be converted into a format that is compatible with the component that would characterize the user behavior. This component of the system has to be efficient in nature as this operation is applied to every row of data in the dataset.

The final problem of the system to be solved is the environment of the system. The Operating System and the implementation languages can ultimately determine the efficacy of the system. As such the system designer must take the time to select the best platform that matches the characteristics of software component involved.

1.3 Proposed Solution

This thesis proposes a novel solution to this problem through a combined application of Computational Mechanics and Agent-Based Modeling. The solution would utilize the capability of Computational Mechanics to take time series data and produce a rule set. The resultant rule set would be used inside the Agent-Based Modeler to simulate the behavior of the humans.

The novelty of this solution is that rule set is individually generated for each user. An observer can appreciate the nature of the advantage this gives the system by simply thinking about the wide variations of humanity. A simple rigid rule set cannot hope to encompass the wide varying behavior of humans. This approach ensures that each user's particular behavior is taken into account when generating the rule set.

There is a single major downside to the selection of CM as the rule set generator. CM assume that past actions observed would be the only actions that would occur in the future. Which, when considering humans is not necessarily true. However prior research [4] had found that the CM approach is capable of predicting with an accuracy of around 80% correctly into the future. This accuracy was considered sufficient for the initial development of the system.

Another novelty of the system is the usage of Agent-based Modeling where rule set is automatically generated. In the majority of past application of ABM, model is theoretically developed then generally applied to all similar agents. This system uses unique auto-generated rules for each agent/user, ensuring heterogeneity.

Ultimately the development was carried out using a general purpose language. Using a general purpose languages to build ABM simulations is often not performed. Most ABM simulations are conducted in special software made for ABM simulations. However, this development method achieves shorter development time, higher flexibility, and high performance through the development of the ABM modeler system using a general purpose language. In reality development was not based on a single general purpose language. In fact this system uses two distinct general purpose languages and a specialized statistical programming language, three distinct languages in total, to build the whole system. While this approach might seem maverick from the point of traditional software development, however, this thesis views this kind of development as "programming as a tool development" methodology.

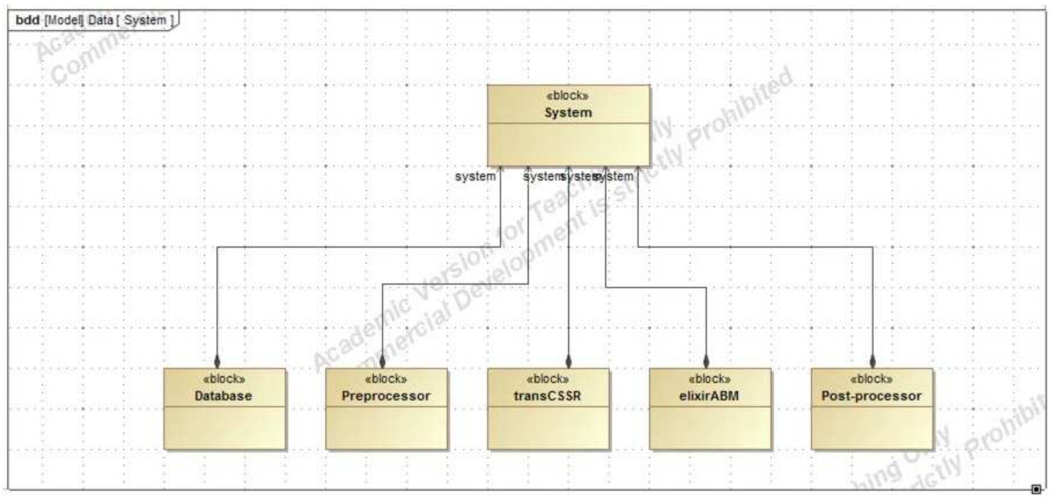


Figure 1.2: System Overview

Overview of the system is shown in Figure 1.2. The system is comprised of a database, a pre-processor, a behavior rule generator called transCSSR, an Agent-Based Modeler named elixirABM, and a post-processor that would interpret and visualize the results.

Chapter 2: System Development

2.1 Overview

System development started with a survey of existing tools and methodologies. The primary objective at this stage was to understand the behavior of individual elements that will compose the system. Understanding the individual elements are essential to the understanding of the system as a whole.

2.2 Computational Mechanics

2.2.1 Overview

This system uses computational mechanics [5] to make sense of the time series data that is the input of the system [6]. Twitter user behavior is fed into the system as a time series data set. Computational Mechanics has the capability to solve the problem of pattern discovery that is a central problem in the system. From the input time series data, this system aims to build a cohesive and homogeneous rule set. To figure out the patterns in the input data, this system will use the Computational Mechanics based algorithm named transCSSR. This section will explain the Computational Mechanics first and then give an introduction to transCSSR.

The question of pattern discovery is an enduring question from the nascent days of artificial intelligence. However one must first recognize that pattern discovery and pattern recognition are two distinct problems [5]. Pattern recognition is the task of identifying previously known patterns inside a data set and classifying then into proper categories. Pattern discovery is the task of identifying patterns that there are to be found, patterns that have been seen before and has never seen before. In addition, it complements the need to build an understanding of the processes, to give a causal explanation, to give an explanation on how a forecaster worked. This is the result of a pattern discovery algorithm. This problem is an amalgamation of several fields: unsupervised learning, data mining and knowledge discovery, causal inference in statistics, and the statistical physics of complex and highly organized forms of matter. In this context, pattern implies a kind of regularity, structure, symmetry, and organization [5].

In order to understand computational mechanics, this subsection will provide an overview of the notations used in Computational Mechanics as with relations to its application in the Twitter analysis. This thesis follows the notation published in Darmon et al. [4].

Consider the relative times of tweets or mentions in respect to a reference time. These times will be noted by $\{\tau_j\}_{j=1}^{n}$. Let the reference start time be t_0 and the time step to be Δt. From a given time stamped Twitter stream, we can then generate a binary time series $X_{i=1}^{T}$, where,

$$X_i = \begin{cases} 1: & \exists \tau_j \epsilon [t_0 + (i-1)\Delta t, t_0 + i\Delta t) \\ 0: & otherwise \end{cases}.$$

In other words terms, if the user X_i has tweeted during the time frame $[t_0 + (i-1)\Delta t, t_0 + i\Delta t)$ this results in 1, else 0. Resolution of the time frame is considered at second, minute, or hourly intervals depending on the use case.

Once the resulting stream has been encoded with the above encoding scheme, where $X_{i=1}^{T}$ is obtained, attention now turns to prediction of future events. Now we wish to predict the future based upon past events observed. We donate past finite history $X_{i-1}^{i-L} = (X_{i-L}, ..., X_{i-2}, X_{i-1})$ of L. We assume that $X_{i=1}^{T}$ is a conditionally stationary stochastic process, and that the optimal choice for $r(x_i; x_{i-1}^{i-L})$ is the conditional distribution of X_i given $X_{i-L}^{i-1} = x_{i-L}^{i-1}$. In this thesis computational mechanics is used to define the r.

Computational Mechanics results in a state-space representation of the the observed dynamics, with hidden states, noted as $\{S_i\}_{i=1}^{T}$, which represent the behavior observed, noted as $\{X_i\}_{i=1}^{T}$. This hidden state S_i when considered in a process, is called the causal or predictive state. These states are discovered states that would be corresponding to a set of all past behaviors that would result in a predictive distribution similar to observed past x_i, the observed variables. The mapping of past to labels is called ϵ.

Two pasts x and x' will have same label $s_i = \epsilon(x) = \epsilon(x')$ if and only if

$$P\left(X_i | X_{i-:}^{i-1} = x\right) = P\left(X_i | X_{i-L}^{i-1} = x'\right)$$

as probability mass functions. With CM it's now possible to use $s_i = \epsilon(x_{i-L}^{i-1})$ and

$P(X_i|S_i = s_i)$ instead of the $P\left(X_i|X_{i-L}^{i-1} = x_{i-L}^{i-1}\right)$.

In practice, the conditional distribution, $P\left(X_i|X_{i-:}^{i-1} = x\right)$ is not known. However using the Causal State Splitting Reconstruction (CSSR) algorithm, model where ϵ is known can be inferred.

2.2.2 Causal State Splitting Reconstruction

Causal State Splitting Reconstruction (CSSR) is an algorithm developed by Shalizi et al. to infer the estimated ϵ machine for a given time series data under the certain conditions [7].

CSSR is based on similar methods that would use state-splitting methods for finite state machines. It starts by assuming that the model is simple (simple state machine) and that model components should be added only when statistically justified. CSSR begins by assuming the processes is independent, identically distributed (IID) over a specified discrete alphabet. A result of this assumptions is that initial model before the process is analyzed of a random (no pattern) process. From this initial model, CSSR will use statistical tests to see when to extend the model [7]. In other words, from an initial model of a random process to a process where patterns exist as the model is extended.

The algorithm that implements CSSR can be surmised into having three stages. These three stages are summarily described bellow.

1. **Initialize** Create a initial simple model based on with IID assumption.

2. **Homogenize** Extend the model to have states that are significantly different

than each other states.

3. **Determinize** Create transitions to relevant states.

Using this algorithm, it is possible to generate a model that is maximally predictive and minimally complex [7]. This model, in a form of a state machine, now can be thought of as having all the rules of that specific agent.

2.3 Agent-Based Modeling

2.3.1 Overview

Before explaining the theories behind Agent-Based Modeling, one must first dive into the world of Complex systems. Complex Systems theory develops principles and tools for making sense of world's complexity and defines complex systems as systems that systems that are composed of multiple individual elements that interact with each other yet aggregate properties or behavior is not predictable from elements themselves. The interaction between these elements gives rise to a phenomenon called "emergence."

The concept of emergence is vital to understanding the benefit of using Agent-Based Modeling. Wilensky and Rand [8] define emergence as "The arising of novel and coherent structures, patterns, and properties through the interactions of multiple distributed elements". The important fact to notice is that emergence behavior cannot be observed via observing a single element, it can only be observed through observing multiple elements interacting with each other.

As a result of emergence, complex systems tend to exhibit several unique characteristics. Complex systems tend not to have a central coordinator. Instead, they will self-organize based on element level rule sets. This, in turn, has the effect of dynamic reforming at the system level. This behavior is mostly probabilistic at the element level.

Agent-Based Models are a collection of agents in a simulation environment. Agents are computational entities that are comprised of several properties. Namely "state variables" and "values". State variables contain the state of the agent while value variables contain internal properties to the agent. In addition to these variables, the agents contain a set of behavior rules.

One another consideration when conducting Agent-Based Modeling is the environment. Agent-environment interaction is a significant observation in itself. The environment may offer obstacles, set time or any other non-agent interaction in the simulator.

The methodology of Agent-Based Modeling is simple in its essence. The primary objective is to encode a set of rules into each agent. Then by observing agents and its communication between other agents and with the environment, one can start to gain an understanding of the complex interactive behavior of the system as a whole.

2.3.2 Agent-Based Modeling vs. Other Modeling Techniques

If Agent-Based Modeling can be used for understanding systems, what other methods are out there that can achieve or at least propose to achieve the task of simulating complex system? This subsection briefly compares other published methods in simulating complex dynamical systems.

The most common form of simulation found is the equation form. Termed Equation-Based Modeling (EBM), EBM takes form of an equation that captures the behavior of the system. However, EBM is dependent on the assumption of heterogeneity. The equation must be able to capture all behaviors of the system. Also, EBM tends to treat the population as a continuous quantity, while it is commonly understood that elements of the systems tend to behave is a discrete manner. As a result forcing the EBM to make assumptions that population size is large and spatial effects are unimportant [8].

Another advantage of ABM is that in order to successfully model it is not necessary to gain an understanding of the aggregate phenomena, while in EBM it is necessary. Also, as the system behaviors can vary widely it is essential in EBM to test the hypothesis to ensure that it confirms the aggregate behavior. Another factor is that to encode a set of rules as equations in EBM, it might require very complex mathematical formulations. Meanwhile, an ABM system would simply require a set of rules.

One of the major advantage in ABM over EBM is the fidelity of the simulation and resulting dataset. ABM produce both the element level results and the system level results during a simulation. EBM will only produce the system level results, leading to loss of fidelity in the dataset. As a result, many EBM will not have the property of emergence. Furthermore, ABM's allow indirect causation via emergence resulting significant advantage to the outcome of ABM simulation [8].

2.3.3 Randomness in Agent-Based Modeling

Randomness in agents is an important factor when building ABM simulators. Most Agents has embedded rule sets that are activated upon a probabilistic distribution. Hence random number generators are an important factor in designing ABM system.

Producing true randomness in software is nearly impossible. Most systems that require true randomness uses external hardware systems that are very expensive and hard to purchase. These external systems mostly rely on quantum mechanics to obtain truly random numbers. That brings us to the question if there is a need for a truly random number. In fact, most real-world applications, even in the field of cryptography, do not use true random number generators. Most widely used random number generators are Pseudo-Random Number Generators.

Pseudo-Random Number Generators, or PRNGs for short, are used in computer science to generate a number that looks random. Designing PRNGs is a field of its own, hence would not be discussed in detailed in this thesis. However, it is a field of particular interest to any simulator designer due its importance on the results and the nature of PRNGs that leads to them being one of the main components of a simulator system. One of the most important factors in selecting PRNGs is the period of the PRNG. As PRNGs cannot produce a truly random number, it tends to have cyclic behavior. In other words, a previous number can start appearing after a certain period. The larger the period, the better the randomness of the PRNG.

One of the most popular PRNGs available today is the Mersenne Twister PRNG. Mersenne Twister remains the PRNG of choice in R, Matlab, and, Python to name the few. Utilizing Mersenne Twister of good implementation should be sufficient for most ABM simulators. A good implementation must have undergone validation testing for the specified period.

Chapter 3: Implementation

3.1 Overview

This chapter details the implementation of the simulation system. The simulation system contains the rule-set generator, the agent-based modeler called elixirABM and the glue code that integrates the system. The rule-set generator is a modified 3rd party tool named transCSSR. This chapter primarily deals with the building of elixirABM from scratch. This chapter will also contain a brief introduction to the transCSSR tool. Finally, this chapter will describe how all of these components are integrated together to produce a single system.

3.2 elixirABM

elixirABM was designed from the start to be an Agent-Based Modeler that utilizes a general purpose language. It was imperative that qualities of general purpose language to be present due to the shortened time scale of development, and that many tools that are integral to an Agent-Based Modeling Platform to be part of the language or available through 3rd party libraries. Another factor that influenced the design is the learning curve of the language. It was determined that elixir programming language, a derivative of Erlang programming language, would be a suitable candidate for this work.

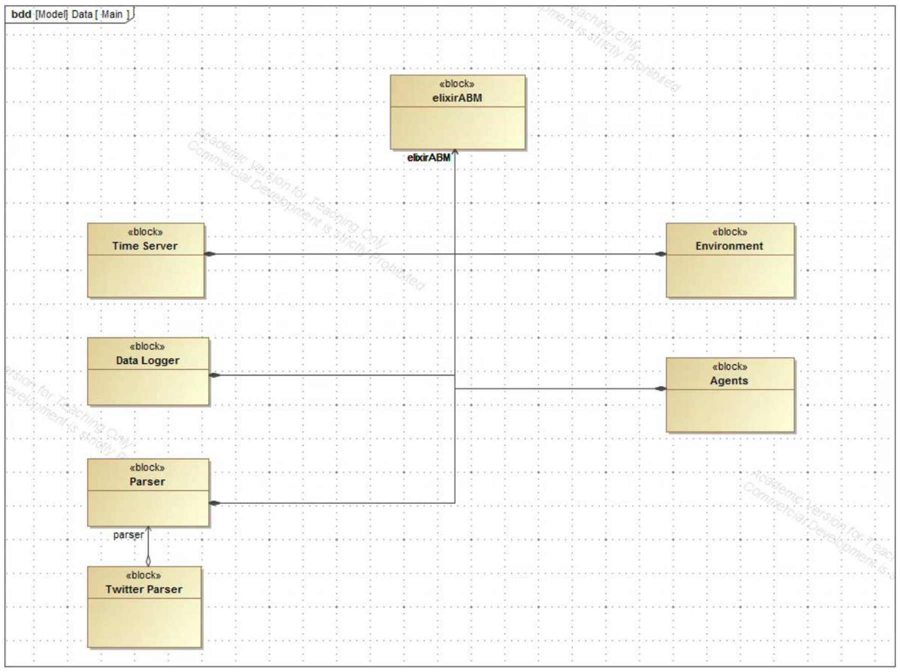


Figure 3.1: elixirABM Overview

3.2.1 Erlang as a general purpose Agent-Based Modeler

There has been at least two known occurrences of Erlang being selected as an Agent-Based Modeling Platform [9] [10]. The primary reason for selecting Erlang is the Actor Model of a programming paradigm that is the core of Erlang language. The Actor Model has a 1-to-1 mapping of how agent communicates in an Agent-Based Modeler to how the programming language is structured [11]. This fact is shown graphically in Figure 3.2.

To understand the Actor Model one must first understand threading, a concept from computer science. Thread is an execution unit which consists of a unique program counter, a stack, and a set of registers [12]. Threads are also known as lightweight processes. Threads are a popular way to improve application performance through parallelism. In typical threads, the CPU switches rapidly back and forth (performing context switches [13]) among the threads giving an illusion that the threads are running in parallel. In multi-core and multi-processor machines, threads are capable of running parallel as long as the operating system kernel supports it. As each thread has its own independent resource for process execution, multiple processes can be executed in parallel by increasing the number of threads.

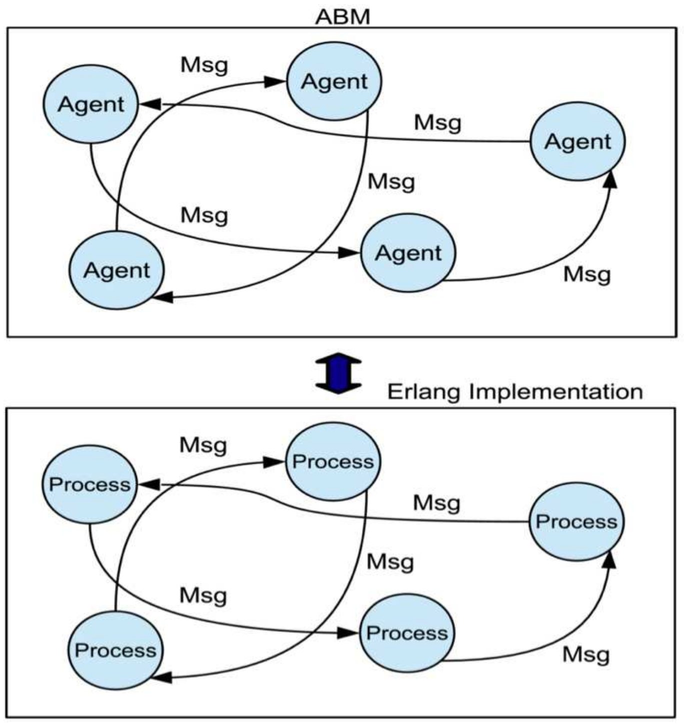


Figure 3.2: Erlang 1-to-1 mapping with ABM

Mapping as shown in Figure 3.2 occurs due to the nature of the threading model Erlang follows [11]. Erlang threads are not the natural Operating System

(OS) threads as introduced in computer science textbooks. Natural OS threads have very high overheads to them. Invoking them in large numbers will degrade the performance of the system heavily. Erlang threads are lightweight and exist inside the Erlang Runtime. Scheduling of the Erlang threads is done using the Erlang scheduler, not the OS scheduler. These properties make it possible to invoke from thousands to millions of threads without a substantial performance penalty. As a result, Erlang code is inherently concurrent, and can fully utilize the multi-core and multiprocessor hardware available today. Also, one can even extend this by inter-networking Erlang Runtimes in multiple machines. As such, developing software with an Actor model as shown in Figure 3.2 becomes possible with Erlang threading model [11].

In addition to containing a favorable language paradigm, Erlang runtime is excellent in Input/Output (I/O) bound operations. ABM workloads are not computationally intensive. There are no significant repetitive calculations such as integration or differentiation involved in Agent-Based Modeling. However, a large number of messages are passed during Agent-Based Modeling. These communications are I/O bound operations. As a result, Erlang tends to be efficient in Agent-Based Modeling workloads.

Another advantage of Erlang is that data structures are immutable by design [14]. As a result, any data structure initialized cannot be modified after initialization. If a variable with the same name is to be updated, a new part of memory will be allocated anew. If part of the variable has to be updated, for example adding a new value to a dictionary, this means the current memory will be copied and inserted the new value. Many traditional programmers will find this arrangement as a disadvantage, however in a highly concurrent application like Agent-Based Modeling; this forces the developer to write efficient, safe and decoupled code. By decoupled code, it is meant as a having non-shared variable. Every agent will have a unique variable which cannot be modified by other agents or the environment. As a result, every information is passed on to the agent as a message [15]. It is up to the agents' internal rules to properly take action and change internal variables. As stated above, this ensures every agent is efficient, safe and decoupled.

In addition to being very competent from a technical standpoint, Erlang is very mature from an ecosystem standpoint. When choosing a programming language, ecosystem strengths are a significant consideration. It is paramount that technical support during development can be obtained in online forums and chat rooms. These technical resources must be able to aid in solving extremely hard problems with the help of community support. Also, the applications under development should be able to be extended via 3rd party libraries that are computationally efficient and has short integration time.

While Erlang has many good qualities and especially suited for Agent-Based Modeling tasks, it has a significant drawback that is hindering its adaptation in the general population. The issue in question is Erlang's steep learning curve. This issue of the steep learning curve can be attributed to two main problems. One is Erlang programming syntax. Second is the functional nature of the Erlang language.

Erlang language syntax has its roots in Prolog syntax [16]. While Prolog is mature and widely used programming language, it is an old language. Many young programmers do not learn Prolog or use a language that is influenced by Prolog. Many popular languages such as R, Python, and Java are primarily influenced by C programming language; this leads to a significant unfamiliarity for first-time users of Erlang. This, in turn, creates a very large learning curve.

In addition to problematic syntax, Erlang is a functional language. Functional programming style which models computations as evaluations of expressions. As such, there are no conditional evaluations such as if..else constructs that are a mainstay in procedural languages. They are in turn replaced by case statements. Also, functional languages do not have loop constructs such as "for" loops commonly found in procedural languages, they are replaced by recursive functions. This broad variation from procedural languages induces a significant learning curve to the programmer as the programmers now have to change the way they think. Another factor is the requirement to learn functional data structures. Learning a new kind of data structures is essential as many data structures used in procedural languages assume mutability in the language. This, in turn, increases the learning curve.

3.2.2 elixir

elixir is language that is built on top of the Erlang Virtual Machine. In fact, elixir code ultimately compiles into Erlang Bytecode. Hence it brings all of Erlang's advantages while minimizing disadvantages in addition to new features introduced by elixir itself.

elixir language syntax is more familiar to modern programmers. elixir language is heavily influenced by Ruby syntax, which in turn was influenced by C to some extent. Any programmer who has used a language developed in past decade can feel right at home using elixir.

Another advantage of using elixir is the maturity of both elixir and Erlang. Erlang has a history spanning 30 years with a sprawling ecosystem. elixir is much younger and has only 3 years of history. However, the language specification is stable for the version 1 of the language. Besides, elixir community is highly active and aggressively growing with companies in silicon valley showing a great interest in the ecosystem. Ultimately this leads to a better and a longer lasting foundation to the system.

elixir threads automatically gain several attributes inherited from an Erlang thread [17]. Specifically, the thread inherits a state and a message inbox. Also, these threads can be run in any core in today available multi-core computing environments, increasing performance significantly. Furthermore, it opens up the possibility where in future, with further development, the system can be made in such way that system can be operated on multiple machines available in cloud computing environments.

The elixir threads keeps the local variables in memory. These local variables cannot be shared with other threads, as such scope remains exclusive to the thread that owned the variable [17]. The consequence of this is that programmer does not have to worry about thread synchronization issues. Thread synchronization issues are issues that pop up when dealing with multiple threads modifying the same variable at the same time [18]. Solving such problems are hard and time-consuming work, and also result in undetectable errors in code. A programmer who does not have to deal with such issues can develop highly parallel software in a very short amount of time. This advantage is available in this system. However, it is a necessity in most coding problems to have two or more threads communicate with each other. Since accessing common variables are not possible, some other method must be used to communicate. elixir solves this by message passing [17].

elixir message passing is a tool aimed at sending data between multiple threads. Data sent are copies of the original data that can be any data type or a combination of data types. elixir message passing system supports asynchronous and synchronous message passing. Synchronous messages will block the originating thread until destination thread process the message and acknowledges the message as received. On the other hand, threads that send asynchronous messages will not expect a destination thread to acknowledge the messages as received; hence they are non-blocking messages. Messages that are not processed immediately are queued in an inbox, where the thread can process at its leisure [17].

Ultimately, this threading model which is combined with message passing system allows seamless programming of highly concurrent applications.

3.2.3 elixirABM implementation

This section details the inner working of the elixirABM. elixirABM is heavily dependent on elixir's inherit threading model. Every block shown in Figure 3.3 is created as a separate thread in elixirABM.

elixirABM implementation attempts to make a 1-to-1 mapping of Actor model

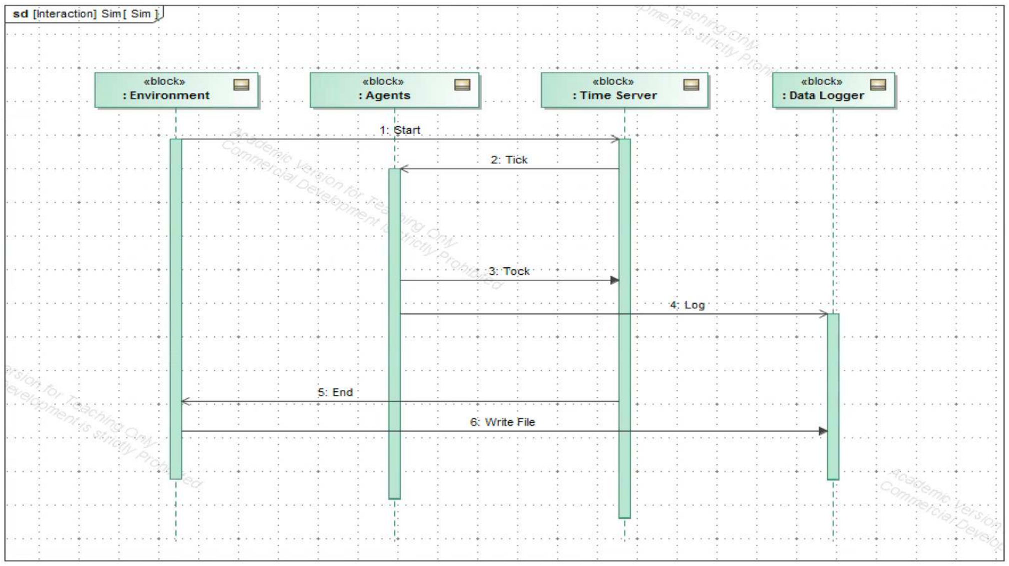


Figure 3.3: Sequence diagram for elixirABM simulation

of programming used in elixir/Erlang to network linked agents. In the Actor Model, Actor is a primitive unit of computation. In elixir/Erlang, this is represented as a thread. In elixirABM, we conceptually and practically allocate an Agent as an Actor. As such, the agent automatically gains the threads message passing and multithreading capability. This also results in thousands to millions of threads been invoked. In fact, the number of threads is defined by the number of Agents the user wants to spawn. Agents can act on its accord depending on the state of the system.

In elixirABM, the agent internally carries its Computational Mechanics derived state machine, a message inbox to denote if the agent has received a tweet or not, a list of friends of the Agent, and, the current state of the state machine. This information is fed to the agent at the moment of its spawning. After initialization, these internal variable can be modified only by the agent itself. Any external input, for example, a signal indicating that Twitter message has been sent to the Agent, has to be sent through the message passing system. Once the message is received by the Agent, the Agent will perform the appropriate computation and increment the internal state. Agents internal state machine is heavily dependent on the uniform random number generators to produce correct results. As Erlang PRNG is known to be faulty, elixirABM uses a 3rd party library that computes the uniformly random number using a Mersenne Twister algorithm.

The initialization of the agent's internal state at the start is a major step during the simulation. A parser module was written so that JSON (JavaScript Object Notation) formatted state machine data can be imported. This parser module will read the JSON file and convert the data embedded within into an internal data structure. This new information is then used for initialization of the individualized Agents.

Now the question becomes how to properly manage these individualized Agents. We use an another thread, named the Environment thread, to manage these individual Agents. Environment thread is a supervisory thread that provides information pertinent to operation of other threads. Environment thread remembers and provides other threads information on relevant threads such as time server and data logger.

Time server thread acts as the thread that ensures the entire simulation is in-sync with the time step. A time step is a unit of time where the period of the time depends on period set on input data. It is safe to say that inside the system, time step has no unit of time attached. At the start of the time step, the time server indicates to all the agent threads that new time step that started. Agents will perform their assigned processing tasks and once completed, individually will report to the time server, that that particular agent has completed its task. Time server will continuously monitor if all agents have completed its tasks, if so it will start a new time step. This work will be performed until the specified time steps have elapsed, at which point it will notify the data logger to save all results.

The data logger thread is a particular thread that is in charge of logging any data the user has specified to log. The data logger thread utilizes Erlang standard library component named "ets". ets is the component within Erlang standard library that is employed to provide term storage to applications [19]. It has significant similarities to key-value pair databases. The advantage of this component is that it is an Erlang library component, as opposed to 3rd party solution. Hence ets is available within any default Erlang installation. ets allows a large amount of data to be stored in memory for logging purposes.

3.2.3.1 Pseudo-Random Number Generators in elixir

elixir inherits the random number generators embedded within Erlang. Unfortunately, Erlang PRNG is notably weak [20]. Hence alternative methods of producing a PRNG was considered.

In elixirABM, the PRNG in use is Tiny Mersenne Twister (TinyMT) for Erlang [21]. This 3rd party library has been tested for its specified period.

3.2.4 elixirABM Verification and Validation

Like any system, elixirABM must go through rigorous Verification and Validation (V&V) process to ensure that build is correct and working as expected. The primary method of V&V used in elixirABM development is to use Unit Testing methods.

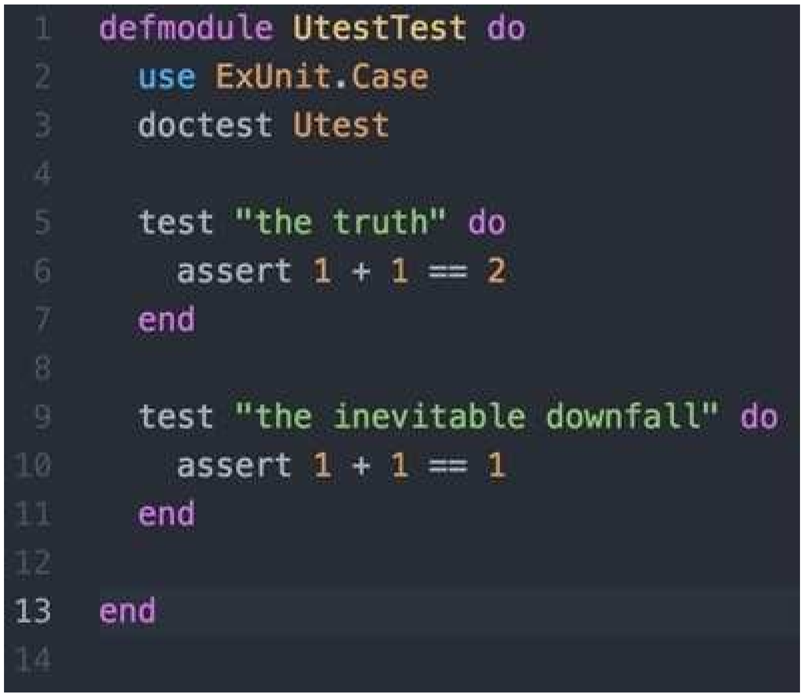
3.2.4.1 Unit Testing

The Microsoft Developer Network documentation defines unit testing as follows: “The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect” [22]

A Prime example of a unit test is the testing of a function. For example, a function written to add two numbers may have a unit test that tests the said function by providing two values and checking the result by comparing with an expected value that has been calculated beforehand.

This is an another place where the earlier decision to use elixir as the implementation language shines. elixir natively supports a unit testing framework named ExUnit [23]. Utilizing the ExUnit framework, elixirABM was written with a significant amount of unit tests that was written concurrently with the functional code.

Figure 3.4 shows a example unit test file. This file contains two unit tests of one which will fail. This example exemplifies how unit test scripts are written.



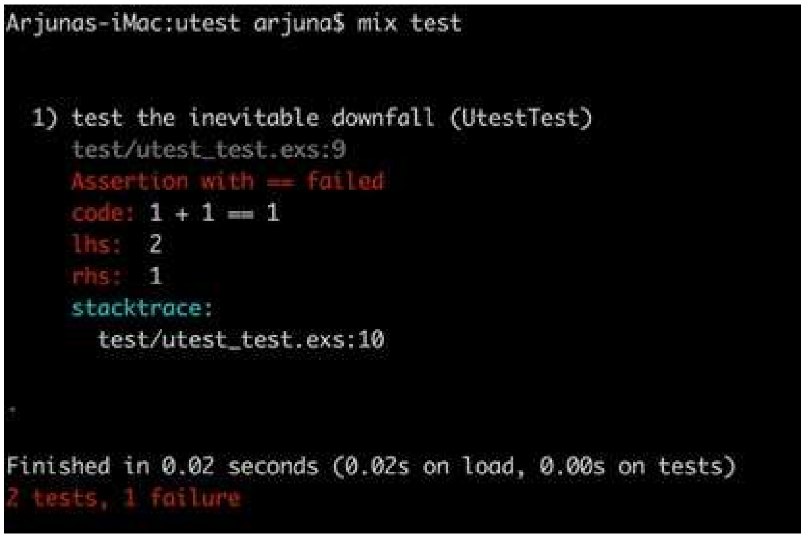
```
defmodule UtestTest do
  use ExUnit.Case
  doctest Utest

  test "the truth" do
    assert 1 + 1 == 2
  end

  test "the inevitable downfall" do
    assert 1 + 1 == 1
  end

end
```

Figure 3.4: Unit Test Example

They are written as a single function that would call the function under test with predefined inputs. Then the resulting output is compared with known output to check if the function is working correctly or not.

Unit tests in elixir are invoked with a single command. This command will run all unit test scripts within the project and state the results. Results of running the above example unit test script are shown in Figure 3.5.

The importance of unit testing is significant in ABM. The idea of ABM where you find emerging and complex phenomena is dependent of the premise that individual agents exhibit correct behaviors. Unit testing allows us to test at the agent-level for correct behavior. The assumption is that if an agent behaves correctly for all instances individually, they will exhibit correct behavior when Agents are interact-



```
Arjunas-iMac:utest arjuna$ mix test


  1) test the inevitable downfall (UtestTest)
     test/utest_test.exs:9
     Assertion with == failed
     code: 1 + 1 == 1
     lhs:  2
     rhs:  1
     stacktrace:
       test/utest_test.exs:10

.

Finished in 0.02 seconds (0.02s on load, 0.00s on tests)
2 tests, 1 failure
```

Figure 3.5: Running The Unit Test Example

ing with each other and the environment. The results of units tests written for elixirABM is detailed in Appendix A.

3.3 transCSSR

transCSSR is an application created by David Darmon as a part of his dissertation [24]. transCSSR implements a modified form of Causal State Splitting Reconstruction (CSSR) algorithm for inferring epsilon-transducers from data generated by discrete-valued, discrete-time input/output systems. In this system, a modified version transCSSR is used to generate behavior rules for the Agents.

transCSSR takes the input as form files with a string of discrete values of inputs and outputs of the system. Once the input/output alphabets are defined within transCSSR, it can generate the casual model and run a rudimentary statistical analysis of the efficacy of the casual model.

transCSSR was modified such that rather than outputting the statistical metrics, output to be a causal model itself, the causal model, structured as a state machine is converted into JavaScript Object Notation (JSON) file format and written to disk. Converting the output to JSON format allowed those files to be read efficiently by other programs within the system, in this insistence by elixirABM.

JSON is a lightweight data-interchange format that is easy for machines to read and write efficiently while enabling humans to see the underlying format easily [25]. Also, JSON parsers are commonly available in many languages and are used widely. Hence, widely used JSON parsers tend to be very efficient in reading JSON format files.

In addition to modifying the output, transCSSR underwent modifications to enable multithreaded invoking. This modification was deemed a necessity as the number of agents within the simulation started to increase, the rule generating potion of the system, transCSSR, was deemed as a bottleneck. Now transCSSR will compute rules for agents one less than the number of cores available in the computer simultaneously. For example, if the computer has 32 cores, transCSSR will be simultaneously computing causal models of 31 agents at a given time.

3.4 System Integration

Once the two main components of the system were developed, the system now has to be integrated.

System integration was done through the use of Python and R scripts. These scripts would handle from extracting the relevant data from the database, computing them and finally showing the results in a visual manner. Achieving integration through scripts was chosen as this will provide maximum flexibility going into future. Not only are these scripting languages OS independent, but scripts can also be modified independently of each other allowing future users to tailor the system to their use cases easily.

3.4.1 Extraction and Pre-processing

This system is designed in a manner that data is initially stored in a MySQL database. This subsection deals with extracting this data from the database and pre-processing them to be suited as the input for the rule generation element, transCSSR. Details provided in this section is meant to be re-created for different use-cases depending upon how the data is stored.

Data used for analysis in this thesis was collected by a program named TwEater. TwEater would archive every public tweet made by the specified users inside a specified MySQL database. Data from the database is extracted through the MySQL client command line interface. The extracted data are stored in a tab separated file.

Once the raw data is extracted, they have to be converted into a format that can be understood by transCSSR. This is achieved through an R language script. This script is responsible for reading the files and processing information depending on the desired time resolution. For example, any tweeting within 10-minute interval can be represented as a single event.

Resulting from this stage is set of files that have the Twitter ”userid” as the filename that contains the input/output for each user.

3.4.2 Rule Generation

The next step of the process is to generate a rule set from those input/output files. To generate these rules, transCSSR will be invoked.

This modified version of transCSSR will be invoked using a specially written python script. This script will get path info of the previously generated files and a list of userids as an input. After the invocation, this script will automatically load individual files and invoke transCSSR in a parallel manner. The result of this section, a set of JSON files, will be written in a folder specified.

3.4.3 ABM Simulation

In this section previously generated rule sets, input/output files and a userid list will be used to initialize and run elixirABM.

First, in the configuration section of the elixirABM, the correct path must be specified. Afterward, inside the elixirABM code, the required output data, and initialization depth must be specified. Currently, elixirABM is capable of outputting the individual agent inputs, outputs, states, and probabilities at each output. Initialization depth in this context is how many time-steps are run using original data rather than outputs of the agents.

Once all the configuration is completed, the simulation can be run using the start command. After the simulation is completed, elixirABM will write the requested outputs into a file.

3.4.4 Visualizing Results

Now the output files must be processed and analyzed. Python was selected as the language to create a script that would be useful in analyzing results. Python is selected as the language of choice as in contrast with R-language, file processing is significantly faster.

This script uses the Pandas toolkit for high-performance file parsing. The use of Pandas was necessitated as the performance of other alternative methods were significantly inferior for the required scale of file handling. A rudimentary benchmark was performed on Pandas importer, Python native importer, and R importer. R importer was unable to handle the scale of the files and promptly crashed after invocation. Python native importer was fast at the start but several minutes into the importing process it crawled to an unacceptable slow pace. Pandas importer, with C engine selected, managed to load the required files within an acceptable time frame.

After importing the data, numpy and matplotlib were used to process and visualize data. These visualizations are shown and described in the next chapter.

Chapter 4: Results

4.1 Overview

Now since the simulator system has been built its time to run a simulation on it. This chapter discusses running a 15,000 agent simulation created based on Twitter user behaviors.

4.2 Background

To perform a large-scale Twitter user simulation, this thesis uses a database of 15,000 users whose public Twitter behavior has been logged over a period of one year. The ultimate objective is to create a simulation of those 15k users that would closely mimic their online behavior. As such one of the most important measures of accuracy for the simulations would be the capability of the Agents to track/follow the tweeting patterns shown in the real world behaviors.

4.3 Optimal Lookback selection

Lookback (L) is a parameter in CSSR algorithm. In vernacular terms, lookback sets the maximum number of i steps CSSR will look back from $t_{i=0}$ to create a unique state.

This parameter influences the performance of the transCSSR and the efficacy of the results. This thesis considers the size of the resulting file and the speed to completion as an performance factor of the transCSSR. Hence before running the simulation, an optimal value for L must be chosen beforehand. To obtain the optimal L value, transCSSR was run multiple times with varying L. The result of that run is shown in Figure 4.1.

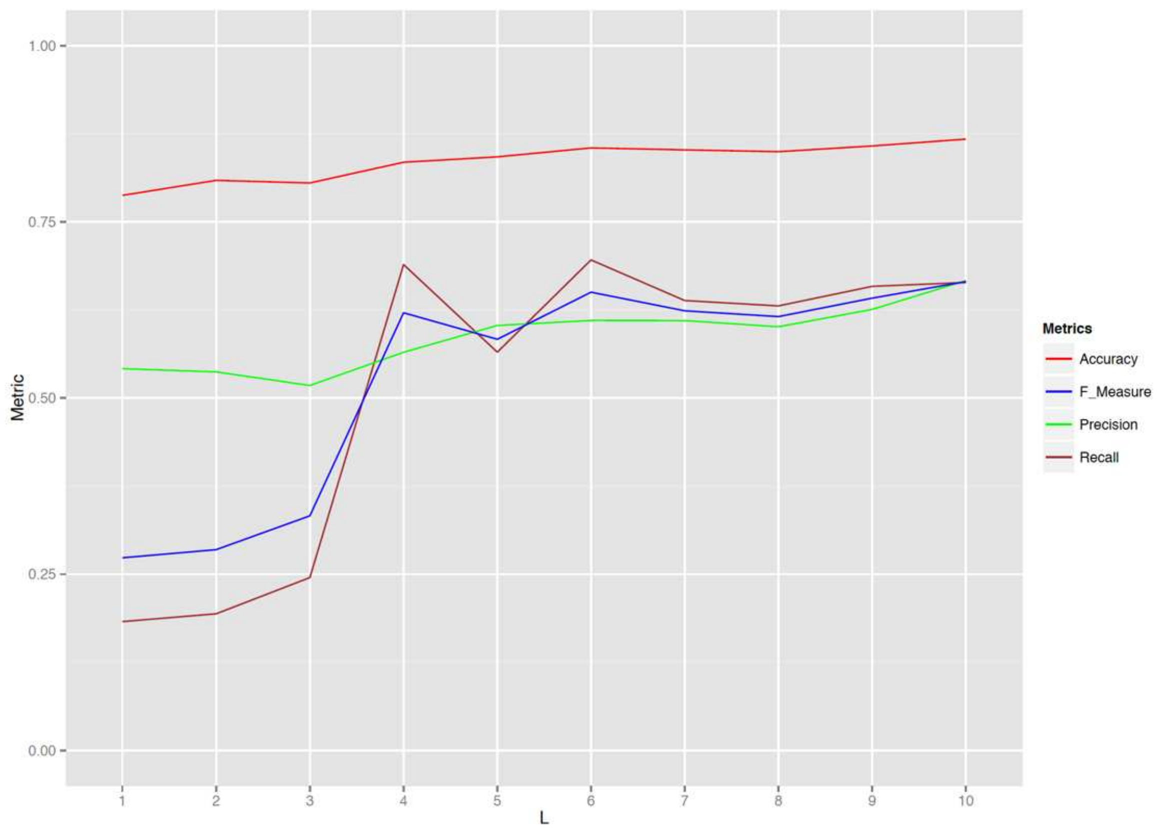


Figure 4.1: Change in prediction performance metrics with L

Before detailing the performance metrics used, few new terminology must be introduced. These performance metrics detailed here follows the definitions of

Fawcett [26]. The term Instant is used for an event. The classifier is the result of the computer in which the computer determined the type of the event. If the instance is positive and it is classified as positive, it is considered as a true positive (TP). If the instance is positive and it is classified as negative, it is counted as a false negative (FN). If the instance is negative and it is classified as negative, it is counted as a true negative (TN). If the instance is negative and it is classified as positive, it is counted as a false positive (FP). The total number of positives are noted as P, while the total number of negatives is noted as N.

Using above definitions, Fawcett [26] defines following performance metrics.

$$accuracy = \frac{TP + TN}{P + N}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{P}$$

$$F - measure = \frac{2}{1/precision + 1/recall}$$

After considering performance metrics obtained through Figure 4.1, it was decided to use lookback value of 6 as the nominal value in future simulations.

4.4 Twitter Simulation Results

This section describes the results obtained by running the 15k Twitter user simulation.

Figure 4.2 shows the first attempt at simulating using the developed system. In the figure, the volume of tweets per time stamp is indicated. Volume is the

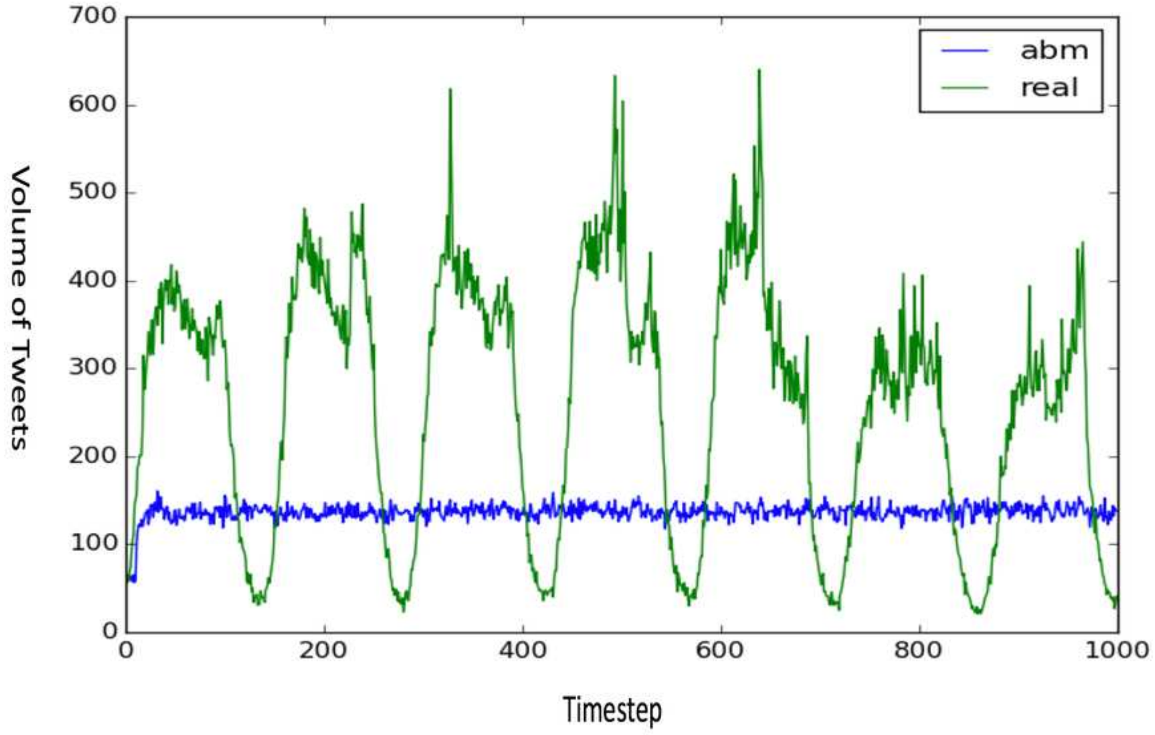


Figure 4.2: Simulation vs Real Tweeting by Volume

aggregate number of agents who tweeted during that time step. A time step in this instant is 10 minutes in real life. If one observe the start of the simulation, it can be seen that the simulation starts to follow the real life and then falls off.

A zoomed-in diagram with extended initialization depth is shown in Figure 4.3. In this figure, another attempt at improving the simulation is shown. In this simulation, initialization is done up to the 50^{th} time step. As such inputs (incoming messages) are is extracted from the real life data stream and used in the state transitions, while the output is recorded. As seen from the Figure 4.3, this does not significantly improve the tracking.

Figure 4.4 shows a diagram where the time resolution has increased to 1-hour

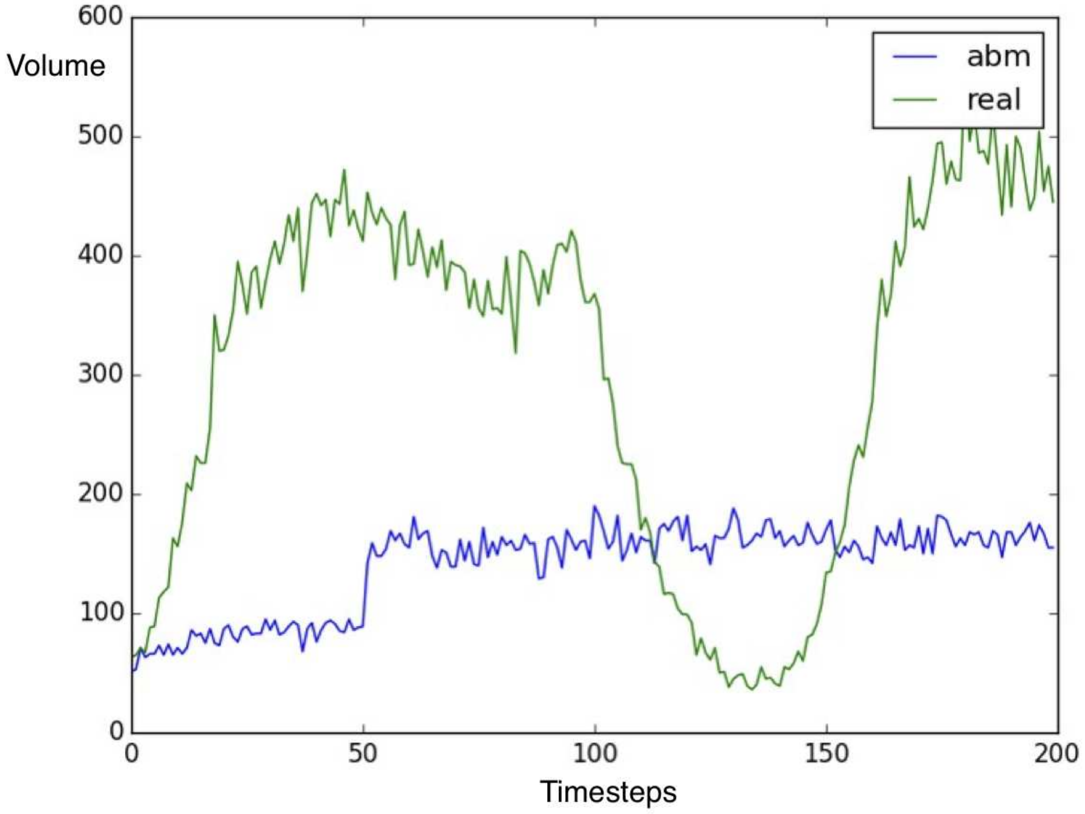


Figure 4.3: Simulation vs Real Tweeting by Volume with extended initialization depth

intervals, while the start of the simulation is now near a peak time of tweeting. In this case, it can be clearly seen that tracking has now improved. However, after one cycle it seems unable to keep track with the real life performance.

One hypothesis for this fault is that the rule generation method selected do not accurately capture the seasonality changes of the real life tweeting. Integrating this seasonality changes would be a complex task. One method is to input the seasonality changes as a separate input to the CSSR algorithm. Complexity comes from the resulting state explosion due to the highly enlarged finite discrete alphabet

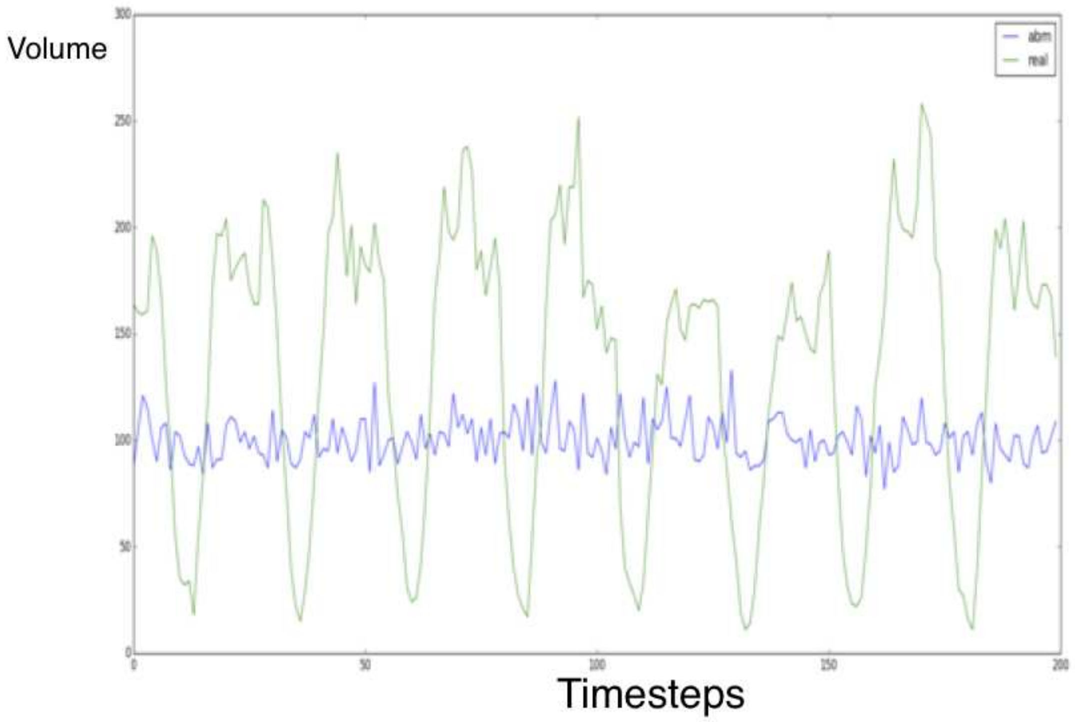


Figure 4.4: Simulation vs Real Tweeting with larger time resolution

defined. This, in turn, leads to a slow performance from the rule generator.

Chapter 5: Conclusion & Future Work

5.1 Conclusion

This body of work details the successful attempt to build a large scale Agent-Based Modeling simulation system directly from big data. The system developed, covers from the initial point of data extraction to visualization of results. In this thesis, Computational Mechanics is used to automatically generating behavioral rules for agents to follow. Then this thesis details the development of a simulator from the scratch that is capable of utilizing current multi-core computing platforms. Then finally, this thesis describes how all the components are integrated together to produce one cohesive system.

5.2 Future Work

One major area of improvement to this system will be the addition of seasonality to the rule generator. A rule generator that can automatically identify the seasonality changes should improve the tracking quality massively, negating a major drawback of the current system.

Another area of improvement would be to add a browser-based Graphical User

Interface (GUI) for interaction with the system. Currently, all interactions are done through the command line. This should massively increase the audience of the system.

An another major improvement would be to add the capability to conduct long-term simulations. For example, agents can be setup so that they are actively tracking real life users over periods of moths and years. Erlang's code hot-swapping capability can be used to realize this long-term task.

This thesis has much that can be improved upon, but it is the first attempt at building at generating an agent-based model directly from data. This has the potential to open gateways to analyze big data through the creation of agent-based models that would forecast wide varying instances of life.

Appendix A: List of Unit Tested Functions

Function Name	Pass/Fail
`ABM.DataStore.start_store`	Pass
`ABM.DataStore.insert_list_new`	Pass
`ABM.DataStore.get_list`	Pass
`ABM.DataStore.get_table_as_list`	Pass
`ABM.DataStore.save_table_as_csv_file`	Pass
`ABM.DataStore.pop_value`	Pass
`CMAgent.Agent.get_state_transitions_from`	Pass
`CMAgent.Agent.get_state_transitions_to`	Pass
`CMAgent.Agent.remove_extraneous`	Pass
`CMAgent.Agent.get_next_move_memoryless`	Pass
`CMAgent.Agent.get_next_move`	Pass
`CMAgent.Parser.resolve_machine`	Pass
`CMAgent.Parser.read_JSON_file_byline`	Pass

Function Name	Pass/Fail
`CMAgent.Twitter.Parser.get_userid_from_username_file`	Pass
`CMAgent.Twitter.Parser.get_machine_for_userid`	Pass
`CMAgent.Twitter.Parser.get_network_from_file`	Pass

Bibliography

[1] James P. Bagrow, Dashun Wang, and Albert-Lszl Barabasi. Collective response of human populations to large-scale emergencies. *PLoS ONE*, 6(3):1–8, 03 2011.

[2] Yichuan Jiang, Senior Member, and J C Jiang. Diffusion in Social Networks : A Multiagent Perspective. 45(2):1–16, 2014.

[3] Tweater. `https://github.com/centerforcomplexityinbusiness/tweater`.

[4] David Darmon, Jared Sylvester, Michelle Girvan, and William Rand. Understanding the Predictive Power of Computational Mechanics and Echo State Networks in Social Media. 2012.

[5] Cosma Rohilla Shalizi and James P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of Statistical Physics*, 2001.

[6] James P. Crutchfield. Between order and chaos. *Nature Physics*, 8(1):17–24, 2011.

[7] Cosma Rohilla Shalizi, Kristina Lisa Shalizi, and James P. Crutchfield. An Algorithm for Pattern Discovery in Time Series. page 26, 2002.

[8] William Rand Uri Wilensky. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo.* MIT Press, 2015.

[9] Antonella Di Stefano and Corrado Santoro. eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang. *Woa*, 2003.

[10] Carlos Varela, Carlos Abalde, Laura Castro, and Jose Gulías. On modelling agent systems with erlang. *Proceedings of the 2004 ACM SIGPLAN workshop on Erlang - ERLANG '04*, pages 65–70, 2004.

[11] Gene I. Sher. Eliminating The Conceptual Gap Between The Programming Language & ABM. Technical report.

[12] Rashid Bin Muhammad. Operating systems notes. `http://www.personal.kent.edu/~rmuhamma/opsystems/myos/threads.htm`.

[13] Context switch definition. `http://www.linfo.org/context_switch.html`.

[14] The erlangelist: Working with immutable data. `http://theerlangelist.blogspot.com/2013/05/working-with-immutable-data.html`.

[15] Concurrent programming. `https://www.erlang.org/course/concurrent-programming`.

[16] Erlang academic and historical questions. `http://erlang.org/faq/academic.html`.

[17] Processes. `http://elixir-lang.org/getting-started/processes.html`.

[18] Vincent Gramoli. More than you ever wanted to know about synchronization: Synchrobench, measuring the impact of the synchronization on concurrent algorithms. *SIGPLAN Not.*, 50(8):1–10, January 2015.

[19] ets. `http://erlang.org/doc/man/ets.html`.

[20] Pseudo random number generation in elixir. `http://www.cultivatehq.com/posts/pseudo-random-number-generator-in-elixir/`.

[21] tinymt-erlang: Tiny mersenne twister (tinymt) for erlang. `http://jj1bdx.github.io/tinymt-erlang/`.

[22] Unit testing. `https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx`.

[23] Testing - elixir school. `https://elixirschool.com/lessons/basics/testing/`.

[24] David Darmon. *Statistical Methods for Analyzing Time Series Data Drawn from Complex Social Systems*. PhD dissertation, University of Maryland, College Park, 2015.

[25] Json tutorial. `http://www.w3schools.com/json/`.

[26] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.