January 2007

Agent Based Modeling of Electronic Markets To Analyze the Sustainability of Mutual Cooperation

Ravindra R. Lote
University of Massachusetts Amherst

AGENT BASED MODELING OF ELECTRONIC MARKETS TO ANALYZE THE SUSTAINABILITY OF MUTUAL COOPERATION

A Thesis Presented

by

RAVINDRA R. LOTE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

May 2007

Industrial Engineering and Operations Research

AGENT BASED MODELING OF ELECTRONIC MARKETS TO ANALYZE THE SUSTAINABILITY OF MUTUAL COOPERATION

A Thesis Presented

by

RAVINDRA R. LOTE

Approved as to style and content by:

_____
Abhijit Deshmukh, Chair

_____
Robert Gao, Member

_____
Sara McComb, Member

_____
Mario Rotea, Department Head
Mechanical and Industrial Engineering

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to Dr. Abhijit Deshmukh for his guidance and encouragement throughout the entire course of this research. His ideas and concepts have had a remarkable influence on my thesis as well as my professional career.

I am also thankful to my committee members, Dr. Robert Gao and Dr. Sara McComb for their constructive criticism and helpful suggestions.

It is a matter of great pleasure and pride to be associated with an institute like Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst. I want to make every use of this opportunity to express my gratitude towards this department and university.

My sincere thanks are due to all the members of C. D. D. M. laboratory for their detailed review and excellent advice during the preparation of this thesis.

Finally, I am forever indebted to my parents, sister, wife and childhood friend Rohit for their understanding, endless patience and encouragement when it was most required.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

MOTIVATION

According to game theory, a dominant strategy of Prisoner's Dilemma game is defecting (Epstein, 1997). Online trading between two strangers falls in the realm of a Prisoner's Dilemma (Yamamoto et al, 2003). Needless to mention, failure should be the only logical conclusion of such electronic commerce situation since a trader might never have to deal with the same buyer again given the enormous population of online traders. Thus one could argue that markets like eBay should never exist. Then what is the reason behind resounding success of such electronic markets? The answer lies in the reputation system that they established. Market reputation means a lot to the trading community, simply because it directly impacts their business prospects. It is thus the driving force behind responsible behavior of the otherwise selfish online traders.

In fact, importance of reputation is recognized by those outside the trading community as well. People in general are more likely to shy away from irresponsible behavior if their reputation is at stake. 'Google page rankings' and reader reviews on various sites like Amazon.com and Epinions.com are some other examples of the reputation system pointing towards enormous influence that they wield in our everyday life. This research thus attempts to analyze importance of reputation system in online trading using Agent Based Simulation.

Although it is evident that increased co-operation between online traders is critically important for improved performance of electronic markets, it is very difficult to sustain cooperation among online traders since defect-defect is the only Nash

equilibrium for the online trading when modeled as a Prisoner's Dilemma (Scali, 2006). In absence of any rules and regulations, breakdown of these electronic markets is inevitable. The problem then boils down to setting up a framework of rules and regulations that can ensure the success of such electronic markets. For example, system designers at eBay introduced the concept of feed back score. Feed back score of a particular trader increases by one when he cooperates with his trading partner and gets the reward of positive feedback. Similarly, feedback score of a trader reduces by one if an online trader defects in order to get higher payoff, thus facing the wrath of his trading partner expressed in terms of negative feedback. Each negative feedback brings down the feedback score of a trader and the neutral feedback leaves the score unchanged.

Trust can be considered as the single most important factor that makes the online trading possible simply because an online trader has no other means to verify the quality of the product or genuineness of his trading partner (Dellarocas, 2000). The most common online trading problems being buyer receiving an inferior quality of the product or seller receiving delayed and/or insufficient payment for the products he sold. The problem of seller and/or buyer not abiding by the contractual commitments is so overwhelming that a trader with relatively low feedback score gets quickly branded as untrustworthy. Such a trader is obviously loathed in the online trading community and is unlikely to find a trading partner unless he undertakes an uphill task of improving his feedback rating. Needless to mention, the most attractively priced product offered by a trader with low feedback score usually fails to lure buyers because buyers are more likely to buy the similar product at a higher price from a reputable trader with higher feedback score.

Although the significance of reputation system is obvious, finding answers to some of the what-if scenarios can be very puzzling. For example, what if past mistakes of the traitor are absolved and more importance is given to his/her recent co-operative behavior in order to motivate him/her to co-operate? What if the perceived reliability of reputation system decreases substantially? Would online traders stop co-operating with each other as a result of decreased reliability of reputation system? If yes, would it be possible to quantify the effect of perceived reliability of the reputation system on the co-operative behavior of the online traders?

Impact of these changed conditions on the performance of an electronic market would largely depend on the dynamic behavior of the complex interactions between many interdependent agents. This research tries to answer the aforementioned questions by employing 'Agent Based Simulation Modeling' approach. Agent based modeling is essential for this problem because global effects of locally interacting agents are very useful in strategic planning. The complex behavior of a system as a whole emerges from interactions of large number of online traders in an adaptable system. An adaptable system is characterized by an ability to automatically improve its performance over time in response to what has happened in the past. In other words, complex system actively tries to make the best out of whatever happens.

Answers to these questions could be crucial in predicting the change in the behavior of the online traders in response to the policy changes adopted by e-commerce companies. Understanding behavioral aspects of online traders would certainly be helpful in ensuring respectability and long term stability of the electronic markets.

This thesis thus attempts to establish a trust based reputation system and analyze its effect on the sustainability of mutual cooperation between online traders by taking into account key factors like level of gullibility of online traders and the weight of influence given to their past behavior.

CHAPTER 2

LITERATURE REVIEW

2.1 Relevant Topics of Interest

Since agent based modeling of online trading, in this thesis, is based on Iterated Prisoner's Dilemma (IPD), past research done in the areas of both agent based modeling as well as Iterated Prisoner's Dilemma is discussed in this section. Furthermore, a literature review is also done for evolution of cooperation and adaptive processes. Agents in this thesis use some of the standard IPD strategies to interact with each other so various IPD strategies and agent interactions are reviewed in detail too.

2.2 Agent Based Modeling Vs Equation Based Modeling

Agent Based Modeling is a computational methodology that allows the analyst to create, analyze and experiment with artificial worlds populated by agents that interact in non-trivial ways (Cederman, 1997). Agent Based Modeling helps to fill the gap between formal but restrictive models and wide ranging but imprecise qualitative frameworks.

Agent Based Modeling however, differs significantly from traditional Equation Based Modeling. Various entities in the system are represented by group of agents in Agent Based Modeling. Each agent is a software programs that imitate the behavior of system entities. On the contrary, a system is represented by mathematical equations in Equation Based Modeling and the system behavior is determined by solving them. (Dutta et al, 2006).

Some of the basic differences between the two approaches are summarized in the Table 2.1:

Table 2.1: Differences between Equation Based and Agent Based Modeling

Principle	**EBM**	**ABM**
Building block	Feedback loop connecting Behavioral variables	Individual agents connected by feedback loop
Object of interest	Structure of the system	Agent's rules
Research approach	Deductive: Infer from structure to behavior	Inductive: Infer from individual **agent's** behavior to system Behavior
Development of object of interest over time	Structure is fixed	Agent's rules can be adaptive
Handling of time	Continuous Simulation	Discrete or continuous simulation

As mentioned in Table 2.1, another fundamental difference between Agent Based Modeling and Equation Based Modeling is the granularity of the model focus. Agent Based Modeling relies on the agent interactions at the grass root level, on the contrary Equation Based Modeling tends to make use of system level observables.

2.3 Prisoner's Dilemma

Agents in this research deal with each other using IPD (Iterated Prisoner's Dilemma) strategies. Thus it is imperative to review the past work done in this area. At

the outset, let us first take a quick look at classic prisoner's dilemma game. The prisoner's dilemma game was first formalized by Tucker (Tucker, 1950). It is widely used in modeling problems in Social Science. Socio-economic applications of prisoner's dilemma include: collusion between firms, trade barriers between countries, and public goods problems. The basic Prisoner's dilemma is a two player game. Each player has a choice of either cooperating or defecting. Thus, the two agent prisoner's dilemma game is an abstraction of social situations where each agent is faced with two alternative actions: cooperating, i.e. doing a socially responsible thing and defecting i.e. acting according to self interest regardless of how harmful this might be to other agent. Although each agent is better off defecting regardless of the opponent's choice, the sum of the agents' payoffs is maximized if both agents choose to cooperate, and thus the dilemma. In game theoretic terms, defecting is a dominant strategy of the game and so the defect-defect action combination is the only dominant strategy equilibrium (and therefore also the only Nash equilibrium). On the other hand, social welfare is maximized at cooperate-cooperate action combination; if social welfare is defined to be the equi-weighted sum of the agents' payoffs. Table 2.2 shows a payoff matrix for a two player game, where each agent has two possible actions. Values in parenthesis are used in the experiments of this research. These are the payoffs that can be typically found in the literature.

Table 2.2: Payoff Matrix for the Row Player

		Column Player	
		Cooperate(C)	Defect(D)
Row Player	Cooperate(C)	R (3)	S (0)
	Defect(D)	T (5)	P (1)

Note that the payoff matrix describes a PD game if the following inequalities hold:

$$T > R > P > S$$

and

$$2R > T + S > 2P$$

Negotiations, before, during or after the Prisoner's Dilemma game are not allowed. Agents do not commit to any action, and agents' payoffs are non transferable.

2.4 Iterated Prisoner's Dilemma

Since agents usually deal with each other more than once in real life, social interactions can be modeled more effectively by repeated PD games. This super game of Prisoner's Dilemma game is called the Iterated Prisoner's Dilemma (IPD) game. The IPD game structure captures dilemma between defecting with the hope to get high payoff and the cooperation resulting in smaller individual benefit. In iterated prisoner's

dilemma, an agent's strategy is a result of its own and its opponent's past moves. Unlike pure strategy, an agent chooses its action stochastically from a distribution in a mixed strategy.

2.5 Evolution of Cooperation

In an IPD game, even a selfish agent is forced to cooperate on some of the iterations in order to seek cooperation from its opponent. If the number of iterations of the PD game in an IPD game is known, then the last iteration becomes the standalone PD game. So in the last iteration each agent is motivated to defect, because both agents know that the opponent is going to defect on the last round any way. The backward induction can be carried out all the way to the beginning of the interaction. Thus in some sense it is rational to defect throughout the sequence (Luce and Raiffa, 1957). Thus given these payoffs, it can be easily shown that mutual defection is the only Nash equilibrium (it is also dominant strategy equilibrium). Of course, the intrigue of the prisoner's dilemma is that this unique equilibrium is Pareto inferior to the mutual cooperation outcome. If the basic Prisoner's dilemma is iterated for a finite but unknown number of times, or it is played for an infinite number of times with payoff averaging, then cooperative outcomes can theoretically emerge. In fact the Folk theorem implies that with sufficiently little discounting, any individually rational outcome can be supported as a (sub game perfect) Nash equilibrium. Since fixed horizon IPD games have this characteristic, the agents in this research interact with each other using IPD strategies with an indefinite horizon i.e. the agents do not know how many iterations are still to come.

Deciding the best strategy for iterated prisoner's dilemma game is quite difficult since it calls for consideration of arbitrarily long input histories. A couple of approaches have been used to address this problem.

- Next action is based only on fixed amount of previous moves.
- Some numeric indicator representing entire history of moves is tracked throughout the game.

Two classical examples of the first approach are the pure strategies called Tit-for-Tat (TFT) and PAVLOV. A player using TFT cooperates on the first move and then mirrors its opponent's previous move. Although simple, TFT has been proven to be very successful in evolutionary IPD experiments (Axelrod, 1984). PAVLOV cooperates if and only if the agents chose the same action on the previous move. In evolutionary IPD games with certain random disturbances PAVLOV outperforms TFT (Nowak and Sigmund, 1993). However both TFT and PAVLOV have certain inherent limitations. TFT ignores the older history, and PAVLOV gives an abstraction of the true state but the important details may be lost.

The search for an appropriate way to model the strategic choices of agents has been a central topic in the study of game theory. While a variety of approaches have been used, few of them have explicitly incorporated notions of learning and adaptation. Rubinstein (Rubinstein, 1986) analyzed meta-agents who optimized their selection of strategies constrained by the costs of implementing such strategies. This approach can be developed by modeling the meta-agent's choice through genetic algorithm. As a simple example of this approach co-evolution of strategies in the iterated prisoner's dilemma problem can be analyzed with perfect and imperfect reporting. In imperfect

reporting (Miller 1996), a noise level of X% indicates that X% of the time an opponent's move is reported to be the opposite of what the opponent actually did, while the remainder of the time this move is perfectly transmitted. For this notion of reporting noise to make sense in the context of iterated game, an aggregation of sub-game payoffs is required (otherwise the payoff information will be sufficient to reveal the actual move). However, the possibility of imperfect reporting is not included in this research.

Typically, populations starting with a random sample of strategies go through cycles. Initially, 'Traitors' feed on 'Loyals' in the population. It pays for the Traitors to defect since Loyals fail to retaliate against defections. However, sufficient number of Practical (Tit-For-Tat) players resulting from chance mutations begin playing each other frequently and invade the mutual defectors. Eventually the population moves to mutual cooperation (Axelrod 1984, Nowak and May, 1992). As TFT becomes dominant, cooperative strategies start proliferating by genetic drift, since TFT and similar strategies won't take advantage of them (Boyd and Richardson, 1987). As the proportion of cooperative Loyals increases relative to the number of TFT like strategies, the population becomes more and more susceptible to invasion by Traitors. Eventually the Traitors proliferate, when their large payoffs from the Loyals leads to higher fitnesses than even Practical players. Then the cycle repeats itself, as the average payoff falls in a population of mutual defectors.

2.6 Adaptive Processes

An adaptive process controls how agents adapt or learn over time. There are two ways to model players who learn adaptively while deciding their strategies. The first is imitative approach that allows players to exactly copy the best performing strategies. This is implemented when players decide their strategy by copying the strategy of the most profitable player that is known to them. This approach of imitation is based on learning heuristic. A player tends to copy the strategy of the better performing player because he thinks that it is going to work better than his current, poor performing strategy. Imitation is the commonly studied adaptive process in the iterated prisoner's dilemma literature, either explicitly (Nakamura, Matsuda and Iwasa, 1997) or implicitly via replicator dynamics (Nowak and May, 1992). It is worth taking note that the player might or might not have access to the complete information about the profitability of all the players around them. They are thus required to make the decision depending upon the information that they have. It is assumed that players have information about profitability of all the other players in this research.

The second approach is an innovative approach, whereby players form new programs by combining different parts of existing strategies along with some unique modifications. This approach is derived from genetic algorithm (Holland, 1975). Genetic algorithm is used for solving optimization problems in difficult domains. However, this approach is not used in this research.

2.7 IPD Strategies

The player playing IPD have different options of strategies. For example, Loyal (always cooperate), Traitor (always defect), TFT (Tit for Tat i.e. cooperate on the first move and then mirror the opponent's last move), Anti-TFT (Just opposite of TFT), PAVLOV (Win Stay Lose Switch i.e. to cooperate if and only if the other player chooses the same action on the previous move), Trigger (cooperate till the other player cooperates and always defect thereafter), Punish Twice (cooperate till other player cooperates, if the other player defects, then defect exactly twice irrespective of his moves and then cooperate again), GTFT (Nowak and Sigmund, 1993) i.e. Generous Tit for Tat (like TFT it cooperates after the opponent has cooperated in the previous round but also it cooperates with some probability after opponent has defected) etc.

The players in IPD are often modeled as a Moore Machine which can be described by four elements. The machine consists of a set of internal states. One of these states is designated as the starting state, and serves as the initial state of the machine. Every internal state has associated with it a single strategic action, i.e. in IPD, every state indicates whether the machine will cooperate or defect during the next period. Finally there is a transition function associated with each internal state that determines the next internal state given the reported action of the opponent.

Per Axelrod (Axelrod 1998), since the overall success of a strategy depends on its performance over the mix of others encountered, there is no strategy that is best against all possible populations of others. For example in a world of all Practical Players (TFTs), a single traitor is ruined. It is precisely because the mix of others encountered is so crucial that variation in interacting structures can play a large role in the emergence

of a particular strategy. The dynamics of the system are not directly determined by global proportions of strategy types, but rather by who is meeting who on a local scale, and how the agents adapt to the resulting experience.

This research employs three of the aforementioned strategies: Loyal, TFT and Traitor. It is evident that full cooperation can not be sustained unless an external policy is enforced on the supply chain. Thus policy of 'Track Record Score' (TRS) is introduced in this research to achieve robust cooperation among players.

2.8 Interaction Processes

Different interaction processes can lead to emergence of different strategies among the players. There are several interaction processes which can be used to generate nonrandom interactions among players including –

- Cost of complexity is applied to TFT strategy (Imhof et al, 2005) with the modified payoff matrix, evolutionary oscillations has been observed among all three strategies (AllC, AllD, TFT).
- Players can be more inclined towards interacting with nearby Players ([Nowak and May, 1992], [Oliphant, 1994], [Hoffman and Waring, 1996] and [Cohen et al, 1998]).
- Players resort to tag mediated partner selection (Riolo, 1998) Players in these models are always in search of acceptable partners to play the IPD.
- Players remember the past moves of other players and can bias their interactions based on behavior of particular player in the past ([Stanley et al,

1994]). For example, players might just ostracize those who have ever defected against anyone.

- When a defector gets punished by another player adopting altruistic punishment, then both defector as well as punisher incurs some cost (Boyd et al, 2003). With the modified payoff structure, altruistic punishment has been shown to evolve in populations engaged in one time anonymous interactions.

The aforementioned approaches do report sustenance of cooperation in some cases. For example, Hauk (Hauk, 1996) concludes that due to partner selection cooperative behavior is immune to invading mutants. However, these approaches may not be always applicable in the supply chain context, for example: a specific spatial topology may promote cooperation but it might be impractical for the arrangement of players in the supply chain context. Social tags may prove to be useful to make players cooperate with each other but the concept of 'tagging' individuals may not be applicable in every scenario, similarly the luxury of choosing the partner may not be always available to the players because of other business constraints. Thus this research attempts to use some practically applicable concepts such at Track Record Score (TRS) to promote the cooperation among players. The concept of Track Record Score is explained in detail in the following section.

CHAPTER 3

AGENT BASED SIMULATION

3.1 Agent Based Modeling

An Agent Based Simulation Modeling approach is employed in this thesis to model the electronic commerce situation described in the Motivation section. Although mathematical formalization is more popular among researchers, application of mathematical modeling in social sciences usually involves numerical treatment of various kinds of complex differential equations. Needless to mention, use of mathematical equations typically leads to a very complex solution. On the contrary, graphical representation available in the agent based simulation is much easier to understand and thus assists in the better analysis of the problem.

An Agent, as depicted in Figure 3.1 below, is nothing but a computer program that is designed to imitate human behavior. Every agent is autonomous and has his own perception of the surrounding world. An agent's behavior is usually defined by set of simple rules. Within the framework of those rules, an agent independently decides how to interact with other agents and tries to achieve his personal goal in the least amount of time. Thus the inherent uncertainty characterized by all the multi-agent systems (also called complex adaptive systems) can be attributed to the fact that future actions of other agents are essentially not known, just as in real life.

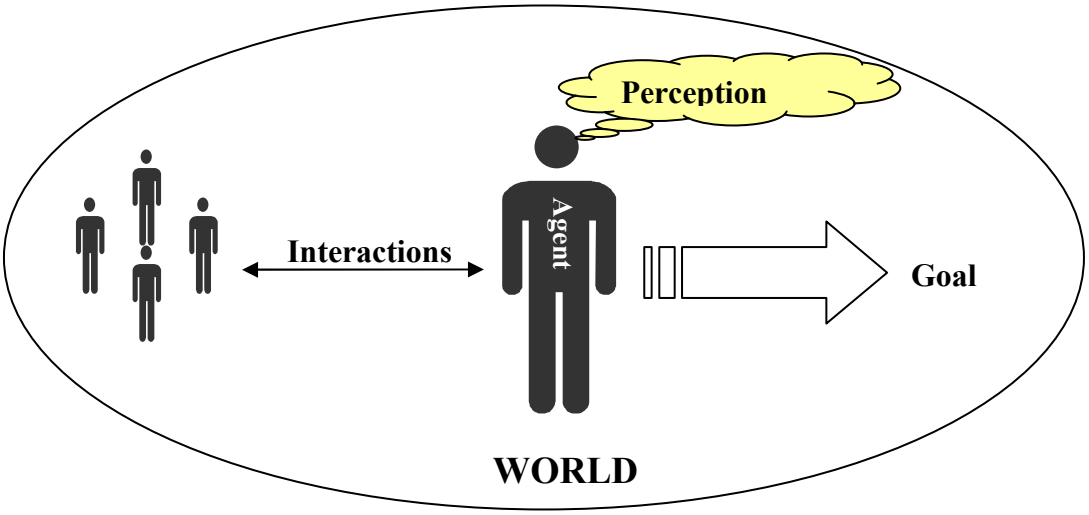


Figure 3.1: Agent in the Simulation Model

Agent based simulation is based on a 'Bottoms Up' approach unlike the 'Top Down' approach of Discrete Event Simulation. In other words, a global structure in agent based simulation emerges from interactions among agents at the local level. Once the initial conditions are set by the modeler, agents start interacting with each other and the agent based model evolves over time without further outside assistance and/or intervention.

3.2 Repast: Software Framework for Agent Based Simulation

RePast is used for developing agent based model in this thesis. NetLogo, MASON and SWARM are some other popular platforms available for building agent based models and are reviewed later in this chapter. RePast is developed by the University of Chicago's Social Science Research Computing Department. RePast provides a library of objects for creating and running models, displaying graphics and

charts, and collecting output data from an agent based simulation. Agent based models in RePast typically contain a set of agents. Each agent is characterized with a unique behavior. RePast represents most of the key elements of agent-based simulation as a Java class or classes. These classes create a framework for building agent-based simulations.

Conceptually, RePast is similar to Discrete Event Simulation Software like AutoMod. Events are scheduled to happen at a specific time 'tick'. A tick is assumed to be quantum unit of time and is used to sequence the execution of events. RePast has a scheduling mechanism that is capable of creating dynamic schedules such that the execution of an event can itself schedule other events to be executed in the future. This schedule not only controls the execution of agent behaviors, but also actions within the model itself, such as updating the display, recording data, and so forth. Scheduling can be automated or manually implemented by the modeler (Collier 2000).

RePast uses Java since it eliminates the kind of memory leaks associated with C, C++, and Objective-C, a particular problem for long-running simulations. Furthermore, Java is well documented and its cross platform design enables it to be easily installed on a variety of platforms. Java allows the programmer to organize his or her code into packages. The package system is used to partition code into logical units. Agent based simulation in Java is depicted in Figure 3.2 below.

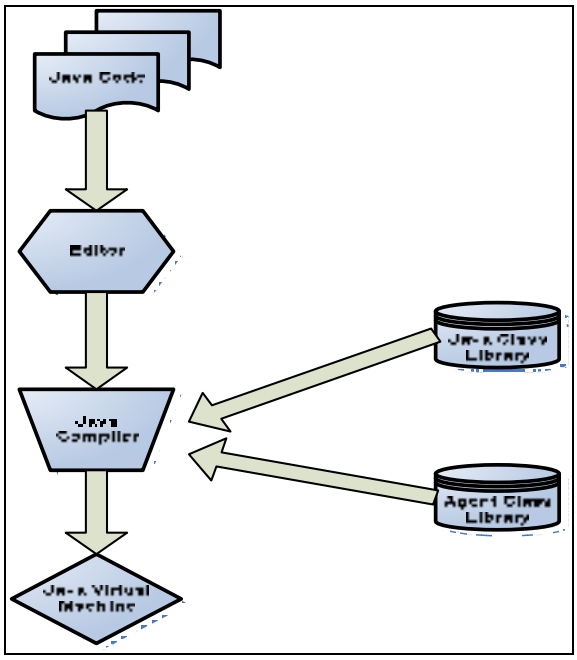


Figure 3.2: Agent Based Simulation in Java

RePast offers performance comparable to similar frameworks and will only get faster with the continual improvement of Java Virtual Machines. As a result of these goals, RePast is robust, extensible, and fairly easy to use.

RePast framework is explained in Figure 3.3 below. RePast can be started by double clicking the 'repast.jar' file in the 'lib' folder. After starting, RePast displays the Control Tool Bar. Model library can be browsed by clicking on the 'Folder Icon' on tool bar. The selected model then can be loaded by clicking on 'Load'. RePast then loads the model and Actions Tab for managing parameters is displayed. After setting the run time parameters, 'Single Step' or 'Forward' button can be clicked to run the model and 'Pause' can be clicked to pause the model.

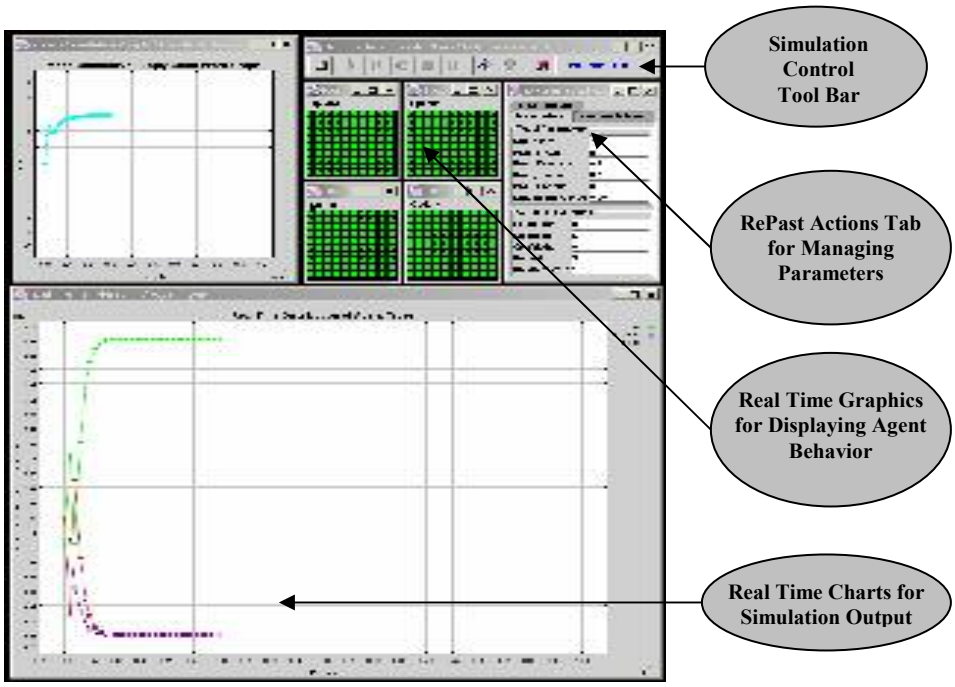


Figure 3.3: RePast Framework

Typical steps involved in agent based modeling with RePast are explained in the Figure 3.4 below.

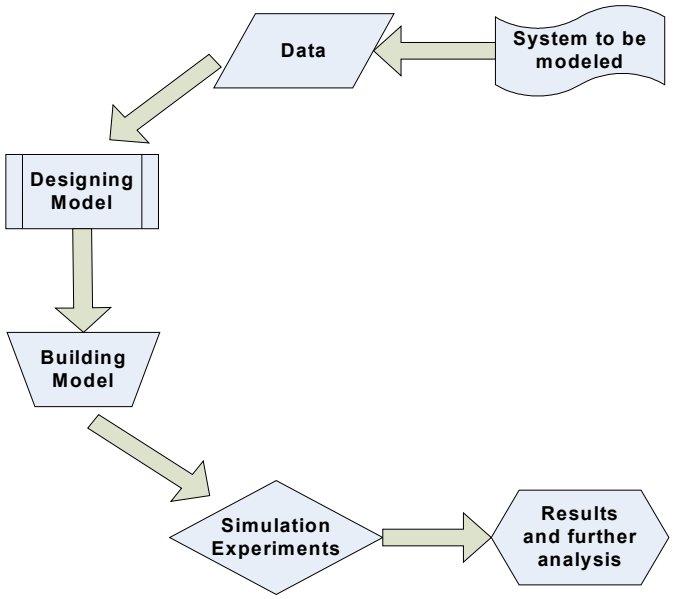


Figure 3.4: Steps in Agent Based Simulation

An agent-based simulation modeling usually has two steps. The first step prepares the model for running, and the second step is the actual running of the model. The running of the simulation is divided into time 'ticks'. Actions occur at every tick using the results of the previous tick. For example, to build an agent based model for a prisoner's dilemma with two players, setup would create the two players and provide each with an initial strategy (Co-operate, Defect, Tit-for-tat, etc.). Each tick or time step, players would play the game and decide to co-operate or defect depending on their strategy. The strategy adopted by players will in turn depend on the results of previous play. As a next step of building an agent based model, placeholders in the SimpleModel class are used to describe what happens during setup and what occurs during every tick.

As described in the RePast tutorial, agent based simulation models in RePast typically have at least two classes. An agent class describes the behavior of agents e.g. co-operate or defect while interacting with other agents and a model class coordinates running the model. SimpleModel class in the RePast library can be used to inherit the model class. For example, class DSCM (Decentralized Supply Chain Model) is inherited from SimpleModel class below:

```
import uchicago.src.sim.engine.*;
public class DSCM extends SimpleModel {
• • •
• • •
}
```

Please refer to the Appendix A for the entire source code.

While building agent based simulation models, it is important to describe the setup of the model, and the actions executed during every time step. ‘Setup’ and ‘BuildModel’ methods provided by SimpleModel class can be re-described in the model to do the custom setup. For example:

```
public void setup() {
    super.setup();
    name = "Agent Based Simulation Model for Online Trading: Version # 8";
    grid_Side = 20;
    prob_Loyal = 0.1;     prob_Practical = 0.1;     prob_Traitor = 0.8;
   . . .
   . . .
}
 public void buildModel() {
    agentList = new ArrayList();
    agentPopulation = grid_Side * grid_Side;
    num = new int[3];
    num[Loyal] = (int) (agentPopulation * prob_Loyal);
    num[Practical] = (int) (agentPopulation * prob_Practical);
. . .
. . .
  }
```

‘Setup’ and ‘BuildModel’ methods are used to customize the SimpleModel class. ‘Super.setup’ method allows SimpleModel class to do the necessary setup on its own so that the player strategies can be set. Since player strategies may change from their default values either through user interaction or during the course of a previous simulation run, ‘setup’ method is used to set the model variables back to their default values. Setup method is called whenever the setup button is clicked or when the simulation is first started.

Once the setup method resets the simulation model, ‘buildModel’ method is used to create the various objects that the model uses. Agents are thus created here and added to the agentList array, which acts a master list of agents. Per the order of execution in RePast, the setup method is called first followed by buildModel. Nonetheless as explained above, the setup method is also called whenever the setup button is pressed as well as when the simulation model starts running. The ‘buildModel’ method is called when run, step or initialize buttons are pressed thus allowing the user to change the parameters through the graphical user interface.

After completing setup method, actions occurring during each time-step of the simulation are defined. ‘preStep, ‘step’ and ‘postStep’ are the three methods offered by SimpleModel class for this purpose. During each time tick, preStep is executed in the beginning, then step method is executed and lastly postStep is executed. The behavior of agents in step method is thus separated from any necessary pre- or post- processing. The agent based model in this research uses only step method since pre-processing or post-processing is not required. The example of Step method is as below:

```
public void step() {
   interactions();
   StrategyDecider();
   Summery();
  }
```

Step method calls the ‘interactions’ method. The interactions method then iterates through all the agents and calls on them the ‘BusinessDealing’ method that facilitates the interaction between agents. The step method will be executed at each time step during the course of model run.

The relationship between the various simulation control tool bar buttons and the execution of methods is as follows. A click on setup button executes the code in the setup method. Similarly, the code in buildModel method is executed after clicking the initialize button. The click on step button triggers execution of the code in buildModel method. The preStep, Step, and postStep methods are executed in that sequence after that. When the start button is clicked, buildModel is executed. The preStep, Step, and postStep sequence is executed repeatedly until the user clicks the stop or pause button.

Model parameters are defined by accessor methods. These methods begin with ‘get’ and ‘set’. Example of accessor methods is as below:

```
public double getprob_Traitor() {
   return prob_Traitor;
  }

  public void setprob_Traitor(double p) {
```

```
    prob_Traitor = p;
  }
```

In the ‘Constructor’ of the model, RePast is made aware of the parameter by including the parameter name in the string array of parameters called ‘params’. The params variable is provided by SimpleModel class. It is an array containing the names of the model parameters. For example:

```
public void setup() {
    super.setup();
    params = new String[] {"grid_Side","prob_Loyal", "prob_Practical",
                "prob_Traitor", "prob_Change", "smoothing_Const",
                "MaxSearches_PerDay"};
• • •
• • •
  }
```

After creating parameter ‘accessor’ methods and populating the ‘param’ array with model parameter names, the parameters are displayed in the parameter pane when the simulation is run. These parameters are set to the values returned by their get accessor methods. The parameter can set by entering a new value in the parameters text box as shown in Figure 3.3 above. The newly entered value then becomes the argument to the parameter's set ‘accessor’ method.

Agents in this thesis represent online traders. Business interactions between online traders are modeled as Iterated Prisoner’s Dilemma (IPD). Traders will have the

option of choosing any one of the IPD strategies. The strategy space includes three IPD strategies:

- Loyal: To cooperate every time irrespective of the strategy adopted by the opponent
- Practical: Tit for Tat i.e. to cooperate on the first move and then mirror opponent's last move
- Traitor: To defect every time irrespective of the strategy adopted by the opponent

Loyal, Practical and Traitor are also referred to as AllC (always cooperate) TFT (Tit for Tat) and AllD (always defect) respectively in game theory literature. Typical IPD payoff values of 3, 0, 5 and 1 are used and this payoff structure is strictly adhered to throughout the simulation.

NetLogo, Swarm and Mason are some of the popular alternatives for RePast. According to Steven et al, (2006) each of these tools has certain advantages and drawbacks. NetLogo is very easy to use and has excellent documentation. However, NetLogo requires all the code to be in one file. It thus provides less organizational discipline as compared to Java and thus is less suitable for large size models. Also NetLogo does not provide access to its algorithms and thus restricts the flexibility for a modeler. Original Swarm developed in Objective C is a pioneer of Agent based platforms. It is a very stable tool and offers a fairly complete set of library classes to the modeler. The drawbacks of Swarm include the unpopularity of the Objective C language and poor availability of documentation and tutorials. Swarm is also available in Java, however the source code is still in Objective C and thus debugging run time

errors becomes very difficult. Unlike Swarm, MASON offers relatively fewer tools but is the fastest agent based platform and thus an ideal choice for computationally intensive models. MASON is however not very user friendly and is more geared toward experienced modelers.

CHAPTER 4

PROBLEM DESCRIPTION

The purpose of this research is to understand the effect of changes in the trust based reputation system on the cooperation levels of online traders. An online trader almost always makes his business decision based on trust because the current mechanism of customer to customer online transaction provides an added incentive for cheating. For example, despite receiving full payment as per the agreed price of the advertised product, a seller might send an inferior quality product to the buyer or still worse might not send anything at all. Similarly, despite receiving the satisfactory product a buyer might refuse to make appropriate payment to the seller. In the absence of any reputation system, it becomes difficult to spot such 'Traitors' because of the enormous population of online traders. Such impunity for the past misbehavior coupled with the abundant availability of new trading partners thus enables Traitors to thrive at the expense of honest and co-operative traders and eventually jeopardizes the very existence of electronic markets.

It is evident that sustained cooperation among online traders is absolutely essential for ensuring the success of electronic markets. Thus exploring the underlying relationship between reputation system and cooperation level is of great practical significance.

In this research, agent based simulation methodology has been used to model an electronic market place with 400 traders. The number of traders is limited to 400 because the computational load of an agent based simulation model increases

exponentially with the number of traders. However, it was observed that the effect of higher numbers of traders on simulation results is statistically insignificant.

Traders can buy as well as sell the products depending on their requirement. Every trader tries to find suitable business partners online and does business with them. Out of the total trader population of 400, 30%, i.e. 120 traders, are assumed to be 'Loyals' (always co-operate), 40%, i.e. 160 traders, are assumed to be 'Traitors' (always defect) and the remaining 30% are assumed to be 'Practical' Traders' (tit for tat). However this initial mix can quickly change once the simulation begins and traders start adopting the strategy that they perceive to be the most profitable one. A Trader's payoff depends not only on the IPD strategy that he adopts but also on the strategy that his business partner uses.

Strategies adopted by traders evolve as they start interacting with each other. As explained earlier in the Section 3, the strategy space available to traders comprises of three strategies, namely: Loyal i.e. to cooperate every time irrespective of the strategy adopted by the opponent, Practical or Tit for Tat i.e. to cooperate on the first move and then mirror opponent's last move and Traitor i.e. to defect every time irrespective of the strategy adopted by the opponent. During the course of simulation model run, traders can change the strategy that has been randomly assigned to them at the start of the run. This adaptation of agents is based on the principle of imitation i.e. agents are likely to imitate the strategy of the most profitable agent. Whether or not to change the strategy is decided solely by an agent. Every agent is assumed to be a unique individual with certain level of gullibility. The more gullible the agent is, the higher the probability of him imitating the strategy of the most profitable agent. Thus, as the simulation run

progresses, agents keep reviewing their strategies and decide whether or not to imitate the strategy of the most successful agent resulting in evolution of different strategies.

Traders will be able to update their strategy by imitating the strategy of the most profitable player with certain probability. The probability of imitation represents the skeptic nature of human beings. Some traders might not be willing to imitate their more successful counterparts because of the lack of trust. Thus it is assumed in this research that the trader population is more trusting if the probability of imitation is higher. For example, a more trusting trader would easily get influenced by the strategy adopted by the profitable traders and might end up imitating their strategy. On the contrary, a less trusting trader would typically be more skeptical. This level of skepticism, which is unique to each individual trader, is captured by a randomly assigned 'Probability of Imitation'.

During the course of the simulation, traders try to buy or sell the products by making a business proposition to the online traders who they think are appropriate. The trader, who successfully searches a business prospect online and makes a business proposal, would need a high Track Record Score to make sure that the deal materializes. His business proposal is likely to get rejected by the prospective business partner if he has a lower Track Record Score raising serious concern about his trustworthiness.

As explained in the previous section, Track Record Score (TRS) is used as a reputation based system designed to make sure that Traitors would have to pay dearly in terms of their reduced market reputation. TRS enables honest traders to build their market reputation by co-operating with other online traders and honoring the contractual terms. Such a trader would be able to earn a very high Track Record Score. Traders

would obviously prefer to deal with only those traders who they feel are trustworthy. An excellent Track Record Score will enable traders to win the confidence of other cooperating traders.

Each trader is allowed to decide his own strategy. He can either improve his TRS by cooperating or choose to earn more short term profit by defecting and thus sacrificing his TRS. It is worth taking note that TRS does NOT alter the Payoff Matrix. However, it does affect the chances of a trader getting accepted by other traders for an online trade. Thus traders who choose to defect for the short term gains will have to do so at the cost of their own market reputation i.e. Track Record Score. Each trader will receive an initial TRS of 100. Upon the conclusion of a deal between two agents, their scores are updated. The mechanism for updating scores is as follows:

Case 1: None of the agent adopted 'Always Defect'

- Both the agents get 100 points each

Case 2: At least one of the agents adopt 'Always Defect'

- Agent gets 100 points if he adopts 'Always Co-operate'
- Agent gets 0 points if he has adopted 'Tit for Tat'
- Agent gets –100 points if he has adopted 'Always Defect'

This Track Record Score is conceptually similar to the Feed Back Score system implemented by eBay. However, unlike this Track Record Score system, eBay does not use exponential smoothing for scaling feedback scores. Furthermore, although eBay investigates complaints of misbehavior and prevents monitory losses of honest and co-operative traders, it does not make any effort to cancel the reduction in the feedback

score caused by dishonest feedback. The Track Record Score system assumes investigations are undertaken immediately to nullify the effect of imperfect reporting of the trader's behavior on the Track Record Score. In other words, the possibility of imperfect reporting is not considered in this research.

Following mathematical equation is used for scaling track record scores using exponential smoothing:

New Track Record Score = A * Old Score + (1- A) * Current Score

- Where A is a smoothing constant

For example:

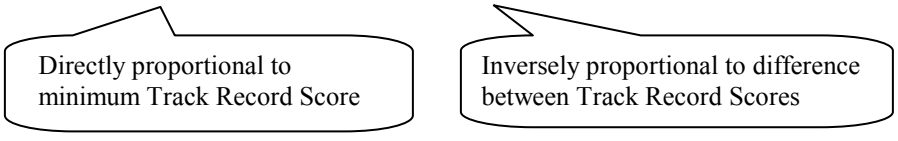
If the Smoothing Constant is 0.8 then:

New Track Record Score = 0.8 * Old Score + (1 - 0.8) * Current Score

Thus the higher Smoothing constant signifies that more importance is given to the old score of an agent.

The probability of two agents agreeing to do business with each other is weighted equally on 'minimum of the two agents' scores' & 'Difference between Agents' scores'. For example, if the track record scores of agents X and Y are 40 and 70 respectively, then the probability that X and Y will agree to do business with each other is:

0.5/100 * [40 + (100- (70-40))]



While calculating the probability of two agents agreeing to trade with each other, a 50% weight is given to the 'minimum of the two agents' scores' because it is assumed

that the trader with the lower track record score is likely to get rejected by the trader with the higher track record score. Similarly the rest of the 50% weight is given to the "difference between agents' scores" because it is assumed that two agents would probably agree to trade with each other if the difference in their Track Record Scores is not high. If one of the traders has a substantially higher Track Record Score then he would not be interested in a trade proposal from his counterpart. It is assumed that a trader with a higher Track Record Score would always want to deal with traders who have equal or higher scores than him. On a similar note, two traders, each with low track record scores, are likely to trade with each other since they won't probably be accepted by traders with higher track record scores anyway.

Although it is assumed in this research that a trader with a high Track Record Score is more likely to reject a business proposal from a trader with a low Track Record Score, one could argue that a trader with a lower Track Record Score can lure the more reputable traders by offering attractive prices. However, in order to maintain the applicability of Prisoner's Dilemma to this research, it is assumed that traders will not change the price structure.

Upon completion of the trading between two agents, an agent's payoff will be added to the cumulative market profit if the player has cooperated. If the player has defected then his payoff will be deducted from cumulative market profit. Mean Trader Profit will be thus calculated using the following equation:

Mean Trader Profit = (Cumulative Market Profit) / (Trader Population)

A higher Mean Trader Profit indicates that traders are co-operating with each other since the IPD payoff structure is used to model agent interactions. Mean Trader profit will be used as a Performance Metric for simulation experiments since the main objective of this research is to understand the effect of changes in the trust based reputation system on the cooperation levels of online traders.

The flow chart of the agent based model discussed above is depicted in the Figure 4.1 below:

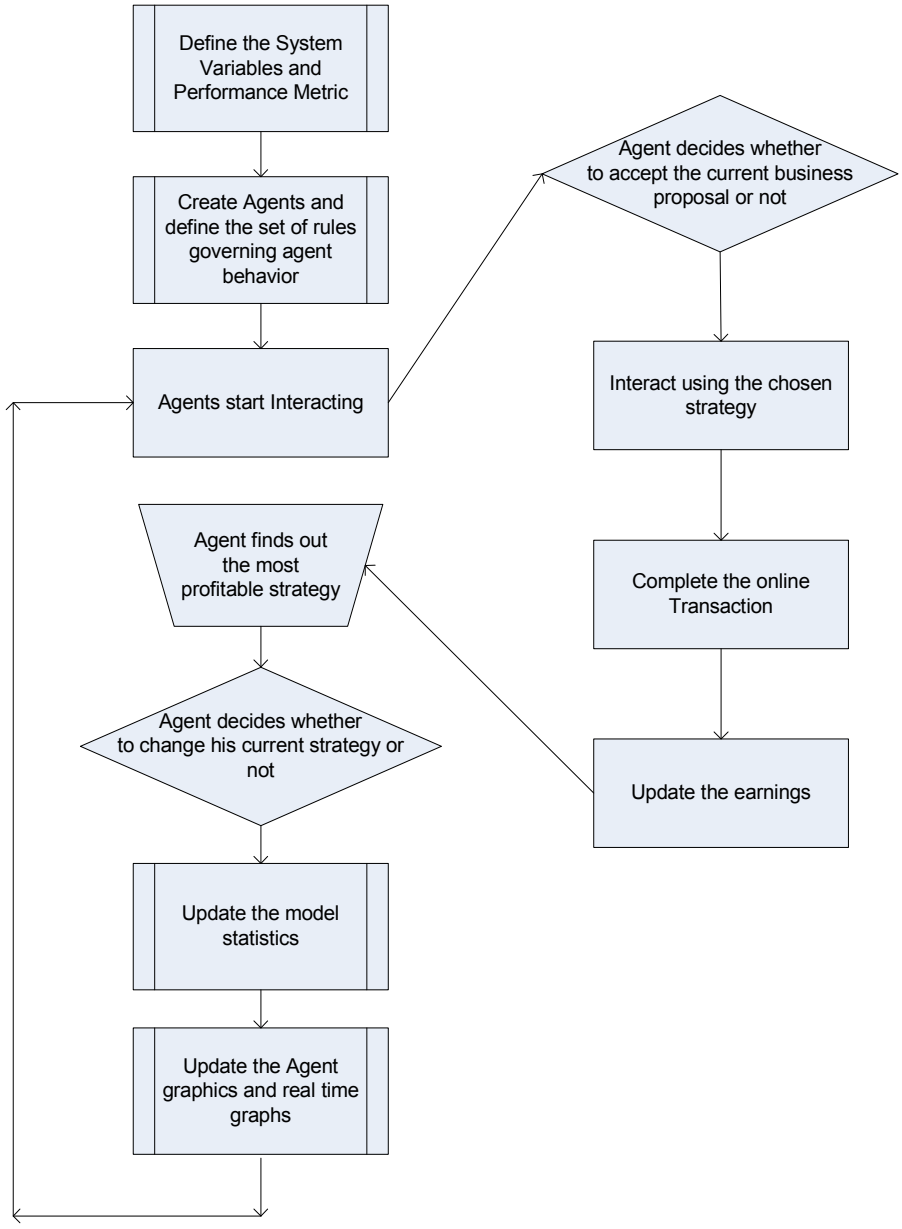


Figure 4.1: Agent Based Simulation Model Flow Chart

As per the first step in the Figure 4.1 above, the Mean Trader Profit was selected as the performance metric. Smoothing Constant and Probability of Imitation were selected as the key system variables.

As a next step, agents were created in the model by employing the 'buildModel' method recommended by the standard RePast procedure described in Chapter 3 above.

All the agents were given a random strategy at the beginning of the model. Once the simulation run begins, using 'interactions' method, agents search for their appropriate partner and make a business proposal. However, whether or not the two agents interact would depend on their Track Record Scores. As explained earlier in this section, the probability of the agents interacting would be directly proportional to the minimum of the Track Record Scores of two agents and inversely proportional to difference between their Track Record Scores.

Agents use the 'beginDealing' method to interact with their business partners. During interaction, each agent employs his or her premeditated strategy and chooses to either co-operate or defect. The defection would result in lowering their TRS and cooperation would result in increasing their TRS as explained earlier in this section. The impact of agent behavior on TRS depends on the 'Smoothing Constant' parameter. The 'profitSum' method keeps track of the profit or loss made by agents during these interactions. Since the standard Iterated Prisonner's Dilemma (IPD) payoff structure used in this thesis, agents would make a profit only when neither of the agents uses the 'Defection' strategy. The payoff structure is explained in the Table 2.2.

After the start of agent interactions, the 'StrategyDecider' method provides all the agents with an opportunity to imitate the strategy that they perceive to be the most profitable one. Whether or not the agents would imitate the strategy of the most profitable player would depend on the parameter 'Probability of Imitation'.

Figure 4.2 below depicts the effect of key system variables 'Smoothing Constant' and 'Probability of Imitation' on the performance metric 'Mean Trader Profit'. It can be seen in Figure 4.2 that cooperation can be sustained when the

smoothing constant is set to 0.8 (i.e. 80% weightage is given to the past behavior of agents while calculating Track Record Score) and probability of imitation is set to 0.4 (i.e. the probability of agents imitating the strategy of other successful agents is 40%). Since agents are somewhat skeptical in this case they did not copy the Traitors during the initial periods when Traitors were making more profit. Eventually the Traitors started losing since their Track Record Scores reduced substantially and agents with higher Track Record Score refused to do business with them. Cooperation thus is sustained in the agent population resulting in the approximate Mean Trader Profit of 3.

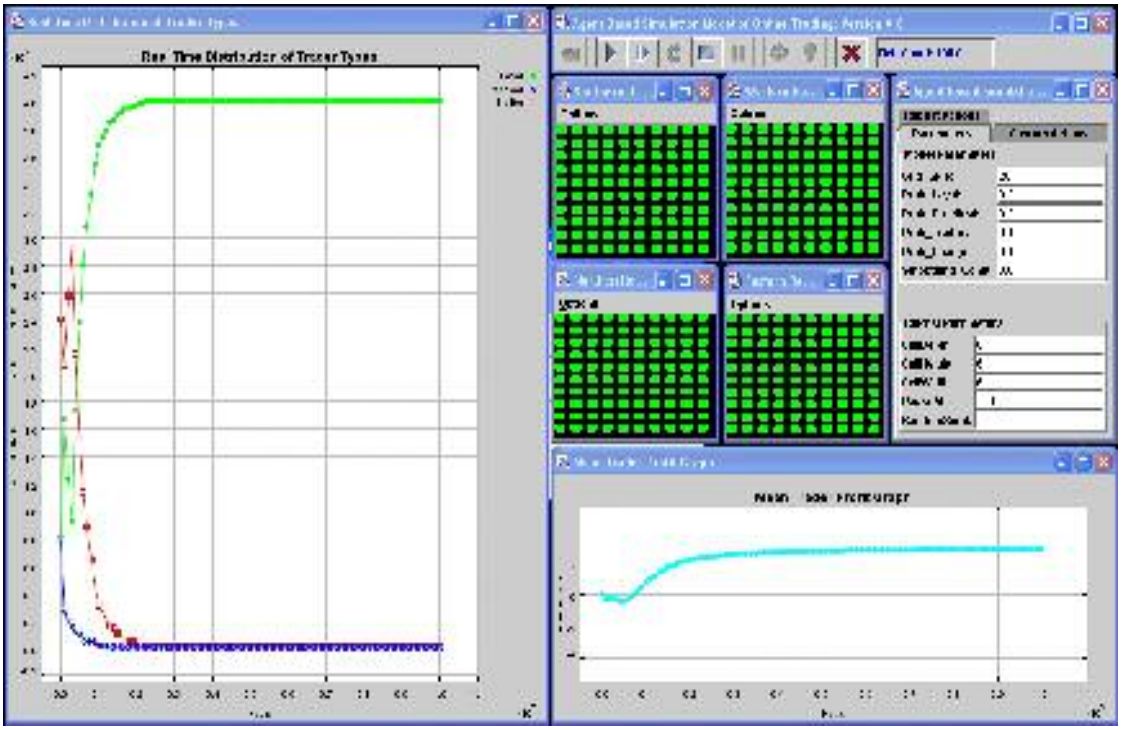


Figure 4.2: Snapshot of Simulation Model with Probability of Imitation = 0.4

Figure 4.3 below depicts the effect of having more gullible agents in the population i.e. increasing probability of imitation without changing smoothing constant. Since agents are more gullible they are more likely to imitate Traitors during initial periods when Traitors are making more profit at the cost of their Track Record Scores. This triggers the frenzy among agents and soon Traitors start proliferating. The vicious circle eventually results in wiping out the remaining cooperation and thus resulting in approximate Mean Trader Profit of -1.

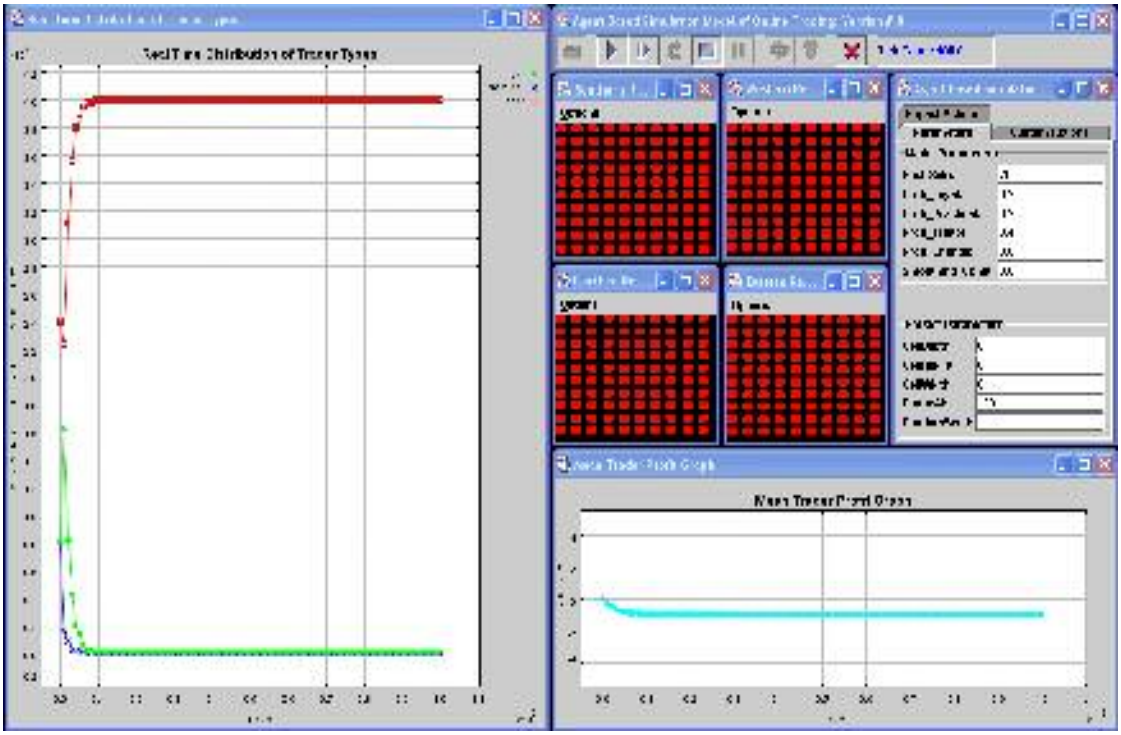


Figure 4.3: Snapshot of Simulation Model with Probability of Imitation = 0.8

It is clear from Figure 4.3 above that high values of the Smoothing Constant result in zero cooperation among agent population characterized by high level of gullibility. One of the solutions to this problem could be to reduce the value of the Smoothing Constant.

This will ensure that the current behavior of agents gets more weightage than the past behavior thus bringing down Track Record Scores of traitors relatively quickly. Traitors will thus find it difficult to feed on gullible co-operators since agents with higher Track Record Scores are more likely to refuse the business proposal from agents with lower Track Record Scores. This will prevent the snowball effect of traitor proliferation initially and allow co-operators to earn enough profit not only to survive but also to prosper. Cooperation thus gets sustained as a depicted by Figure 4.4.

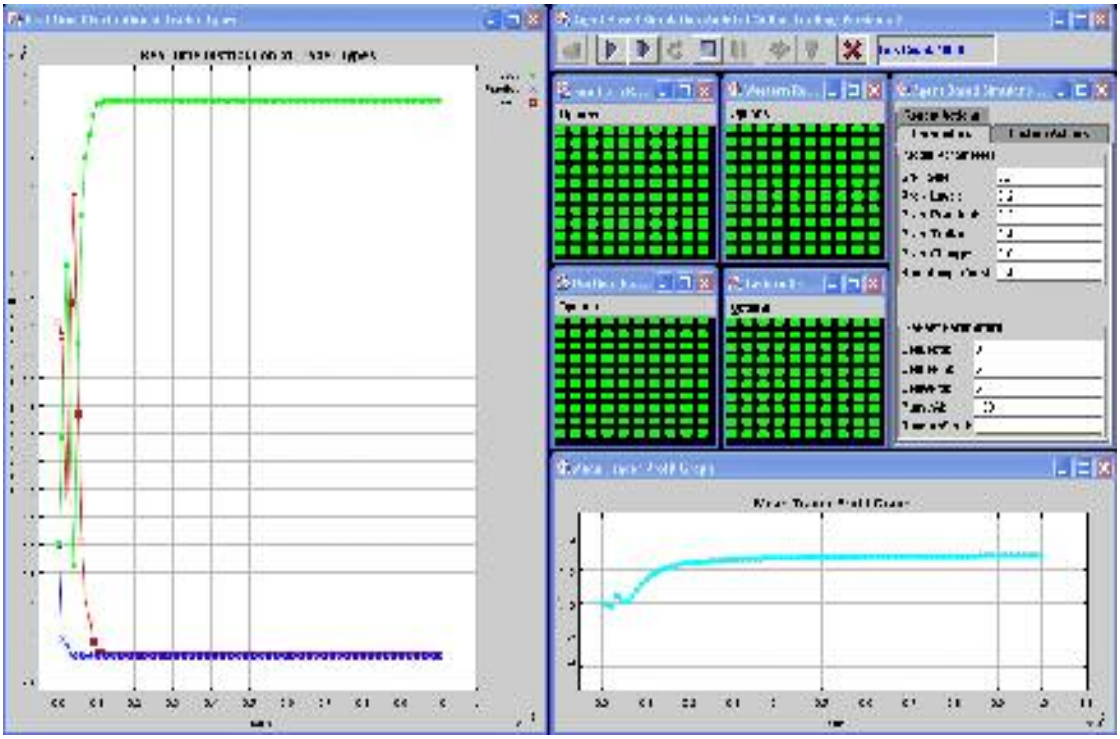


Figure 4.4: Snapshot of Simulation Model with Smoothing Constant = 0.4

It is critically important for a system designer to test the effect of different factors on the level of cooperation among agents. Agent based methodology can be effectively used for such analysis.

Although the trust based system is meant to penalize agents for their dishonest behavior, it would be unjust to punish an agent forever even if he is willing to behave responsibly. It would be tantamount to refusing rehabilitation opportunities to the former criminals. The obvious question that follows is how much weightage should be given to the past as well as recent behavior of the agents while calculating their Track Record Score. Unfortunately, it is impossible to find an elixir to such a problem, simply because each solution is going to be population specific. As the population mix changes, the solution will have to be reviewed too. For example, the trust based system designed for the predominantly skeptic agent population will not prove to be an ideal solution for the population filled with mostly gullible agents.

This thesis thus attempts to analyze changes in the level of cooperation as weightage given to the past behavior of agents (i.e. 'Smoothing Constant) is gradually reduced and more weightage is given to their current behavior. This analysis is done for a variety of population mixes; ranging from the one that is predominantly skeptic (i.e. 'Probability of Imitation = 0.2) to the one that is mostly gullible (i.e. 'Probability of Imitation = 0.95).

CHAPTER 5

RESULTS

The main objective of the agent based model in this research is to analyze the factors that influence the co-operative behavior of agents. As a first step, the model was run without the Tract Record Score system and validated using the results of the previous research. Per Axelrod et al, 1998, when imitation is used as an adaptive process, cooperation can be sustained with Tit For Tat strategy outshining Always Defect as the most popular strategy, if players interact with only with their von Neumann neighbors and the neighbors are not changed during the course of simulation run. In other words, agents will be able to choose the strategy but not the neighbors with whom they will be interacting. Furthermore Axelrod et al (1998) state that cooperation can not be sustained if players are allowed to interact randomly with each other. Similar results were observed when the base model without Track Record System was run under the two conditions mentioned above. It can be seen in Figure, 5.1 that Tit for Tat emerges as the most successful strategy when agents interact only with their neighbors. Figure 5.2 indicates that Always Defect emerges as the winner if agents are allowed to interact randomly with each other.

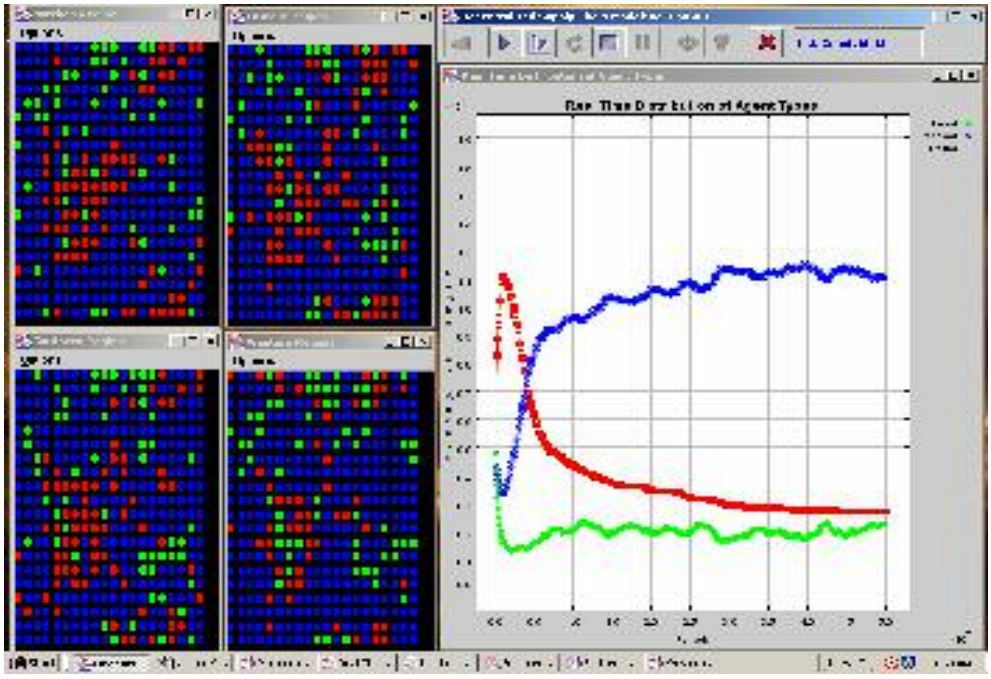


Figure 5.1: Agents Interact Only With the Neighbors

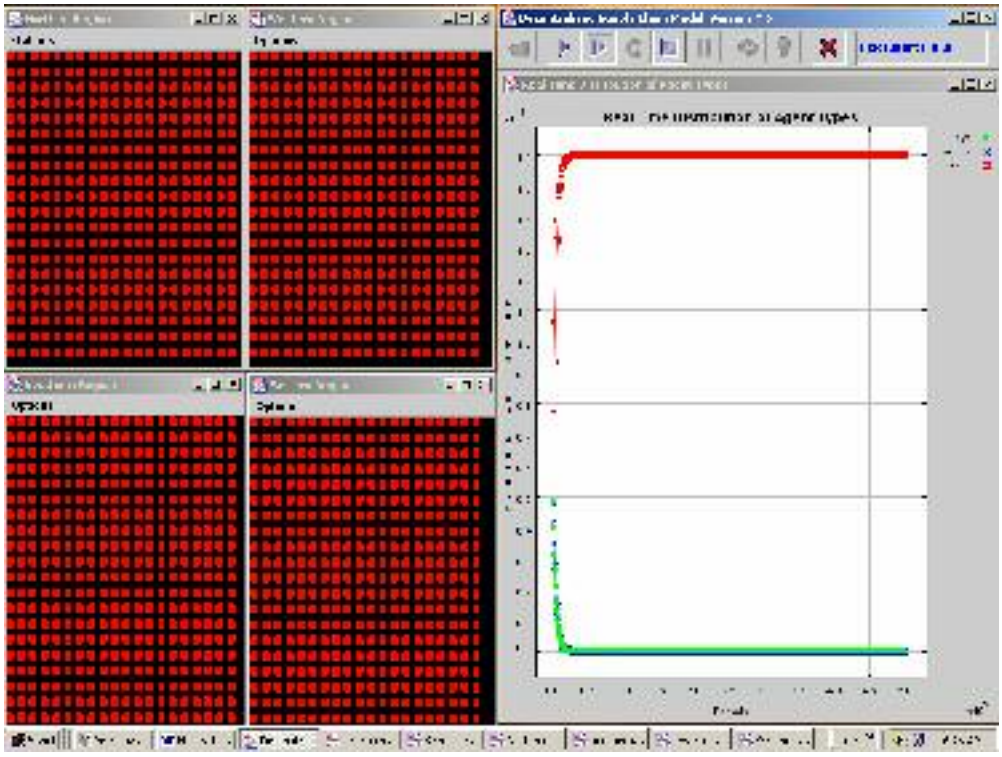


Figure 5.2: Agents Interact Randomly with Each Other

As a next step, Track Record Score was introduced in this research to establish a reputation system capable of promoting cooperation among agents with different levels of gullibility. The next obvious step is to design the 'what-if' scenarios. Designing the 'what-if' scenarios is one of the most important steps in the agent based simulation modeling methodology. Hence the 'Smoothing Constant' (i.e. weightage given to the past behavior of agents) and 'Probability of Imitation' (i.e. level of gullibility of agents) were identified as the critical factors for designing the what-if scenarios. 'Percentage of honest traders in the population mix' and 'payoff structure for agents' were also identified as the probable factors for what-if scenarios. However, it was observed that their impact on the level of cooperation is statistically insignificant as discussed later in this chapter.

As stated earlier, the Track Record Score depends on the current behavior of an agent and how the agent has behaved in the past. The following equation is used to calculate the Track Record Score (TRS).

New Track Record Score = A * Old Score + (1- A) * Current Score

- Where A is a smoothing constant

Thus it is evident that any change in the value of the smoothing constant is bound to affect the Track Record Score and in turn the level of cooperation among agents. Agents in this model can imitate the strategy of other agents who they perceive to be more successful. The more gullible an agent is the higher the probability is that he would be influenced by the strategy of the seemingly more successful agents. On the contrary, driven by inherent mistrust, a skeptic agent would be more apprehensive in

imitating the strategies of other agents. Thus the level of gullibility of agents would also be an important factor that can influence the level of cooperation among agents.

Needless to mention, the agent based simulation model developed in this research can be used as a tool to answer a few basic questions in the mind of a system designer tasked to improve the Trust Based Reputation System for achieving higher cooperation levels among online traders. Some of those questions could be, how would the change in the gullibility levels of traders affect the cooperation levels? For the given gullibility level of a trader population, exactly how much emphasis should be given on the past behavior of agents while calculating the Track Record Score? At what point would the right answer not be right any more? In other words, for the given trader population, what are those critical points where any further change in the gullibility levels and/or change in the emphasis given on the past performance of agents while calculating Track Record Score would result in proliferation of traitors? The what-if scenarios designed in this research are aimed at answering exactly these questions.

Lack of in-depth statistical analysis often results into unreliable simulation results. Per Axelrod (Axelrod, 1997), the analysis of a single run can be misleading. In order to determine whether the conclusions from a given run are typical it is necessary to do several simulation runs using identical parameters (using different random number seeds). While it may be sufficient to describe detailed history from a single run, it is also necessary to do statistical analysis of a whole set of runs to determine whether the inferences drawn from the illustrative history are really well founded. The ability to do this is one major advantage of simulation: the researcher can rerun history to see whether the particular patterns observed in a single run are idiosyncratic or typical.

The Mean Trader Profit is calculated by making 6 replications of each of the 9 experiments fed with antithetic variates of random seeds to minimize the variance. So the total number of replications carried out is 54. Since all the experiments are of terminating type a warm up period is not required. Run length of 100 is chosen because all the experiments hit the steady state well before that.

Table 5.1 below summarizes results for the different what-if scenarios:

Table 5.1: Summary of What-if Scenarios

1. Probability of Change = 0.60 Smoothing Constant = 0.90 Mean Trader's Profit = -0.969 Standard Deviation = 0.02 95% C. I. for Mean = (-0.952,, -0.985)	2. Probability of Change = 0.40 Smoothing Constant = 0.90 Mean Trader's Profit = 2.381 Standard Deviation = 0.30 95% C. I. for Mean = (2.621, 2.141)	3. Probability of Change = 0.20 Smoothing Constant = 0.90 Mean Trader's Profit = 2.690 Standard Deviation = 0.10 95% C. I. for Mean = (2.768, 2.612)
4. Probability of Change = 0.95 Smoothing Constant = 0.70 Mean Trader's Profit = -0.977 Standard Deviation = 0.02 95% C. I. for Mean = (-0.966, -0.989)	5. Probability of Change = 0.70 Smoothing Constant = 0. 70 Mean Trader's Profit = 2.839 Standard Deviation = 0.04 95% C. I. for Mean = (2.871, 2.807)	6. Probability of Change = 0.40 Smoothing Constant = 0. 70 Mean Trader's Profit = 2.818 Standard Deviation = 0.05 95% C. I. for Mean = (2.859, 2.778)
7. Probability of Change = 0.95 Smoothing Constant = 0.40 Mean Trader's Profit = 2.840 Standard Deviation = 0.07 95% C. I. for Mean = (2.899, 2.781)	8. Probability of Change = 0.7 Smoothing Constant = 0. 40 Mean Trader's Profit = 2.891 Standard Deviation = 0.04 95% C. I. for Mean = (2.926, 2.856)	9. Probability of Change = 0.40 Smoothing Constant = 0. 40 Mean Trader's Profit = 2.924 Standard Deviation = 0.03 95% C. I. for Mean = (2.947, 2.901)

* C.I. : Confidence Interval

It is evident from the simulation results that the weightage given to the past behavior of traders while calculating TRS influences the cooperation level within the trader community. Furthermore, the influence increases with the increase in the level of gullibility of traders. A simple approach like completely condoning the past misbehavior of agents or not condoning it at all unfortunately would not help in sustaining cooperation among traders. It is likely that the agents that have misbehaved i.e. not cooperated in the past, could refuse to show any repentance and continue to misbehave in future. Thus condoning the past misbehavior is tantamount to providing incomplete information about some of the potential traitors. On the contrary not condoning the past behavior at all would most certainly de-motivate the traders that are willing to make amends for the past misbehavior and ready to cooperate in future. Agents who have misbehaved in the past can be encouraged to cooperate if their past mistakes are given less weightage while calculating TRS. Thus it is critically important to fine tune the weightage given to the past behavior of agents by taking into account the level of gullibility of agents. An agent based model can be effectively used to analyze this issue. An agent based model can predict the limit to which the weightage given to the past behavior of traders can be safely increased without compromising on cooperation levels.

The careful analysis of the simulation results reveals the correlation between the Smoothing Constant (i.e. weightage given to the past behavior of an agent) and the Probability of Imitation (i.e. level of gullibility of traders). The maximum permissible probability of imitation to maintain full cooperation decreases with the increase in the smoothing constant. In other words, as weightage given to the old TRS is gradually increased, sustaining cooperation becomes increasingly difficult if the population is very

trusting. This is quite intuitive since giving higher weightage to the old TRS would result in making defection an attractive strategy for the traders. Furthermore, such dishonest traders would be able to quickly proliferate since the agent population is very trusting.

Probability of imitation i.e. level of gullibility of agents also affects mean trader profit. Mean trader profit decreases as the agent population becomes more and more trusting. In other words, the nuisance value of dishonest traders increases substantially as agents become more trusting. The extent to which the mean trader profit gets affected depends on the value of the smoothing constant. The higher the smoothing constant, the greater is the impact of level of gullibility on mean trader profit. Since the population mix is always going to be dynamic, it is important for a system designer to be able to predict the change in the mean trader profit for a particular population mix as the weightage given to the past behavior of agents is changed. This would enable him to fine tune the smoothing constant and increase the mean trader profit. So the agent based model would be used to calculate the change in the mean trader profit caused by a change in the level of gullibility of agents at different levels of smoothing constants.

A Smoothing Constant of 0.9 signifies that very high weightage is given to the old TRS. Figure 5.3 depicts the effect of a change in probability of imitation on mean trader profit for smoothing constant = 0.9. For this value of the smoothing constant, the mean trader profit decreases with the increase in the level of gullibility of agents. In fact, as the percentage of gullible agents in the population approaches 60%, the mean trader profit becomes -1 indicating that agents have stopped cooperating. This is an important

observation. For a particular population mix, it helps the system designer to understand the range of smoothing constants with in which the cooperation can be sustained.

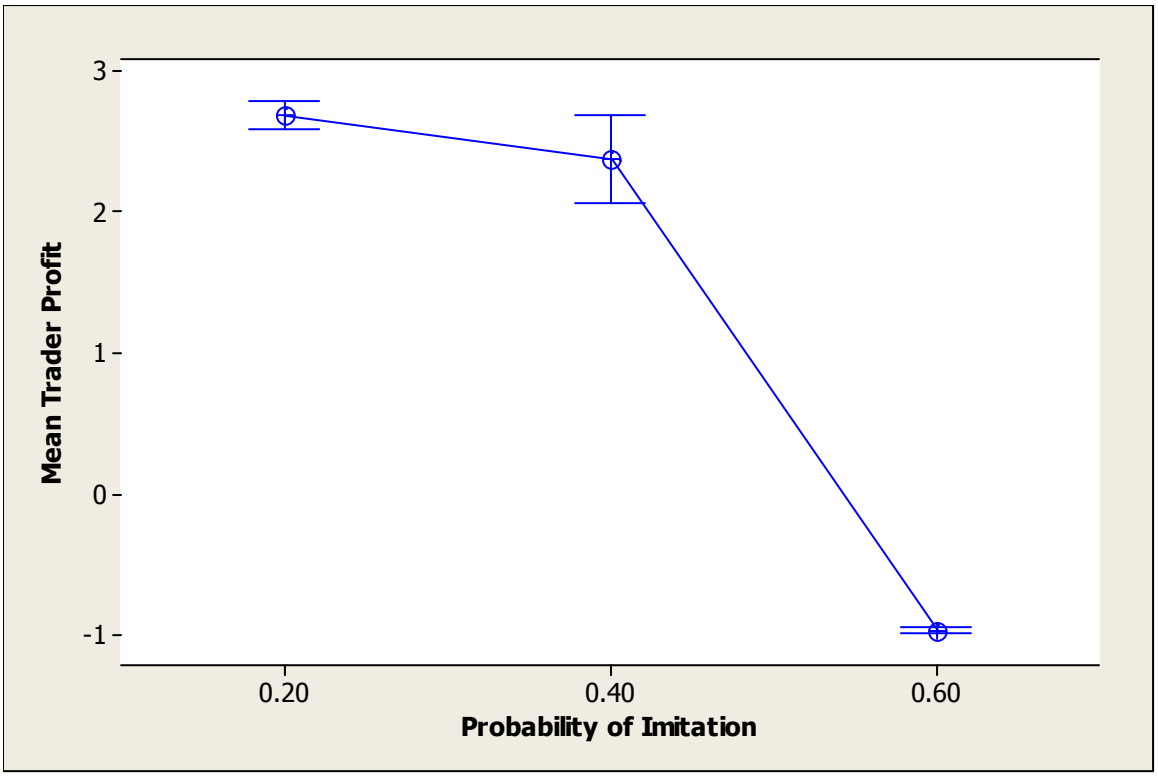


Figure 5.3: Effect of Change in Probability of Imitation on Mean Trader Profit for Smoothing Constant = 0.9

Figure 5.4 depicts the effect of a change in the probability of imitation on mean trader profit for a smoothing constant = 0.7. The reduction in Smoothing Constant from 0.9 to 0.7 signifies that weightage given to the old TRS is reduced. The mean trader profit becomes -1 when the probability of imitation approaches 95%. It is worth noting that the maximum permissible probability of imitation to avoid negative mean trader profit increases with the reduction in the smoothing constant.

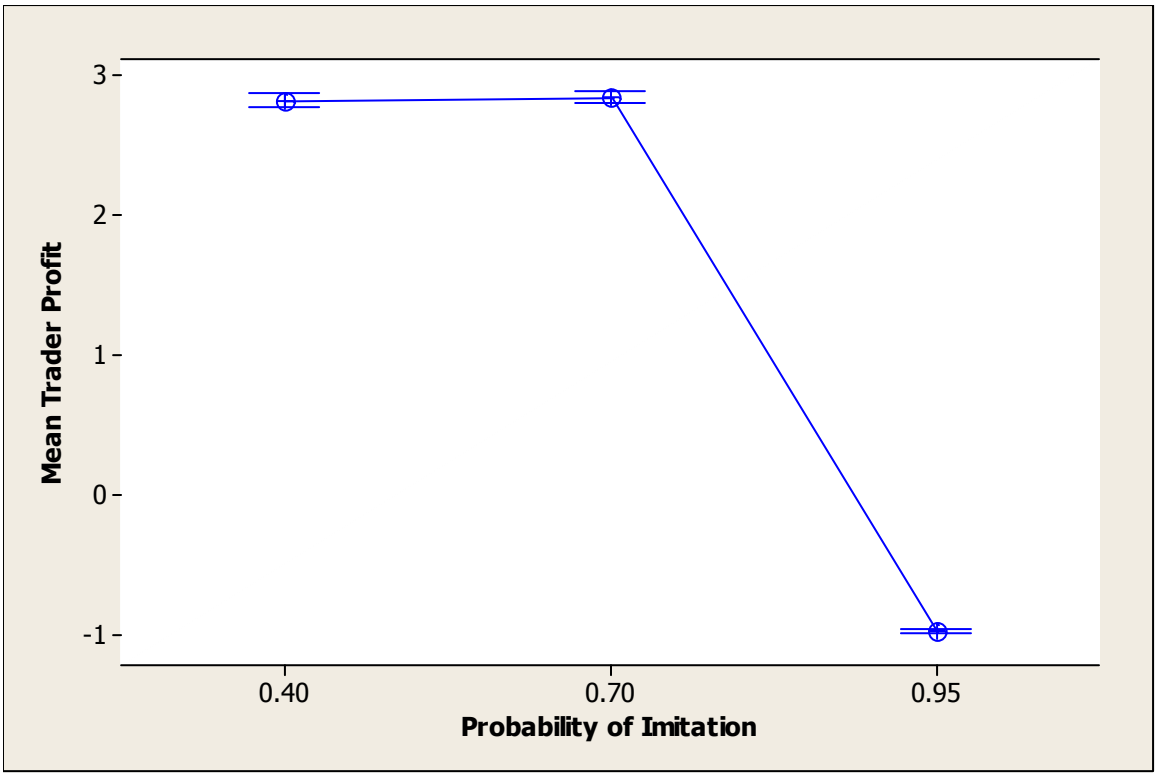


Figure 5.4: Effect of Change in Probability of Imitation on Mean Trader Profit for Smoothing Constant = 0.7

Figure 5.5 depicts the effect of a change in the probability of imitation on mean trader profit for a smoothing constant = 0.4. The further reduction in Smoothing Constant from 0.7 to 0.4 signifies that lesser weightage is given to the old TRS. As seen in the previous scenarios, Mean Trader Profit decreases with an increase in the probability of imitation i.e. as the traders become more and more trusting.

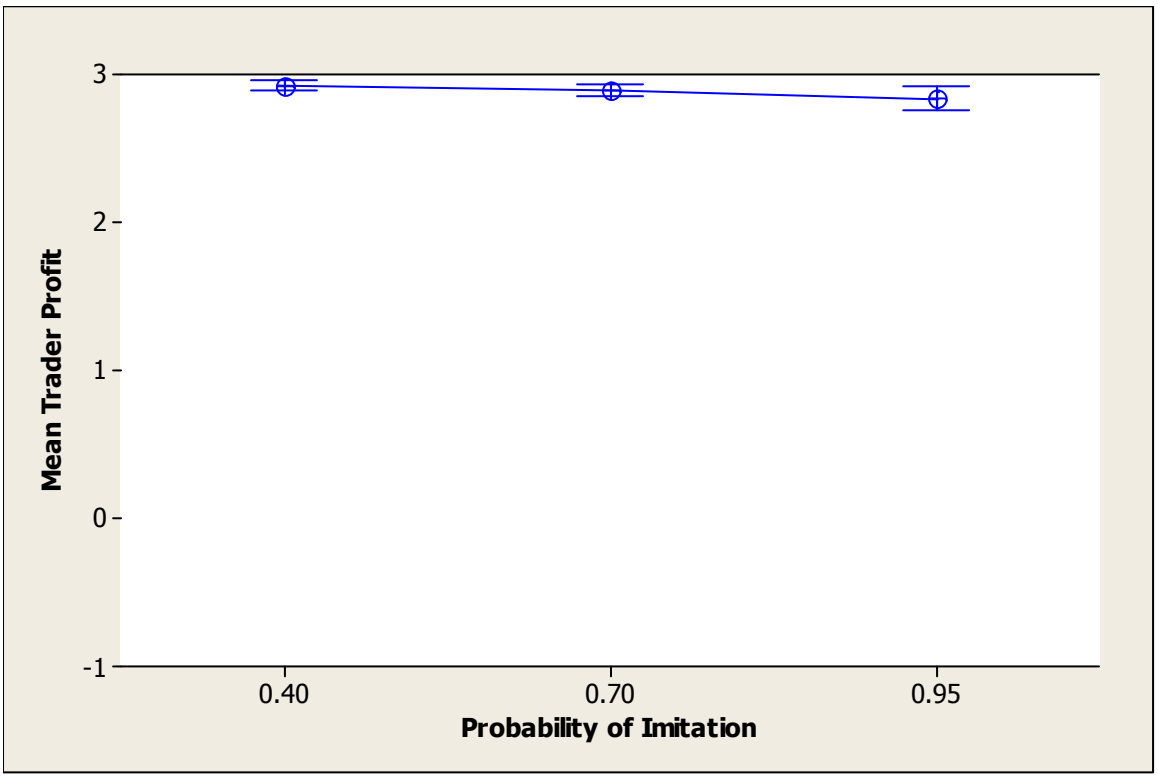


Figure 5.5 Effect of Change in Probability of Imitation on Mean Trader Profit for Smoothing Constant = 0.4

If the population mix (i.e. level of gullibility of traders) is known then the simulation model can be used to predict the mean trader profit at different levels of smoothing constants. Figures 5.6, 5.7 and 5.8 depict the effect of a change in smoothing constant on mean trader profit for different values of the probability of imitation. It can be seen that mean trader profit decreases as the smoothing constant is increased gradually, i.e. more weightage is given to the past behavior of agents. In other words, if excessive weightage is given to the past behavior of an agent then some of the agents with good track record scores might decide to defect and earn more profit without

affecting their track record score much since their past behavior was exemplary. Soon other agents will start imitating these traitors and thus mean trader profit will reduce. Furthermore, the drop in the mean trader profit is substantial for the agent population with high levels of gullibility simply because higher levels of gullibility allows traitors to proliferate very quickly.

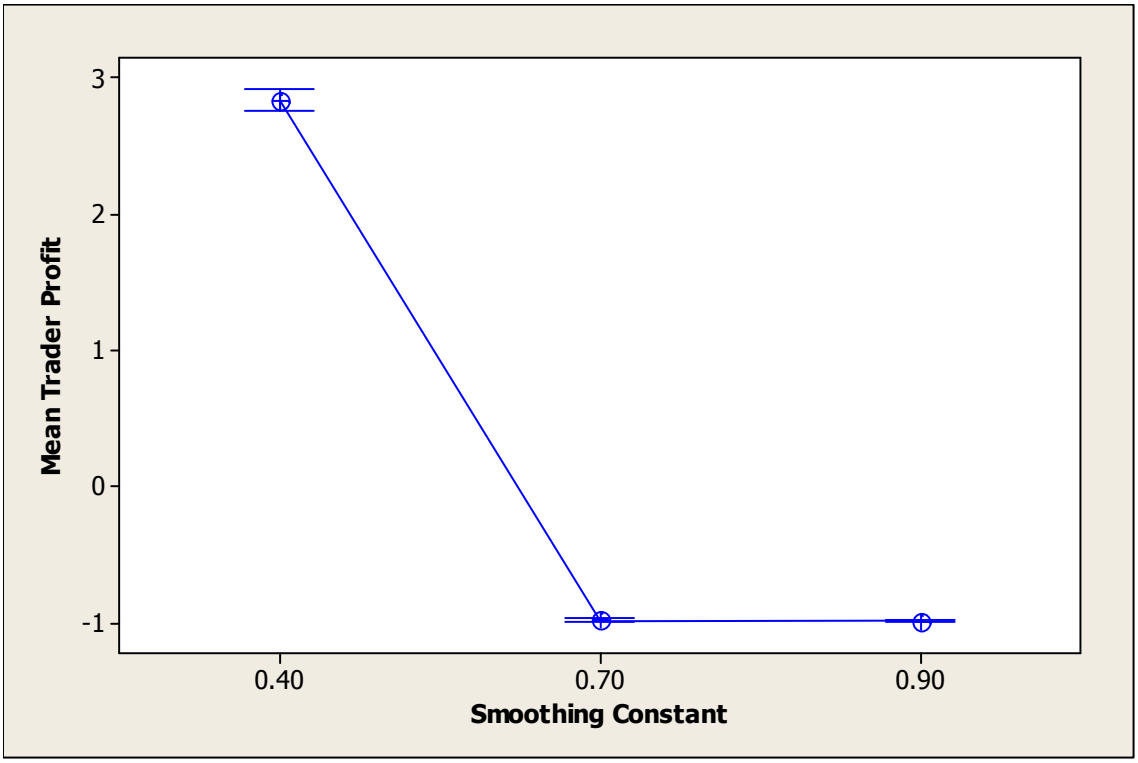


Figure 5.6 Effect of Change in Smoothing Constant on Mean Trader Profit for Probability of Imitation = 0.95

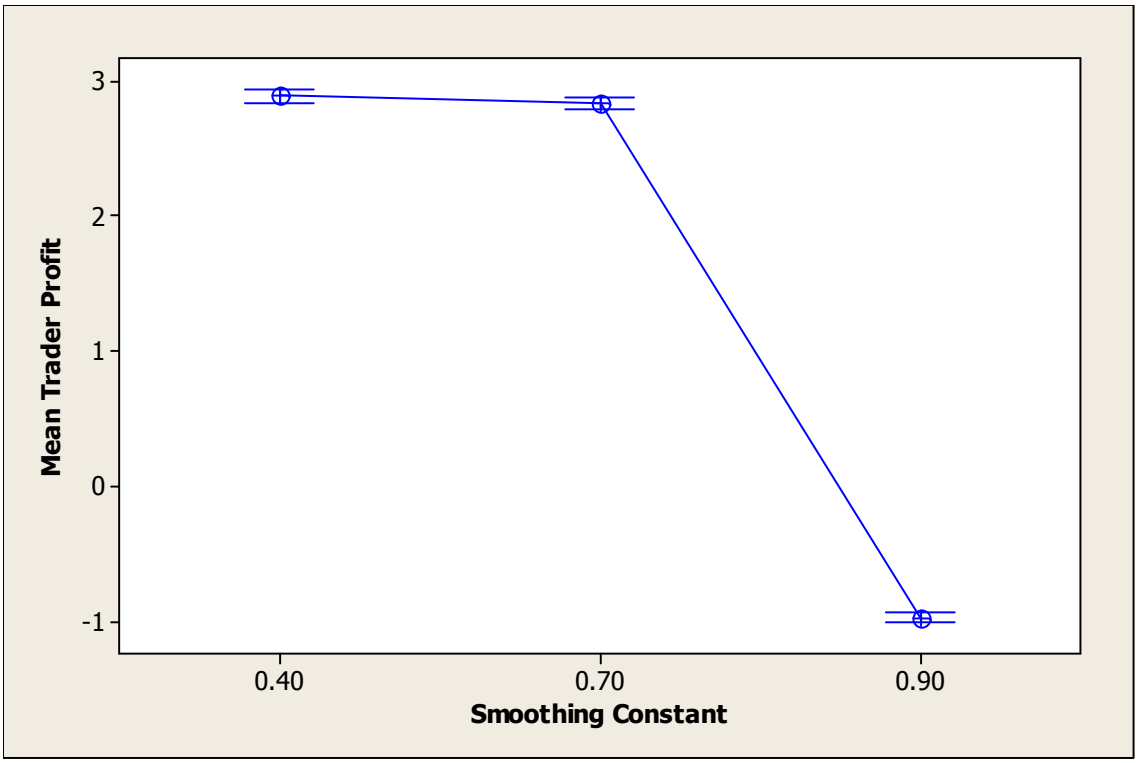


Figure 5.7 Effect of Change in Smoothing Constant on Mean Trader Profit for Probability of Imitation = 0.7

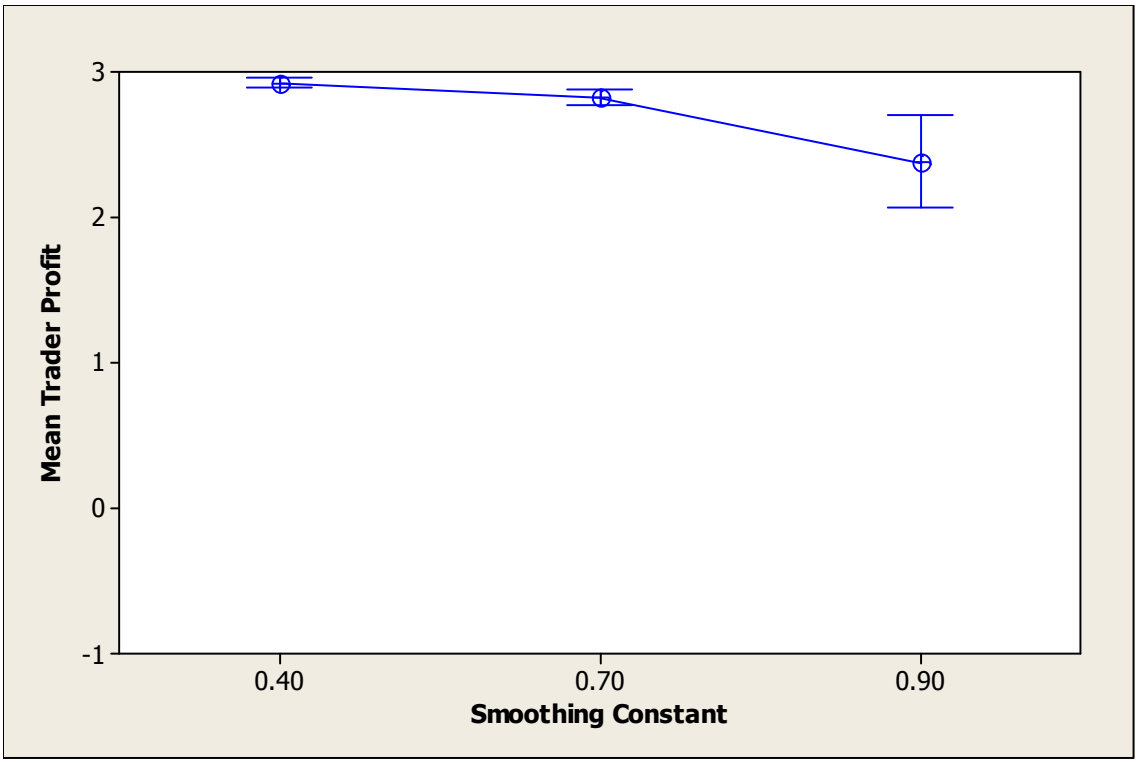


Figure 5.8 Effect of Change in Smoothing Constant on Mean Trader Profit for Probability of Imitation = 0.4

It is also necessary to test the sensitivity of the results for changes in the values of 'percentage of dishonest traders in the initial population mix' and 'payoff structure for agents'. To test this, the regular payoff structure is doubled ensuring that the fundamental T > R > P > S and 2R > T + S > 2P conditions as explained in the previous chapter are satisfied. The new payoff structure is explained in Table 5.2 below:

Table 5.2: Payoff Matrix for the Row Player

		Column Player	
		Cooperate(C)	Defect(D)
Row Player	Cooperate(C)	R (6)	S (0)
	Defect(D)	T (10)	P (2)

Also the percentage of dishonest traders in the initial population mix was also doubled from 40% to 80%. As expected, the results did not change except for the fact that the Mean Trader Profit was doubled because of the changed payoff structure. This is intuitive since the percentage of the dishonest traders in the initial population mix will not affect the results as long as cooperation emerges as the most profit making strategy and thus is likely to get adopted by other agents.

Analysis of the results above indicates that the trust based reputation systems like Track Record Score used in this research, could be effectively used to sustain the cooperation among online traders. However, every trader population is characterized by the unique level of gullibility of its traders. Furthermore, this level of gullibly of traders could change over time. The failure to factor in the level of gullibility of traders while designing the trust based reputation system may result in rendering the reputation system ineffective. In other words, there is no elixir for eliminating defection in the given trader population.

The system designer has to carefully study the population mix and then run the various what-if scenarios of an agent based simulation model gradually increasing the smoothing constant, i.e. the weight of influence attached to the past behavior of agents. This would enable the system designer to estimate the maximum permissible value of smoothing constant that can sustain the cooperation for the given population mix, although further fine tuning might be necessary after studying the dynamics of the population mix.

As a last step, the system designer would have to test the robustness of the Reputation System design. It is important not only to find the 'right answer' but also the details about when the answer isn't right any more. Sensitivity analysis on the important parameters like level of gullibility of agents would thus be required to ensure the robustness of the reputation system design.

CHAPTER 6

CONCLUSION

According to game theory, a dominant strategy for online trading modeled as Prisoner's dilemma game is defecting (Epstein, 1997). However, the Track Record Score reputation system designed by taking into account the level of gullibility of online traders is capable of sustaining cooperation among online traders.

It is essential to take into account past behavior of a trader while calculating his current Track Record Score because it serves as a deterrent for agents that are tempted to defect and earn larger short term profit. However, the weightage given to the past behavior should be decided after taking into account the level of gullibility of agents. It is observed that the cooperation would be difficult to sustain among relatively trusting online traders at higher values of the Smoothing Constant, i.e. if higher weightage is given to the past behavior of a trader. It is also observed that for a given value of the Smoothing Constant, the Mean Trader Profit decreases with an increase in gullibility of traders. This is exactly what happens in the real life online trading where dishonest traders feed on the relatively naïve and gullible traders who can be easily cheated.

The results presented in this thesis can be extended in several different ways. One of the several extensions could be to modify the reputation system in this research and include domain specific reputations. Domain specific reputation usually depends on quality of after-sales service in addition to the honest behavior of an online trader. For example, Amazon has a better reputation in the realm of books where as Wal-Mart has better reputation in the area of consumer electronics. Although both are considered

trustworthy traders, the buyers would still make their purchases by taking into account their domain specific reputations. In fact domain specific reputation can be considered as one of the reasons to explain unsuccessful attempts of Wal-Mart to replace Amazon.com as the most popular site for buying books online.

Another extension could be to give more importance to the feedback from agents that have higher Track Record Score. In other words, importance given to an Agent's feedback would be directly proportional to his/her Track Record Score. This would help honest traders to punish the dishonest traders even harder by quickly bringing down their Track Record Score and help honest traders to build their reputation sooner. The Track Record Score could be considered as the single most effective weapon against dishonest online traders and thus helping honest traders to build their reputation quicker would certainly shield them from being preyed upon by their dishonest counterparts.

This research assumes that all agents report the feedback meticulously. This unfortunately is not always true because many agents do not see any incentive in providing feedback. This in turn weakens the reputation system since many dishonest traders would be able to get away with their bad behavior simply because their business partners just did not report their misbehavior. Furthermore, the honest traders would find it difficult to build the Track Record Score because of less than 100% feedback reporting. It would thus be important to see the effects of imperfect reporting on the overall cooperation levels.

As a counter strategy to prevent imperfect reporting, incentives could be awarded to the agents that ensure 100% feedback reporting. For example, www.emove.com, refunds the transaction fee of $5.95 to their customers upon their

feedback reporting. Such monetary incentives serve as a source of motivation for the people who otherwise would not have provided a feedback. In the moving business, the reputation of the movers matters the most to the customers. Having more reviews helps 'emove.com' to attract more customers to their website. Furthermore, the movers also provide the best service to ensure positive feedback because presence of even a single negative feedback can turn potential customers away from them. This is an excellent example that underscores the importance of feedback reporting. Furthermore, it also leads us to one more interesting possibility that is worth considering. A group of agents might decide to do fake transactions and provide excellent feedbacks to each other in order to increase their feedback scores. The reputation system should have enough checks and balances in place to avoid such cases from happening.

One more possibility is that some of the honest traders might be complacent. After collecting a certain amount of wealth, they might just decide to quit the business. Such complacent traders might make it harder to sustain the cooperation levels. Similarly, the traders with the low Track Record Scores might decide to quit and re-enter with a different identities to hide their dishonest behavior in the past. The research could be extended to quantify the effect of such agent behavior.

The cost of searching for suitable business partners could also be introduced in this agent based model. This research assumes that the agents do not incur any cost for searching for the appropriate partner. However, time is a critical resource so it is only logical to introduce the cost of searching for suitable business partners and ensure that the conservative agents spending more time to search for business partners would have to pay some price for their slow response time. This would create a level playing field

for the risk seeking online traders in the model who choose their business partners relatively quickly.

The agent strategies used in this research are Always Co-operate, Always Defect and Tit for Tat. However, the strategy space could be further expanded to include Pavlov, Mistrust (defects on the first move and then mirrors opponent's move there on), Spiteful (co-operates until first opponent's defection and then always defects) and other popular strategies used in the agent based literature.

The use of agent based modeling for studying dynamics of online trading is a relatively new concept. Sustaining cooperation among online traders is often perceived to be a difficult task. The Track Record Score concept used in this research is expected to make contribution towards this goal.

APPENDIX

JAVA SOURCE CODE FOR THE AGENT BASED SIMULATION MODEL

```
package shriganesha.dscm.Decentralized_Supply_Chain_Model;

import uchicago.src.sim.space.*;
import uchicago.src.sim.engine.*;
import uchicago.src.sim.util.SimUtilities;
import java.util.*;
import java.lang.Math;

public class DSCM extends SimpleModel {

  public ArrayList agentList;
  ArrayList agentListN;
  ArrayList agentListS;
  ArrayList agentListE;
  ArrayList agentListW;

  Object2DTorus Region_W;
  Object2DTorus Region_E;
  Object2DTorus Region_S;
  Object2DTorus Region_N;

  public double cumulSCprofit;
  public int totalnumplays;

  final int Loyal = 0;
  final int Practical = 1;
  final int Traitor = 2;
  final int strategyTypes = 3;


  int grid_Side;
  int agentPopulation;
  double prob_Loyal, prob_Practical, prob_Traitor;
  int num[];
  int numDealings;
  int locals;
  double prob_Change;
  int Searches_PerDay;
  int MaxSearches_PerDay;
```

```
double smoothing_Const = 0.4;

public DSCM(){
  name = "Agent Based Simulation Model of Online Trading: Version # 8";
}

public void setup() {
  super.setup();
  name = "Agent Based Simulation Model of Online Trading: Version # 8";
  grid_Side = 20;

  prob_Loyal = 0.3;
  prob_Practical = 0.3;
  prob_Traitor = 0.4;
  numDealings = 2;

  locals = 4;
  prob_Change = 0.4;
  Searches_PerDay = 5;
  MaxSearches_PerDay = 5;
}

public void buildModel() {


  agentList = new ArrayList();
  agentListN = new ArrayList();
  agentListS = new ArrayList();
  agentListE = new ArrayList();
  agentListW = new ArrayList();

  Region_W = new Object2DTorus(grid_Side/2, grid_Side/2);
  Region_N = new Object2DTorus(grid_Side/2, grid_Side/2);
  Region_S = new Object2DTorus(grid_Side/2, grid_Side/2);
  Region_E = new Object2DTorus(grid_Side/2, grid_Side/2);

  agentPopulation = grid_Side * grid_Side;
  num = new int[3];
  num[Loyal] = (int) (agentPopulation * prob_Loyal);
  num[Practical] = (int) (agentPopulation * prob_Practical);
  num[Traitor] = (int) (agentPopulation * prob_Traitor);

  if (num[Traitor] != agentPopulation - num[Loyal] - num[Practical] )
    num[Traitor] = agentPopulation - num[Loyal] - num[Practical] ;
```

```
double TRS = 100;
int Status = 1;
int Category;
int playerID = 0;
cumulSCprofit = 0;
totalnumplays = 0;

for (int playerType = 0; playerType < strategyTypes; playerType++){
 for (int i = 0; i<(num[playerType])/8; i++) {
  playerID++;
  SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
     Category = 1,  playerType, this);
  agentList.add(agent);
  agentListN.add(agent);
 }

 for (int i = 0; i<(num[playerType])/8; i++) {
  playerID++;
  SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
     Category = 2,  playerType, this);
  agentList.add(agent);
  agentListN.add(agent);
 }

 for (int j = 0; j<(num[playerType])/8; j++) {
  playerID++;
  SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
     Category = 1,  playerType, this);
  agentList.add(agent);
  agentListS.add(agent);
 }

 for (int j = 0; j<(num[playerType])/8; j++) {
  playerID++;
  SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
     Category = 2,  playerType, this);
  agentList.add(agent);
  agentListS.add(agent);
 }

 for (int i = 0; i<(num[playerType])/8; i++) {
  playerID++;
  SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
     Category = 1,  playerType, this);
```

```
    agentList.add(agent);
    agentListE.add(agent);
   }

   for (int i = 0; i<(num[playerType])/8; i++) {
    playerID++;
    SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
       Category = 2,  playerType, this);
    agentList.add(agent);
    agentListE.add(agent);
   }


   for (int i = 0; i<(num[playerType])/8; i++) {
    playerID++;
    SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
       Category = 1,  playerType, this);
    agentList.add(agent);
    agentListW.add(agent);
   }

   for (int i = 0; i<(num[playerType])/8; i++) {
    playerID++;
    SupplyChainAgent agent = new SupplyChainAgent(playerID, TRS, Status,
       Category = 2,  playerType, this);
    agentList.add(agent);
    agentListW.add(agent);
   }
  }

 SimUtilities.shuffle(agentList);
 SimUtilities.shuffle(agentListN);
 SimUtilities.shuffle(agentListS);
 SimUtilities.shuffle(agentListW);
 SimUtilities.shuffle(agentListE);

 for (int x = 0;  x < grid_Side/2; x++){
   for (int y = 0; y < grid_Side/2; y++) {
    SupplyChainAgent agent = (SupplyChainAgent) agentListN.get(x*grid_Side/2+y);
    Region_N.putObjectAt(x, y, agent);
    agent.placeTo(x, y);
   }
 }

 for (int x = 0;  x < grid_Side/2; x++){
```

```
    for (int y = 0; y < grid_Side/2; y++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListS.get(x*grid_Side/2+y);
      Region_S.putObjectAt(x, y, agent);
      agent.placeTo(x, y);
    }
  }

  for (int x = 0;  x < grid_Side/2; x++){
    for (int y = 0; y < grid_Side/2; y++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListE.get(x*grid_Side/2+y);
      Region_E.putObjectAt(x, y, agent);
      agent.placeTo(x, y);
    }
  }

  for (int x = 0;  x < grid_Side/2; x++){
    for (int y = 0; y < grid_Side/2; y++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListW.get(x*grid_Side/2+y);
      Region_W.putObjectAt(x, y, agent);
      agent.placeTo(x, y);
    }
  }
  Summery();
}

public void BusinessDealing(SupplyChainAgent rowPlayer, SupplyChainAgent
        colPlayer) {

  rowPlayer.BusinessPartner = colPlayer;
  rowPlayer.pastBusinessPartners.add(colPlayer);
  colPlayer.BusinessPartner = rowPlayer;
  colPlayer.pastBusinessPartners.add(rowPlayer);


  if ( rowPlayer.type <= 1 &&  colPlayer.type <= 1 ) {
    rowPlayer.TRS =  (smoothing_Const*(rowPlayer.TRS) + (1-
        smoothing_Const)*(100));
    colPlayer.TRS =  (smoothing_Const*(colPlayer.TRS) + (1-
        smoothing_Const)*(100));

  }

  if ( rowPlayer.type == 2  ||  colPlayer.type == 2 ) {
    if  (rowPlayer.type == 0)  rowPlayer.TRS =  (smoothing_Const*(rowPlayer.TRS) +
        (1-smoothing_Const)*(100));
```

```
    if  (rowPlayer.type == 1)  rowPlayer.TRS =  (smoothing_Const*(rowPlayer.TRS) +
        (1-smoothing_Const)*(0));
    if  (rowPlayer.type == 2)  rowPlayer.TRS =  (smoothing_Const*(rowPlayer.TRS) -
        (1-smoothing_Const)*(100));
    if  (colPlayer.type == 0)  colPlayer.TRS =  (smoothing_Const*(colPlayer.TRS) + (1-
        smoothing_Const)*(100));
    if  (colPlayer.type == 1)  colPlayer.TRS =  (smoothing_Const*(colPlayer.TRS) + (1-
        smoothing_Const)*(0));
    if  (colPlayer.type == 2)  colPlayer.TRS =  (smoothing_Const*(colPlayer.TRS) - (1-
        smoothing_Const)*(100));
  }

  for (int i=1; i<=numDealings; i++) {
    totalnumplays = totalnumplays + 1;
    rowPlayer.beginDealing(i);
    colPlayer.beginDealing(i);
    rowPlayer.remember();
    colPlayer.remember();
    rowPlayer.profitSum();
    colPlayer.profitSum();
  }
}

public void statuscheck() {
  for (int i=0; i < agentPopulation; i++) {
    SupplyChainAgent agent = (SupplyChainAgent) agentList.get(i);
    agent.status();
  }
}

public void interactions() {
  for (int i=0; i < agentPopulation; i++) {
     SupplyChainAgent agent = (SupplyChainAgent) agentList.get(i);

    for (int j=0; j<MaxSearches_PerDay; j++){
      int randomNumber;
      SupplyChainAgent BusinessPartner;
      do {
        randomNumber = getNextIntFromTo (0, agentPopulation-1);

        BusinessPartner = (SupplyChainAgent) agentList.get(randomNumber);
      } while (BusinessPartner.playerID == agent.playerID);
```

```
      if ((100* (getNextDoubleFromTo(0.0,1.0))) <  (
        ((Math.max(Math.min(agent.TRS, BusinessPartner.TRS ), 0))) +    (100 -
        (Math.max(agent.TRS, BusinessPartner.TRS ) -
        Math.max(Math.min(agent.TRS, BusinessPartner.TRS ), 0)  ))  )/2)

      BusinessDealing(agent, BusinessPartner);
    }
  }
}

public void StrategyDecider() {

  for (int i=0; i<agentPopulation; i++) {
    SupplyChainAgent agent = (SupplyChainAgent)agentList.get(i);
    agent.strategyDeciding();
  }

  for (int i=0; i<agentPopulation; i++) {
    SupplyChainAgent agent = (SupplyChainAgent)agentList.get(i);
    agent.updateType();
  }
}

public void Summery() {

  for (int i=Loyal; i<strategyTypes; i++)
    num[i] = 0;{

    for (int i=0; i<agentPopulation/4; i++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListN.get(i);
      num[agent.type]++;
    }

    for (int i=0; i<agentPopulation/4; i++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListS.get(i);
      num[agent.type]++;
    }

    for (int i=0; i<agentPopulation/4; i++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListW.get(i);
      num[agent.type]++;
    }

    for (int i=0; i<agentPopulation/4; i++) {
      SupplyChainAgent agent = (SupplyChainAgent) agentListE.get(i);
```

```
        num[agent.type]++;
      }
    }
  }

  public void step() {
    interactions();
    StrategyDecider();
    Summery();
  }

  public static void main(String[] args) {
    SimInit init = new SimInit();
    DSCM m = new DSCM();
    init.loadModel(m, null, false);
  }
}

package shriganesha.dscm.Decentralized_Supply_Chain_Model;

import uchicago.src.sim.gui.*;
import uchicago.src.sim.engine.*;
import uchicago.src.sim.analysis.*;
import java.awt.Color;

public class DSCMGraphics extends DSCM {

  public OpenSequenceGraph graph;
  public OpenSequenceGraph graph1;

  public DisplaySurface dsurfN;
  public DisplaySurface dsurfS;
  public DisplaySurface dsurfE;
  public DisplaySurface dsurfW;

  public DSCMGraphics() {
    Controller.ALPHA_ORDER = false;
  }

  class SeqCumSCProfit implements Sequence {
    public double getSValue() {
      if (totalnumplays > 0)
    return (double) 0.5*(cumulSCprofit/(double)totalnumplays);
    else
      return 0;;
```

```
  }
}

class SeqLoyal implements Sequence {
  public double getSValue() {
    return (double) num[Loyal];
  }
}

class SeqPractical implements Sequence {
  public double getSValue() {
    return (double) num[Practical];
  }
}

class SeqTraitor implements Sequence {
  public double getSValue() {
    return (double) num[Traitor];
  }
}

public void setup() {
  super.setup();
  params = new String[] {"grid_Side","prob_Loyal", "prob_Practical",
    "prob_Traitor", "prob_Change", "smoothing_Const", "MaxSearches_PerDay"};

  if (graph != null)
    graph.dispose();
  if (graph1 != null)
    graph1.dispose();
  if (dsurfN != null)
    dsurfN.dispose();
  if (dsurfS != null)
    dsurfS.dispose();
  if (dsurfE != null)
    dsurfE.dispose();
  if (dsurfW != null)
    dsurfW.dispose();


  dsurfN = new DisplaySurface(this, "Northen Region");
  dsurfS = new DisplaySurface(this, "Southern Region");
  dsurfE = new DisplaySurface(this, "Eastern Region");
  dsurfW = new DisplaySurface(this, "Western Region");
```

```
  registerDisplaySurface("Main", dsurfN);
  registerDisplaySurface("Main", dsurfS);
  registerDisplaySurface("Main", dsurfE);
  registerDisplaySurface("Main", dsurfW);

}

public void buildModel() {

  super.buildModel();

  graph = new OpenSequenceGraph("Real Time Distribution of Trader Types", this);
  graph.setXRange(0, 100);
  graph.setYRange(0.0, (double) agentPopulation);
  graph.setAxisTitles("Hours", "Number of Traders");

  graph.addSequence("Loyal", new SeqLoyal(), Color.green);
  graph.addSequence("Practical", new SeqPractical(), Color.blue);
  graph.addSequence("Traitor", new SeqTraitor(), Color.red);

  graph.display();
  graph.step();

  graph1 = new OpenSequenceGraph("Mean Trader Profit Graph", this);
  graph1.setXRange(0, 100);
  graph1.setYRange(0.0, 5.0);
  graph1.setAxisTitles("Hours", "Profit");

  graph1.addSequence("", new SeqCumSCProfit(), Color.cyan);


  graph1.display();
  graph1.step();

  DisplayConstants.CELL_WIDTH = 6;
  DisplayConstants.CELL_HEIGHT = 6;

  Object2DDisplay displayN = new Object2DDisplay(Region_N);
  Object2DDisplay displayS = new Object2DDisplay(Region_S);
  Object2DDisplay displayE = new Object2DDisplay(Region_E);
  Object2DDisplay displayW = new Object2DDisplay(Region_W);

  displayN.setObjectList(agentListN);
  displayS.setObjectList(agentListS);
  displayE.setObjectList(agentListE);
```

```
  displayW.setObjectList(agentListW);

  dsurfN.addDisplayableProbeable(displayN, "Northen Region");
  dsurfS.addDisplayableProbeable(displayS, "Southern Region");
  dsurfE.addDisplayableProbeable(displayE, "Eastern Region");
  dsurfW.addDisplayableProbeable(displayW, "Western Region");

  addSimEventListener(dsurfN);
  addSimEventListener(dsurfS);
  addSimEventListener(dsurfE);
  addSimEventListener(dsurfW);

  dsurfN.display();
  dsurfS.display();
  dsurfE.display();
  dsurfW.display();

}

public void step() {

  super.step();

  graph.step();
  graph1.step();
  dsurfN.updateDisplay();
  dsurfS.updateDisplay();
  dsurfE.updateDisplay();
  dsurfW.updateDisplay();

}

public int getgrid_Side() {
  return grid_Side;
}

public void setgrid_Side(int n) {
  grid_Side = n;
}

public double getprob_Loyal() {
  return prob_Loyal;
}

public void setprob_Loyal(double p) {
```

```
    prob_Loyal = p;
  }

  public double getprob_Practical() {
    return prob_Practical;
  }

  public void setprob_Practical(double p) {
    prob_Practical = p;
  }

  public double getprob_Traitor() {
    return prob_Traitor;
  }

  public void setprob_Traitor(double p) {
    prob_Traitor = p;
  }

  public double getprob_Change() {
    return prob_Change;
  }

  public double getsmoothing_Const() {
    return smoothing_Const;
  }

  public void setsmoothing_Const(double p) {
  smoothing_Const = p;
  }

  public void setprob_Change(double p) {
    prob_Change = p;
  }

  public static void main(String[] args) {
    SimInit init = new SimInit();

    DSCM m = new DSCMGraphics();
    init.loadModel(m, null, false);
  }
}

package shriganesha.dscm.Decentralized_Supply_Chain_Model;
import uchicago.src.sim.util.SimUtilities;
```

```
import java.awt.Color;
import uchicago.src.sim.gui.*;
import java.util.ArrayList;
import java.util.Iterator;

public class SupplyChainAgent implements Drawable {
  final int C = 1;
  final int D = 0;
  final String[] actionToString = {"D", "C"};
  public final int ifLoyal[] = {C, C, D};
  public final int ifPractical[] = {C, C, D};
  public final int ifTraiter[] = {C, D, D};

  double TRS;
  int Category;
  int Status;
  int x, y;
  DSCM model;
  int playerID;
  SupplyChainAgent BusinessPartner;
  int type;
  int newType;
  int[][] profitMatrix = { {1,5},
     {0,3} };
  int action;
  int memory;
  int totalProfit;
  int numPlays;
  ArrayList pastBusinessPartners;

  public SupplyChainAgent (int i, double trs, int status, int category,  int t, DSCM m) {
    TRS = trs;
    Status = status;
    Category = category;
    type = t;
    model = m;
    playerID = i;
    pastBusinessPartners = new ArrayList();
  }

  public void status() {
    if ( TRS <= 0 ) {
      if (model.getNextDoubleFromTo(0.0,1.0) < model.prob_Change) {
        Status = 0;
      }
```

```
  }
}

public void placeTo(int a, int b) {
  x = a;
  y = b;
}

public void reset() {
  numPlays = 0;
  totalProfit = 0;
  pastBusinessPartners.clear();
}

public void remember() {
  memory = BusinessPartner.action;
}

public void beginDealing(int time) {
  numPlays++;
  if (time == 1)
    action = ifLoyal[type];
  else
    if (memory == C)
      action = ifPractical[type];
  else
    action = ifTraiter[type];
}

public void profitSum() {
  totalProfit = totalProfit + profitMatrix[action][BusinessPartner.action];
  if (action > 0)
    model.cumulSCprofit = model.cumulSCprofit +
        profitMatrix[action][BusinessPartner.action];
  else
    model.cumulSCprofit = model.cumulSCprofit -
        profitMatrix[action][BusinessPartner.action];
}

public void strategyDeciding() {
  newType = type;
  if (model.getNextDoubleFromTo(0.0,1.0) < model.prob_Change) {
    SimUtilities.shuffle(model.agentList);
    double bestPayoff = getAveragePayoff();
    Iterator i = model.agentList.iterator();
```

```
    while (i.hasNext()) {
      SupplyChainAgent aPlayer = (SupplyChainAgent)i.next();
      double payoff = aPlayer.getAveragePayoff();
      if (payoff > bestPayoff) {
        bestPayoff = payoff;
        newType = aPlayer.type;
      }
    }
  }
}

public void updateType() {
  type = newType;
}

public double getAveragePayoff() {
  if (numPlays == 0)
    return -9.999;
  else
    return (double)totalProfit/(double)numPlays;
}

public int getX() {
  return x;
}

public int getY() {
  return y;
}

public double getcumulSCprofit() {
  if (model.totalnumplays > 0)
  return (double) 0.5*(model.cumulSCprofit/(double)model.totalnumplays);
  else
    return 0;
}

public int getType () {
  return type;
}

public void setType (int v) {
  type = v;
}
```

```
  final Color COLOR[] = {Color.green, Color.blue, Color.red, Color.magenta};


  public void draw(SimGraphics g) {
    g.setDrawingParameters( DisplayConstants.CELL_WIDTH * 2 / 3,
                            DisplayConstants.CELL_HEIGHT * 2 / 3,
                            DisplayConstants.CELL_DEPTH * 2 / 3 );
    g.drawFastRoundRect(COLOR[type]);
  }
}
```

BIBLIOGRAPHY

Axelrod Robert. The Evolution of Cooperation. New York, Basic Books, 1984.

Axelrod Robert. Advancing the Art of Simulation in the Social Sciences. Santa Fe Institute, 1997.

Boyd, Gintis, Bowles and Richerson. The evolution of altruistic punishment. PNAS Vol. 100, no. 6, pp 3531-3535, 2003.

Cederman Lars-Erik. Emergent Actors in World Politics: How States and Nations Develop and Dissolve. Princeton University Press, 1997.

Cohen, M.D., Riolo, R.L., Axelrod, R. The Emergence of Social Organization in the Prisoner's Dilemma: How Context- Preservation and other Factors Promote Cooperation.University of Michigan, 1998.

Nick Collier. RePast: An Extensible Framework for Agent Simulation. Technical report, Social Science Research Computing, University of Chicago, Chicago, Illinois, 2000.

Dellarocas, C. Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior. Proceedings of the 2nd ACM Conference on Electronic Commerce, 2000.

Shoumen Palit Austin Datta and Clive W. J. Granger. Advances In Supply Chain Management: Potential To Improve Forecasting Accuracy. Massachusetts Institute of Technology Engineering Systems Division Working Paper Series ESD-WP-2006-11

Epstein, J. Zones of Cooperation in Demographic Prisoner's Dilemma. Technical Report, Santa Fe Institute, 1997.

Hoffman and Waring. The Localization of Interaction and Learning in the Repeated Prisoner's Dilemma. Sante Fe Institute Working Paper, 96-08-064. Santa Fe, NM, 1996.

Holland. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, 1975.

Imhof, Fudenberg, and Nowak. Evolutionary cycles of cooperation and defection. PNAS, Vol 102, No 31. 2005.

Luce and Raiffa. Games and Decisions: Introduction and Critical Survey. Wiley and Sons, 1957.

Miller, J.H. The evolution of automata in the repeated prisoner's dilemma in Two essays on the economics of imperfect information. PhD dissertation, University of Michigan and Journal of economic behavior and organization, 29 (1): 87 – 112, 1996.

Nakamura, M., H. Matsuda, and Y. Iwasa. The evolution of cooperation in a lattice-structured population. Journal Theoretical Biology, 184:65-81, 1997.

Nowak, M.A. And R.M. May. Evolutionary games and spatial chaos. Nature, 359 826-829, 1992.

Nowak, M. and Sigmund, K. A stragegy of win stay lose shift that outperforms tit for tat in the Prisoner's Dilemma game. Nature, 364: 56-58, 1993.

Nowak, M. and Sigmund, K. Chaos and the evolution of cooperation. PNAS Vol 90, pp 5091-5094, 1993.

Oliphant, M. Evolving Cooperation in the Non-Iterated Prisoner's Dilemma: The Importance of Spatial Organization. In Proceedings of the Fourth Artificial Life Workshop, 349-352, 1994.

Riolo, R. L.; Parunak, H.V.D. and Savit, R. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide In Multi-agent systems and Agent-based Simulation, 1998.

Rubinstein A. Finite automata play the repeated prisoner's dilemma. Journal of Economic Theory 39 (June) 83-96, 1986.

Scali Daniel J. Evolving Strategies for the Repeated Prisoner's Dilemma Game with Genetic Programming: Studying the Effect of Varying Function Sets. Undergraduate Honors Thesis Advised by Professor Sergio Alvarez Computer Science Department, Boston College. 2006.

Stanley, E. A., Ashlock, D., and Tesfatsion, L. Iterated Prisoner's Dilemma with choice and refusal of partners. pp. 131-175 in C. G. Langton (Ed.) Artificial Life III (Addison-Wesley, Reading, MA), 1994.

Steven F. Railsback, Steven L. Lytinen and Stephen K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations, Simulation, Vol. 82, No. 9, 609-623, 2006

Yamamoto, H., Ishida, K and Ohta. Jackson. Managing Online Trade by Reputation Circulation: An Agent-Based Approach to the C2C Market, Proc. of The 7th World Multi-Conference on Systemics, Cybernetics and Informatics. vol.1, pp.60-64, 2003.