INTELLIGENT AGENTS AS A MODELLING PARADIGM

By

KAFUI MONU

B.Comm (Honours), University of Manitoba, Winnipeg, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN BUSINESS ADMINISTRATION

in

THE FACULTY OF GRADUATE STUDIES

(MANAGEMENT INFORMATION SYSTEMS)

THE UNIVERSITY OF BRITISH COLUMBIA

August 2005

© Kafui Monu, 2005

Abstract

Intelligent software agents have been used in many applications because they provide useful integrated features that are not available in "traditional" types of software (e.g., abilities to sense the environment, reason, and interact with other agents). Although the usefulness of agents is in having such capabilities, methods and tools for developing them have focused on practical physical representation rather than accurate conceptualizations of these functions. Like other computer systems, intelligent agents usually represent some real world phenomena or environments. Consequently, intelligent agents should closely mimic aspects of the environment in which they operate. In the physical sciences, a conceptual model of a problem can lead to better theories and explanations about the area. Therefore, we ask *how can an intelligent agent conceptual framework, properly defined, be used to model complex interactions in various social science disciplines?*

The constructs used in the implementation of intelligent agents may not be appropriate at the conceptual level, as they refer to software concepts rather than to application domain concepts. Therefore we propose to use a combination of the systems approach and Bunge's ontology as adapted to information systems, to guide us in defining intelligent agent concepts. The systems approach will be used to define the components of the intelligent agents. Once the components have been identified we will use ontology to understand the configurations, transitions, and interrelationships between the components. We will then provide a graphical representation of these concepts for modelling purposes.

As a proof of concept for the proposed conceptual model, we apply it to a marketing problem and implement it in an agent-based programming environment called Netlogo. With the aid of the conceptual model, the user was able to quickly visualize the complex interactions of different agents. The use of the conceptual representation even sparked an investigation of previously neglected causal factors which led to a better understanding of the problem. The implications of these findings, and further research avenues, are also discussed. Since the proof of concept was successful, it can be said that we provide an intelligent agent framework that can graphically model phenomena in the social sciences. However, there are other contributions derived from the work, including; a theoretically driven concept of intelligent agent components, a way of showing the interrelationships between these concepts, and the foundation for an ontologically complete model of intelligent agents.

Table Of Contents

List of Tables

List of Figures

Acknowledgement

I would first like to thank my supervisor, Dr. Carson Woo who was extremely helpful over the course of the thesis. Not only did he encourage me to look into this area of research, but his advice throughout the thesis was invaluable.

I would also like to thank Drs. Yair Wand and Dan Putler. Dr. Wand was able to take a look at my early work and provide very useful feedback on what research direction I should take and methodologies I should study. His insights were highly valued and the thesis became much stronger because of them. I am also grateful to Dr. Putler for his patience since, even though it seemed as if we were speaking different languages sometimes, he took the time to fully understand what I was saying and explain to me what he had said. It was a pleasure to work with someone from "across the aisle".

There are others who helped with the creation of this work, whether they are aware of it or not. I would like to thank the members of St. John's College at the University of British Columbia for our discussions on topics of an academic, and not so academic, nature.

Lastly, I would like to extend my greatest appreciation to my parents and sister who were not only wonderful editors, sounding boards, and reviewers but were also emotional support. This work would not have been possible without them.

Chapter 1. Introduction

There are certain phenomena in business that involve a complex interplay between the macro and micro levels (Tesfatsion, 2002). Some researchers have noted that agent-based simulations can be used to solve such complex problems (Axelrod, 1997). However, no standard method for creating these models exist, and the definition of agent components (even the nature of an agent) is not clear (Drogoul, Vanbergue, & Meurisse, 2002).

Multi-agent systems have been used to simulate the interaction within large groups in various settings (Drogoul, Vanbergue, & Meurisse, 2002). These simulations are used to understand the complex patterns which can emerge from the actions and interactions of individuals. Even though the tools to create these systems have become simpler, the task of building a good agent-based simulation is still complicated, due to the increased complexity of the modelled interactions.

A conceptual model can lead to better theories and explanations of phenomena in a given domain. One example of this is that if gas molecules are conceptualised as billiard balls it is easier to understand their movement in physical space (Casti, 1989). Without this conceptualisation, it is possible that physicists would not know how to reconcile their previously erroneous theory with real world observations. If the wrong conceptualisation is picked then predictions inferred from the model may be incorrect. This is shown in the following diagram adapted from Holland et. al (1986).

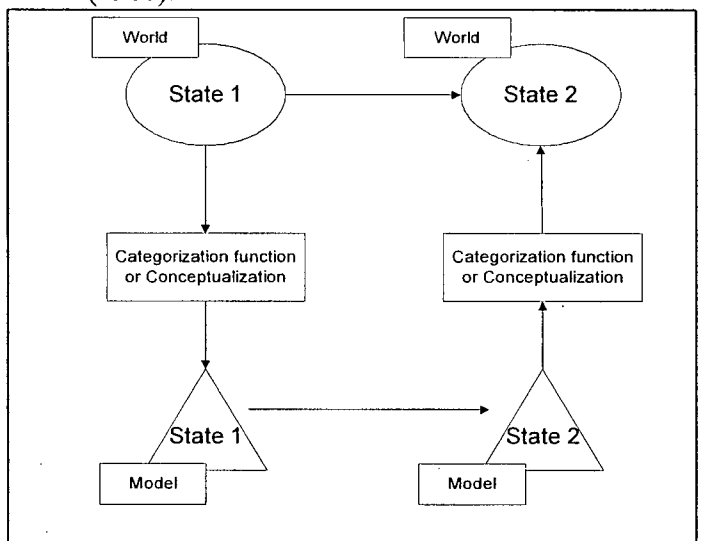

Figure 1: Model / World Congruence (Holland et. al)

As shown in Figure 1, a modeller uses a categorization function (or conceptualisation) to represent the modelled phenomena in the conceptual model. A model has explanatory (and hence potentially predictive) power when its state transitions reflect (are homomorphic to) the state transitions in the modelled domain. If the model's conceptualisation is wrong it will have neither explanatory nor predictive power.

For example, let us say we had a coin and we said that the coin predicts if it is going to rain tomorrow. If it lands heads it will not rain, if tails it will. However, if the conceptualisation is wrong, the state transitions of the model (the flips of the coin) will not correspond to state transitions in the world (whether it rains or not). That being said, it is difficult to know if the "right" conceptualisation has been chosen before testing it, but we can say that if a model is inaccurate, the problem may lie in its conceptualisation.

In the area of agents, researchers have suggested that simulation building be split into three roles; the thematician, modeller, and computer scientist (Drogoul, Vanbergue, & Meurisse, 2002). The thematician creates a domain model which describes how the interactions in that specific domain work, this is also called a theoretical model. The modeller creates the design model, which consists of conceptual agents which are formal refinements of the theoretical constructs. The design model is usually used as a bridge between the modeller and the computer scientist who creates the operational model made up of computational agents used in the actual construction of the system.

A survey of existing agent methodologies (Arazy & Woo, 2002; Shehory & Strum, 2001) shows that many of these methodologies lack a strong focus on the interaction between agents. However, interaction is the main concern for designers of multi-agent-based simulations (Parunak, Savit, & Riolo, 1998). Also, these models are made by computer scientists for computer scientists. This can cause many problems in the development of agent-based simulations which explore areas that are unfamiliar to computer scientists. In these cases, non-technical experts will need to be consulted to create accurate models of the real world environment, or may wish to create simulations that they do not have the expertise to design themselves.

Therefore, the research question is, *how can an intelligent agent conceptual framework, properly defined, be used to model complex interactions in various social science disciplines?*

We chose to focus on the social sciences because the concept of an intelligent agent

for modelling purposes will be most beneficial to those who study human beings. People are intelligent agents, and any methodology which can aid social science researchers to better understand human behaviour would be greatly appreciated by the group.

We propose that a modelling system which will handle complex agent interactions can act as a bridge between the thematician and modeller.

Chapter 2 investigates the current state of intelligent agent modelling and the use of intelligent agents to simulate markets. The area of agent-based computational economics (ACE) is a prime user of cognitive agents that have beliefs about their world and reason about their actions in it (Tesfatsion, 2002).

Chapter 3 defines intelligent agents using a formal method of relating the various agent concepts identified by Arazy and Woo (2002). Since we are interested in using intelligent agents to represent a real world phenomenon or environment, they should closely mimic the reality in which they are based. However, the constructs used in existing physical representations of intelligent agents may not be appropriate to create a model at the conceptual level. To resolve this issue, we rely on the systems approach (Churchman, 1979) and Bunge-Wand-Weber (BWW) Ontology (Wand & Weber, 1990) to guide us in defining intelligent agents concepts. These methods were chosen because they deal with systems that exist in reality and agents have already been defined as a "computer system" (Wooldridge, 2002). Once defined, in Chapter 4 these concepts are converted into a graphical representation and it is shown how this representation may be generally implemented in an existing agent development platform. There is also a discussion on how the representation may lead to a visual interactive programming environment, where non-technical users will be able to build complex, yet accurate, agent-based simulations.

In Chapter 5 we conduct a proof of concept by applying the representation to a problem in the area of marketing, specifically competitive pricing strategy. The model is developed by analysing empirical as well as analytical work done on the topic. The model is then implemented in two programming languages and the results of the simulation are discussed. Along with results from the model, the benefits of the representation are discussed.

Finally, Chapter 6 concludes our work and provides future research direction that can be taken. We also discuss how intelligent behaviour can be better understood by deeply analysing our agent framework.

Chapter 2: Related Work

As mentioned before, agents have been used for various purposes. However, one of the first uses of agents was to model the world and create simulations to explore issues in various research disciplines (Conte, Gilbert, & Sichman, 1998). Agents are useful because they encapsulate not an entire system, but a component of the problem area and can be used in "bottom-up" modelling of an environment (Tesfatsion, 2002). These models can then be turned into simulations of a specific phenomenon, and the resulting interactions between the agents can be studied to better understand the area. Below, we will take a look at how intelligent agents have been used in the area of agent modelling and then at the different agent-based market simulations found in one area of agent simulation, agent-based computational economics (ACE). We will then discuss limitations of agent use in the literature and ways to overcome them.

2.1 Intelligent Agent Modelling

As mentioned previously, various areas have employed intelligent agents in simulation models, however very few research fields have used intelligent agents as a modelling concept. A review of the literature shows that the main users of intelligent agent modelling are supply chain management (SCM) researchers (Nagoli & Bidel, 1993; Kim et. al, 2002). Recently, two SCM researchers (van der Zee & van der Vorst, 2005) have worked on using intelligent agent modelling to act as "a communicative means between the analyst and problem owners". The researchers used a previous conceptual framework for modelling supply chains developed by Swaminathan, Smith, and Sadeh (1998), who wanted to create a set of tools for modellers to build coherent and accurate SCM agent-based simulation models. The framework is split into two major components, structural and control elements. Structural elements are the agents in the supply chain (retailers, distributors, etc.) while control elements are the ways in which the actors influence each other and achieve their goals (contracts, forecasting, routing, etc.).

According to Swaminathan, Smith, and Sadeh (1998), an agent is composed of attributes (which together make up the agents state), knowledge, objectives, commitments, a "selector", priorities and incoming and outgoing messages. Figure 2, shows how the components are interrelated.

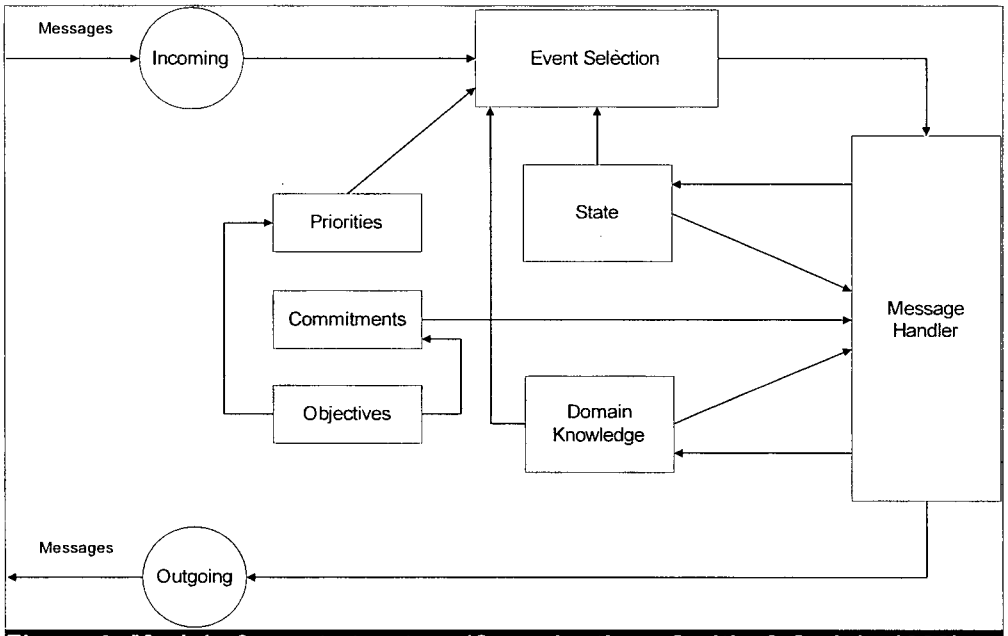

Figure 2: Model of agent structure (Swaminathan, Smith, & Sadeh, 1998)

According to Figure 2, an agent receives messages which are prioritized using event selection which is affected by the state of the agent, its priorities, and its domain knowledge. Once prioritized, the message is sent to the message handler which decides what to do with the information. How the agent handles the message depends on its state, domain knowledge, and commitments (which are affected by its objectives). In turn, the message handler creates outgoing messages to other agents which may affect the agent's own domain knowledge or state.

Using the framework developed by Swaminathan et. al, van der Zee and van der Vorst developed their own visual modelling tool to help analysts communicate the model to problem owners. This visualisation is shown in Figure 3.

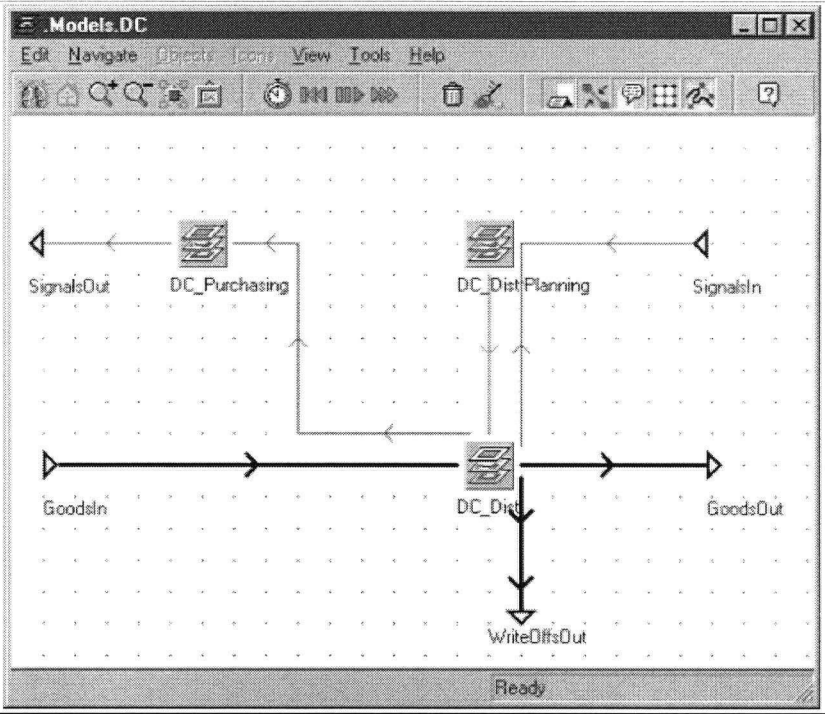

Figure 3: Agent Modelling of Supply Chains (van der Zee & van der Vorst, 2005)

The model focuses on the inputs and outputs (physical and information) of the agents. Agents are also given jobs, which detail the services that they can perform, and the inputs and outputs that they own. Unfortunately, since the model is made to develop supply chains, certain hallmarks of intelligent agent design, like reasoning and learning are not shown in the model. The emphasis is on the inputs, outputs, and jobs of the various agents.

The field of agent-based computational economics (ACE) is also using agents as a modelling concept. However, these researchers say that the simulation itself acts as a model of the concept (Kirman & Vriend, 2001). They have not, as of yet, developed a conceptual framework to develop their models. In this thesis we will use the area of ACE as a test of the proposed conceptual framework. However, to understand this field of research we will highlight some of the different simulations currently used in the field.

2.2 Market Simulations

The ACE area was developed to understand decentralized markets (Tesfatsion, 2002), specifically, how interactions on the micro level lead to macro economic phenomena. ACE researchers use simulation models to build virtual economies from the ground up to

investigate the formation of market norms, or to design agents for use in automated markets. To gain a better understanding of how these researchers view intelligent agents, we will go through three examples of market simulations. To give us a good idea of the field, each of the three models will focus on a particular component of a market. The first model is very consumer-centric and involves a network of consumers influencing each other, and indirectly influencing the retailers. The second model is more retailer (producer) oriented, where consumer agents only try to attain a certain amount of goods and do not directly influence each other. The last model takes a more balanced approach to market conceptualisation by modelling an actual, but simple, market in the real world. With an investigation of the model, the tools used to conceptualise and communicate the design of the simulation will also be discussed.

2.2.1 Customer Behaviour Simulator - CUBES

The CUBES model was developed by Ben Said, Bouron, & Drogoul (2002) who wanted to develop a marketing simulation that took into account the myriad reasons why consumers buy a product. Specifically, they used theories based on the diffusion of innovation to develop a system where agents have behavioral attitudes (BA) which eventually lead to consumption decisions.

Behavioral Attitudes are affected by the agents' interactions with each other, the "outside" elements of the system (businesses, advertising), and the actions of the consumer themselves (what they do and do not buy). Each of the consumer agents also have a social network of family and friends, which it trusts more than others. There are also opinion leaders who can set the trend in the market. Figure 4 shows how consumers make decisions in the CUBES model.

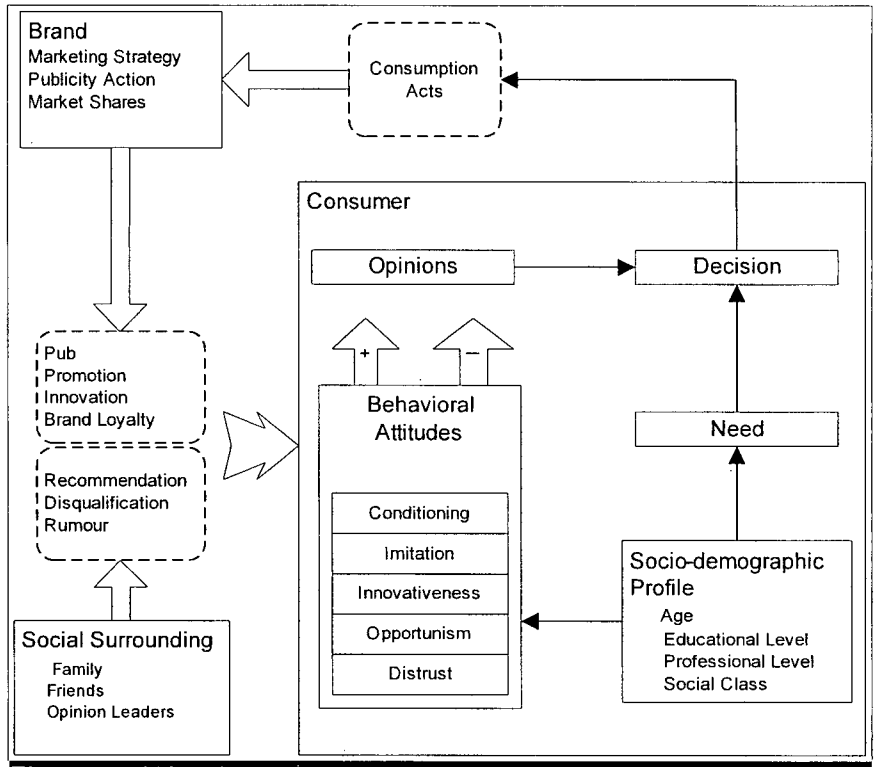

Figure 4: CUBES components (Ben Said, Bouron, Drogoul, 2002)

The authors tested the model by running a simulation of "old" consumers and "young" consumers in a three-brand market. The researchers assumed that young consumers had frequent and intense communication, while old consumers had low intensity and infrequent communication. They found that increased communication led to more destabilized behavioural attitudes. They also ran an experiment to investigate the effect of differing starting positions for brands. They found that if in an initial population a brand was dominant (they had their dominant brand at 70% market share), the dominant brand would on average maintain its market share. In this model, a dominant brand would have a more favourable opinion amongst consumers than a non-dominant brand, so this means that dominant brands start off not only with a market share advantage, but an opinion advantage as well. However, if the brands started at equal positions of market share (and customer opinion), they would engage in heavy competition where at one time or the other, one brand would be the market leader.

Unfortunately, the only design model that the researchers have given is the model shown in Figure 4. It presents the components of the consumer and "brand" (manufacturer) agents, however we are not given any information of how these components work together,

especially in the brand agent. Also, the modelling constructs in Figure 4 are vague, what is the difference between the various types of arrows and boxes in the diagram? Another problem is that it is hard to fit these components into the categories of agent components (reasoning, learning, beliefs, knowledge, etc.) identified by Arazy and Woo (2002).

2.2.2 ACE Trading World

Tesfatsion (2005) developed a simulation model to investigate how market clearing behaviour occurs without artificial pricing mechanisms. For years, many economic models had used a "Walrasian Auctioneer" as a pricing mechanism to make sure that all demand in the market was satiated. The mechanism matches buyers to sellers and also sets the price. However this does not happen in the real world. Firms set their own prices, and consumers search for their suppliers. Tesfatsion, then develops an economic model where buyers and sellers interact "face-to-face". She creates a "hash and beans economy" where the economy is made up of hash producers, bean producers, and consumers who gain utility from consuming hash and beans. Producers who lose a certain amount of money go out of business, consumers who can not attain enough hash and/or beans, die. Producers must make the decision about how much of their product they will offer and at what price. As the simulation runs, they learn what is the best price and best product offering to create the highest profit. She goes on to show that the ACE framework has the ability to answer questions that the Walrasian Auctioneer model can not address (strategic rivalry, market power, learning, etc.) and gives a general description of the benefits of ACE modelling.

The tool used to communicate the design of these agents, other than written descriptions, is a set of pseudo-code. Below is a pseudo-code conceptualisation of a firm in the simulation.

```
agent Firm
{
Public Access:
// Public Behavioral Methods
getWorldProtocol(ownership of stock shares);
getWorldProtocol(collusion among firms);
getWorldProtocol(insolvency of firms);
getMarketProtocol(posting of supply offers);
getMarketProtocol(trading process).

Private Access:
// Private Behavioral Methods
```

```
Methods for sending, receiving, acquiring, and storing
data;
Method for selecting my supply offers;
Method for supply rationing when my demand exceeds my
supply;
Method for recording my sales;
Method for calculating my profits;
Method for allocating my profits;
Method for calculating my net worth;
Methods for changing my methods.

// Private Data
My money holdings, capacity, total cost function, and
net worth;
Data about world attributes;
Data about past and current world events.

Protected Access:
// Protected Data
My address book (communication links);
My recorded communications.
}
```

The code shows the functions that occur within each of the agents and details the public, private, and protected data the agent possesses. Tesfatsion also shows a dynamic model of how the simulation runs using pseudo-code, with procedures detailing which actions occur in the simulation and in what sequence. However, this code is cumbersome and other than data permissions, it is difficult to see how the various agents in the system interact. Another problem is that non technical individuals may be confused even by this simplified version of the simulation code. Once again, even though this simulation uses intelligent agents, there is a disconnect between the intelligent agent concepts outlined in Arazy and Woo (2002) and the representation of agents found in this study.

2.2.3 Marseilles Fish Market Simulation

This study by Kirman and Vriend (2001) models the dynamics of the fish market in the French city of Marseilles. The researchers specifically are trying to discover why there is price dispersion (different prices for different sellers) and loyalty (buyers go to the same sellers everyday) at the same time in the market. They first go through common economic theory reasons why this would occur (quality differences and reputation effects, implicit contracts, etc.), and find that each of the explanations are lacking. They then propose to develop an ACE model of the situation and run the simulation to see if the same characteristics that were found in the fish market occur in the agent-based model.

There are two types of agents in the simulation, buyers and sellers. Sellers receive their fish at a constant price, have to decide how much fish they will stock for the day and the prices they will charge to customers. Sellers also can gain a minimal amount of money for fish not sold on that day. Buyers in the market are usually resellers and know the price they will be able to sell their fish. Buyers also have two chances per day to come to the market, the morning and the afternoon. However there is no replenishment of goods between these times, meaning that if all the fish are gone in the morning, the afternoon customers will have no fish to buy.

The trading interaction between buyer and seller is a "take-it-or-leave-it" price determined by the seller. Buyers can decide to accept the price or reject it and come back later to find another seller with a better deal. Buyers get to trade by queueing in front of their selected seller at the beginning of each session. The queues are "Italian", meaning that the seller has the right to pick who gets to be served next, it is not done on a "first come first served" basis. The modellers then say that a seller can decide to choose randomly, pick familiar customers first, or deal with repeat customers last. As the simulation continues, buyers begin to learn which sellers give the best price/selection and sellers learn which prices, supply offers, and queue management brings the highest profits.

The researchers ran the simulation and found that after 2500 turns, the average price asked by the sellers in the morning session was very close to the average price accepted by the buyers. They also found that there was price dispersion. Even though a commodity was being sold, buyers were likely to end up paying 10, 9 or 8 "credits" per fish. They also found that loyalty occurred. Buyers were willing to go to the same sellers, because sellers were likely to give an advantage to loyal buyers. It should be noted however that this was not set into the simulation but emerged from the agents learning to do what was in their best interests. The researchers then varied their model slightly by creating buyers who could sell their fish at high or low prices (the difference between a supermarket and a restaurateur, for instance). They found that agents who could resell their fish for higher prices frequented only sellers who gave queue advantages to loyal customers. They also found that "high-price" buyers were also more likely to switch between these "high-quality" sellers, but the low-price sellers were more loyal. Also sellers were more likely to ask for higher prices from non-loyal customers than loyal customers.

This study also used pseudo-code to represent the simulation but it is much more detailed than what is presented by Tesfatsion (2005). However, the code is not split into static and dynamic structures but is presented as a full program with procedures for creating agents and

guiding the actions of those agents. Below is a sample of code detailing how seller agents choose the supply level that they want for the next turn.

```
procedure CHOOSE_SUPPLY_LEVEL;
begin
    with all supply levels from 0 to 30 do normalize
    actual fitness to [0, 1]
    for all supply levels from 0 to 30 do
    begin
        bid:= normalized_fitness + e where e ≃ N(0, 0.10);
        with probability 0.025 ignore bid;
    end;
    choose supply level receiving highest bid;
end;
```

As previously stated, the code is much more detailed, giving the mathematical functions involved in the intelligence of the agents. However this brings many different problems. First, it is hard to see if the conceptualisation of the agents was correct using only the pseudo-code. Since the code is dynamic, we would first have to identify which procedures belonged to which agent, then identify what sort of concept the mathematical formulas were trying to represent and build a design model of the simulation from the pseudo-code given to us. In essence we would be working "backwards" from an implementation model to a conceptual one. Another problem with this detailed pseudo-code is that it makes it even harder for non-technical theoreticians to understand and analyse the proposed model.

2.3 Discussion of Previous Work

The above examples show that ACE modelling has been useful to practitioners and may provide insights into the fields of marketing and economics. However, there are some limitations with the studies. The agent modelling framework and visual interactive programming environment by van der Zee & van Der Vorst (2005) is impressive, however it is made specifically to be used in SCM research and may not be suitable for other purposes.

Also, it has been shown that the explanatory devices currently used in the ACE literature to design simulations are deficient, since they do not; 1) clearly define the concepts of intelligent agency used in the model, 2) are geared towards an implementation model, or 3) include too few details to accurately gauge the veracity of the simulation's conceptualizations. Put simply, there is no generalized conceptual model of intelligent agents in the ACE literature.

However, even if we were to accept the view of some ACE researchers that the

simulation model itself is a conceptual model of economic phenomenon, it can not be said that the models in section 2.2 are the definitive ways to model markets. For example, in the CUBES model the evaluative process for consumer agents is not specified. Opinions are derived by interaction between other agents' opinions and agent personality traits rather than product specifications. This means that more detailed reasoning of product choice can not be realized in the CUBES model. Also in the CUBES model, behavioural attitudes are set, which may not work for every particular situation. The ACE Trading World also has its problems since it does not create mechanisms or institutions for the consumer agent to receive the product from the manufacturer agent (Tesfatsion, 2005). The inclusion of supermarkets or other retail stores would further develop the realism of the procurement process. Also, as mentioned in Kirman & Vriend (2001), the Marseilles fish market study was meant to specifically deal with that problem and may not be generalizable to other industries or countries. We also have the problem of the simulation code being too complicated for communicative and analytical purposes. Therefore, it would be beneficial for the field to have a conceptual model which could be implemented as an information system, and could be used to communicate the basic workings of the simulation to non-technical theoreticians.

Chapter 3: Model Development

As mentioned in Chapter 1, more research is needed in the area of agent-based simulation and modelling. The largest problem with the studies discussed in the previous chapter is not their lack of generalizability but that even though the studies are related, there is no current way to show the interrelations between the various models. As shown in Chapter 2, some studies present pseudo-code of the simulation to show the reader the specifics of their models, however the code does not easily show how agents interact. Also, the structure of the pseudo code in the two studies is different, since the researchers decide to highlight two different aspects of their models. What is needed is a more general conceptual model of intelligent agents that can be used to analyse problems not necessarily in a limited field (such as supply chain management). There are however more conceptual reasons for developing a general model of agents.

In Arazy and Woo (2002) it is stated that "there is no theory to guide the selection of constructs and models for representing" agent components, however this has not stopped the use of agent concepts. Some of the current terms used to describe intelligent agent structures and

processes include goals, beliefs, knowledge, reasoning, plans, and learning.

While these terms are commonly used, different definitions are applied for each concept (Wooldridge, 2002). This leads to unclear, general definitions that try to include all idiosyncratic meanings of the terms. Even though the general concepts are vaguely described, it is apparent that they are interrelated. For example, learning, by definition, has some effect on beliefs (Arazy & Woo, 2002). Therefore another reason for creating a general model of intelligent agents is that it would clearly state the definition and interrelation of the agent concepts.

Intelligent agents are used to represent phenomena in the real world, therefore our methodology must not only allow us to understand intelligent agents, but also make sure that these agents accurately reflect the world. One theory which can help us achieve this is systems thinking. In Arazy and Woo (2002), the agent concepts were split into structural and dynamic concepts. One of system theory's main strengths is the ability to identify the components of a phenomenon (system) and study how those components work together (Miller, 1978). It is proposed that intelligent agents can be conceptualised as systems, and that the systems approach can be used to refine and define intelligent agent concepts. Once we know the components of the system we will use the BWW ontology to understand the processes of the system. We expect that because the BWW ontology deals with reality that it will aid in the construction of the agent framework since we want to use agents to represent phenomena in the real world.

3.1 The Agent System

One of the main theses of systems thinking is that all systems have a purpose or objective. Once that objective is found, then the different components of the system that work to this goal can be identified (Churchman, 1979). A system is conceptualised as "a set of interrelated elements" (Ackoff, 1971), which means that to define our system we must define its "sets of elements" and their interactions.

Wooldridge (2002) provides a starting point with a graphic representation of agent behaviour, shown in Figure 5.

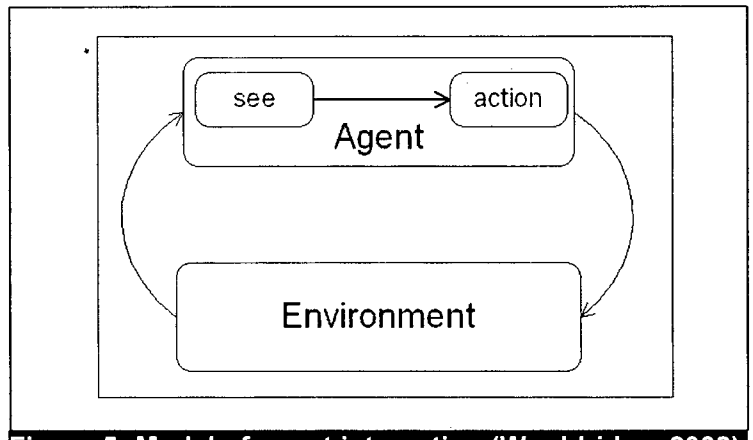

Figure 5: Model of agent interaction (Wooldridge, 2002)

Figure 5 shows the agent as a system. The agent takes in some input that is then transformed into some type of output.

But what exactly are the sets of elements within a system? One definition is that a system has a measure of performance (objective), an environment, components (parts), resources, and management (Churchman, 1979). The environment affects the system but lies outside of it. The resources are "things the system can change and use to its own advantage", components perform the "various kinds of activities performed within the system" (Churchman, 1979), and management is how the system decides on the ways to transform inputs into outputs. In Figure 5 the inputs (changes in the environment) "go in to" the see function, are observed, and then some action takes place. So Figure 5 has partially shown the components of the agent system but has not given an objective or a management process for it. To show how the "observations" change into "actions" we can use systems theory concepts.

An intelligent agent according to Wooldridge, should be adaptive, reactive, communicative, and autonomous. By adaptive he means that an agent should be able to change its state without prompting from the environment. Reactivity is the capability of the agent to react to the environment around it. Also, it should have the ability to communicate with other agents in the environment. And lastly, the agent must be able to act independently (autonomously). Therefore, if we break the system down into components, it means that the agent needs to read the environment, "think" about how to achieve its goal, and then have the ability to interact or communicate with the environment, to be effective.

This is very similar to the idea of a feedback system shown in Figure 6.

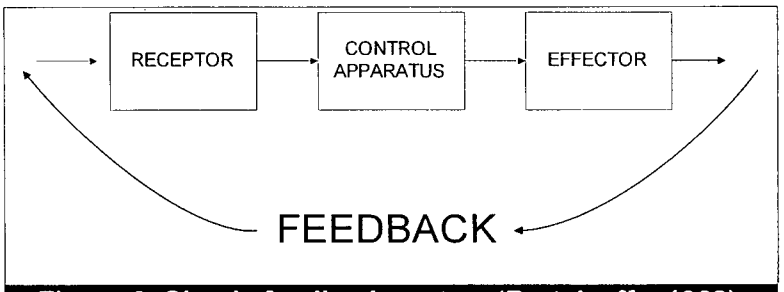

Figure 6: Simple feedback system (Bertalanffy, 1968)

Figure 6 describes a control system as comprising; a sensing component called the receptor that receives input from some feedback mechanism, a control apparatus that decides how the system should act based on feedback, and an effector which changes the environment. This system has objectives, an environment on which it acts, components (the receptor, control apparatus, and effector), and the control apparatus acts as the management of the system. What makes feedback systems unique however is that their inputs come from the environment itself which are then handled by the control apparatus and a response is manifested by the effector. In short, the control apparatus tells the effector what to do. Note the similarity to the model of an agent in Figure 5. However, we still need to clarify what is the control apparatus in terms of intelligent agent concepts.

According to Ackoff (1971) there are two types of systems, abstract and concrete. An abstract system is a system made up of concepts. For instance the system of "culture" could be considered as an abstract system (Miller, 1978) if you define culture as the interplay between different social roles. These roles (mother, son, priest, leader, follower) are all concepts and do not have to be based in space or time. However a concrete system is in a "region in physical space-time" (Miller, 1978).

Systems can also be described by the type of behaviour(s) they exhibit. For instance, a system which does not change at all is classified as a static system and one that does is a dynamic system (Ackoff, 1971). However, dynamic systems can be further classified, based on how they change. State-maintaining, goal-seeking, purposive, and purposeful systems are all dynamic systems.

State-maintaining systems are dynamic but only change to maintain the state of itself or the environment. Think of a system which maintains a certain level of light in a room. When it gets dark outside it would increase the light level, as it gets lighter the level of artificial light would decrease. If the natural light was higher than the state maintenance level, then the agent

would close the blinds.

Goal-seeking systems however are different from state-maintaining systems because they learn. A goal seeker will react one way to a change in the environment and measure if the reaction was "good", depending on what its goal is. The system is given various ways to react to a stimulus and must choose the correct one.

A purposive system is a multi goal-seeking system with a purpose. A purpose is a common property between the multiple goals. An agent that is programmed to win at chess and checkers would have a purpose of winning at board games.

A purposeful system is just like a purposive system except that it can change its goals based on internal matters to meet an objective. The difference between a goal and an objective, according to Ackoff (1971), is that an objective is something that can not be achieved in a specified amount of time. For instance, if we wish to have a good meal next Sunday we can not achieve that now. We need to set and achieve specific goals (decide what to cook by Wednesday, buy the food on Friday, prepare the food on Saturday). Deciding what to cook is a goal, eating a good meal on Sunday is an objective. This can be illustrated by using the game playing agent example again. If the agent was able to play and win in several different games then it would be a purposive system with the purpose of winning board games. If however you created a purposeful system with the objective of winning at every single board game, then once it achieved the goal of winning a game of checkers, the system would then try to play and win a game of chess.

An intelligent agent is a purposeful feedback system which has an objective, learns from the environment, and chooses the best process (combination of goals) to achieve that objective. However there is a fine distinction between goals and actions. For example, suppose that an agent had the objective of watching a football game. The set of goals could be as follows:

1) Turn on TV
2) Turn to Channel 5

The actions necessary to achieve these goals are almost the same as the goals themselves.

This definition of an intelligent agent is very similar to the description of work already done in the systems field by Miller (1978) on simulated behaviour. In his work, the first components are the input transducer and decoder which perform the same functions as the receptor in the general feedback model shown in Figure 6. The input transducer receives information from the environment, and the decoder transforms the input into something that is

usable to the associator and the memory components. The associator has the ability to learn and make associations between various signals decoded by the decoder. These associations are put into memory and sometimes existing memory is used to make new associations. Signals are then sent to the encoder which has the function of transforming the internal signals of memory and the associator to be usable by the output transducer, which produces output.

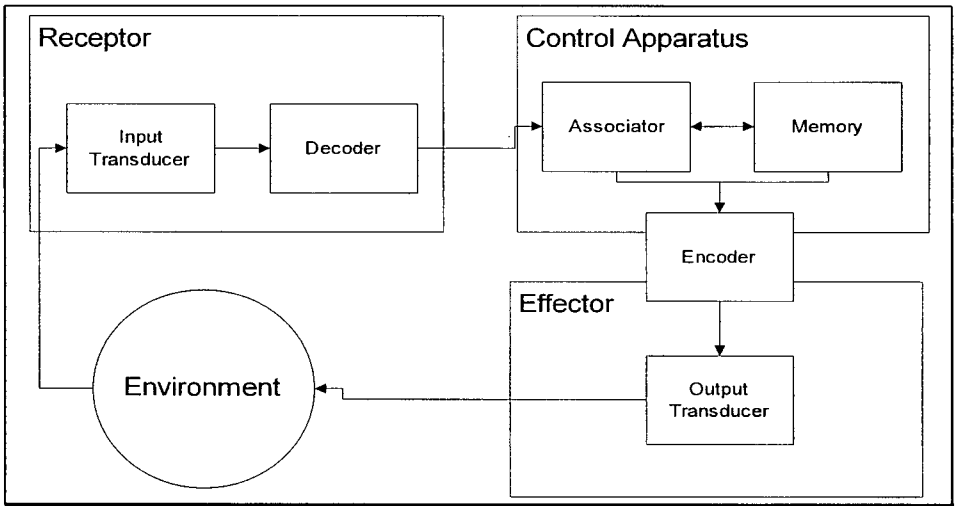

Figure 7: Subsystems of simulated intelligent behaviour (Miller, 1978)

We will use Miller's concepts to help us identify the constructs for our conceptual framework. We propose that on a conceptual design level, the input transducer and decoder can be taken as given. They will be designed to accurately interpret specific variables in the environment. However, the operations of the associator and memory subsystems, as well as the functions of the encoder and output transducer (which make up the *effector*) need to be specified in the design phase. We term the combination of associator and memory as the *simulator*, as it in effect needs to mimic phenomena in the controlled domain. The two proposed components (simulator and effector) are actually subsystems which themselves need to be investigated.

Using systems theory, we have been able to identify the necessary components for an intelligent agent to perform rational behaviour. To be able to perform a reasoned action in its world, the agent must be able to model its environment and put its "thoughts" into action. However, as Miller shows, to fully understand a system, its components' processes also need to be investigated. To gain a better understanding of the simulator and effector, and to ensure that the agents match their real world counterparts, we will use the ontological approach. We expect

that the ontology will help us build a coherent modelling framework and develop clear modelling constructs. We also want to use the ontology to add semantics to the interrelationships, configurations, and transitions of the simulator, effector, agent, and environment.

3.2 Ontological Definition of Intelligent Agents

Ontology "deals with modelling the existence of things in the world". Using the ontological approach, we hope to define and show the relationships between the agent system and its subsystems. Specifically, we will use an ontological approach which is an adaptation of Bunge's ontology (1977, 1979) to information systems (Wand & Weber [1990, 1995]). This ontology provides specific constructs for defining information system components and processes, and includes many of the concepts found in systems theory, making it useful for modelling the components and processes of a system. The reasons for using this ontology are that it is highly formalized and was specifically developed to represent information systems. The ontological concepts and premises will aid us in creating clear relationships and definitions of intelligent agent concepts. Also, an agent system is an information system, therefore, these ontological constructs should have some equivalence with the agent concepts.

To help explain these concepts and the process by which they will be derived, suppose that there is a farmer who raises sheep. Let us also say that the farmer has sheepdogs to help herd his flock from a nearby grazing area back to their pens on the farm. We will use this scenario to explain the BWW ontology and agent concepts throughout this chapter.

3.2.1 Overview of BWW Ontology

3.2.1.1 Things

In the BWW ontology, a thing is the "elementary unit of [the] ontological model". In Bunge's ontology, the world is made up of things which can be primitive or composite. A primitive thing is a thing that exists on its own while, a composite thing is comprised of other things.

We propose that an intelligent agent can be considered a thing in Bunge's ontology. According to Bunge's ontology, the world is made up of things and according to agent literature, agents are part of the overall environment. In fact, the agent can be considered to be a composite

thing consisting of a simulator and effector which are primitive things. Even though we said previously that the components outlined by Miller could be subsumed into the simulator and effector, it does not mean that the simulator and effector are composite things as well, they still can be considered primitive things. In the definition of system components by Churchman, every component performs a specific activity within the system. However, that is not the case in the BWW ontology. As we shall later see, multiple processes can occur within the same basic thing. Therefore we propose that the activities of the associator, memory, and encoder are translated as processes and specific configurations of the thing called a simulator, and the activities of the output transducer becomes a process of the thing called an effector.

The simulator is a model of the agent's world, including information on itself and other agents. According to the ontological terminology, the simulator is "coupled" to other agents (in its environment). When things are coupled it means that they can affect each other.

The effector is also a thing and has the ability to change the agent's world and is therefore coupled to world. The simulator is also coupled with the effector and has the ability to change it. A change in the "world" can affect a change in the simulator, which results in a specific "type" of the simulator. Then a "check" occurs to make sure that the type is correct. If this function holds true then it is possible that there will be a change in the effector which has the consequence of leading to a change in the world.

Using our farm example, suppose that there is a change in the world, specifically that a wolf comes into the farm, and also suppose that our farmer can be considered to be an intelligent agent. When the wolf enters the farm, the farmer's simulator (his model of the world) would change if it is "correct" to believe that a wolf could enter into his farm. This would then lead to the farmer trying to affect the world, for example, shooting at the wolf to scare it away. By shooting at the wolf, the farmer has changed the state of the world.

Yet this is all very vague, what is the definition of "correct"? When the simulator changes type, what exactly does that mean? We will refine the above model by exploring more aspects of the BWW ontology.

3.2.1.2 Properties

According to the BWW ontology, things possess properties (Wand and Weber, 1993). When properties are viewed by humans they are referred to as attributes, individuals

usually know a thing by its attributes. An example of an attribute for the thing "Inventory item" could be "item number" (Wand & Weber, 1990). Since the simulator is equivalent to a representation of the world, some of its properties represent beliefs about the world (there is a desk in the room, the sky is blue, bees exist, etc.). Properties are also split into intrinsic and mutual properties. Intrinsic properties are properties which belong only to the thing, and a mutual property is a property that two things share.

In the simulator, an intrinsic property might be a belief about the world. We will discuss mutual properties later since they are better left to a discussion on dynamic processes.

As a thing, the effector also has mutual and intrinsic properties. One of the intrinsic properties of the effector could be its capabilities, or the internal aspects it uses to change the state of the world.

To make this clearer we will use the farmer example. As stated previously, the farmer is the agent, and his simulator is his model of the world. In the case of the farmer shooting at a wolf, the effector would be the farmer's hand which he uses to hold the gun and pull the trigger. The effector also has mutual properties between it and the simulator as well as between it and the rest of world, however as stated previously, the implications of this will be discussed later.

3.2.1.3 States and laws

According to the literature, states can be defined as an attribute function, a state is a collection of values of various properties. This can be shown as $s(t) = \{x_1, x_2, x_3....x_n\}$ where s(t) is a state of a thing and x_n is the value of an attribute of a thing, (Wand & Weber, 1990). Imagine our farmer has two attribute functions:

1) body_temperature(farmer) = 30°C
2) disease(farmer) = "fever"

The two statements above are property functions which denote a state, but based on our knowledge, they can not be paired together since a person can not have a fever and a temperature of less than 37°C. This is obvious from a "lay" point of view but the BWW ontology also has a term to reflect this, called a law. A law is a specific type of property which restricts the property function for a given state. Biology tells us that the normal temperature for a human is 37°C. In ontological terms, according to a law (biology), a thing (the farmer) can not have an

attribute value (30°C) if it is in a certain state (fever).

For the simulator, since properties represent beliefs about the world, a state would simply be the combination of these beliefs to describe the agent's view of the world. Going back once again to our farm example, let us say that the farmer sees fresh wolf tracks on the farm. His model of the world could lead to the state, "a wolf is at the farm" and then could lead to another state, "a dangerous world", if there is a law for this transition. If the world were safe (no predators, wolf or otherwise) then the farmer should not think that he is in "a dangerous world". We have alluded to the fact that a specific state may lead to another state. How states change is also covered in the BWW ontology by the transition law term, but there are also types of states that change, depending on different circumstances, these states are called stable and unstable states.

These are just like the states described above, however there is a question of transition. Changes of states are called events and will be discussed later. However, in the case of a stable state, it will change only if there is an "external event". An unstable state changes when intrinsic or mutual properties are changed and is called an "internal event".

Common sense tells us that if the state of the farmer's simulator is in the "a dangerous world" state, then most likely he will want to get to some other state, a "safe world" state. Once he thinks he is in danger, the farmer will figure out ways in which he can be safe, meaning that intrinsic properties of the simulator will be changed until he can get to that state.

For example; he would think about what would happen if he did nothing, if he went inside the farmhouse, if he were to shoot at the wolf tracks, etc. Once the farmer has thought of a scenario which would lead to the "safe world" state, his simulator will change the mutual properties between it and the effector to create a "safe world". Once he is in a "safe world" state however, the farmer will not try to take further action until something happens to make the world dangerous (an external event). Therefore, the "dangerous world" state is an unstable state and "safe world" state is a stable state.

The effector also has states. As said previously, the farmer's hands is his effector. If the farmer's finger is on the trigger, but not pressing down, then it is in the "ready to shoot" state. Stable states would be considered states of "rest" where the effector does not have to move unless the agent, through its simulator, "intends it to". An unstable state of the effector would be one where the effector must move to another state based on its own intrinsic properties. For example, if the farmer shot the gun his hand would move because of the recoil. This can only be

stopped if the simulator tells the farmer's hand to be firm when shooting the gun, this would be an external event of the effector.

3.2.1.4 Events and transformations

Transitions between states in the BWW ontology are called events and the mechanism that creates an event is called a transformation. A transformation is defined as a mapping from one domain comprising of states to a co-domain comprising of states (Wand & Weber, 1993). In other words, the transformation acts as a function. This was shown by Wand and Weber (1990), by saying that if s and s' are an ordered pair of states, with s preceding s', and if g is a transformation, s' = g(s).

If the temperature (property) of water (thing), changed from -50°C, to +10°C then it moved from a frozen state to an unfrozen state. The change in state would be an event but how the temperature moved the water from a frozen to unfrozen state would be a transformation. This also holds for the agent when its status changes.

In the simulator, a change in state is a change of the simulated world of the agent. And since a change of state can only occur if there is a change in properties, an event in the simulator is the changing of certain beliefs about the simulated world. If our farmer perceives a wolf on the farm, the state of the simulated world eventually changes from "a safe world" state to "a dangerous world" state. This means that the farmer has "learned" that the world is dangerous.

The BWW ontology also differentiates between two kinds of events, internal and external. Internal events are events that can occur only in the case of an unstable state and mean that properties are changed by the thing itself. An external event is an event where the mutual properties of an agent are changed by a coupled thing. As stated previously, mutual properties are properties that are shared by at least two things. When these properties are changed by one thing they are also changed in the coupled thing. This means that an external event can only occur in a thing when something other than itself changes a mutual property. This is important in the simulator since a differentiation can be made between "learning" based on observations of the world and "learning" based entirely on internal mechanisms. In the situation of the intruding wolf, the farmer "learns" that a wolf is near by observing the wolf entering the farm. This is an external event since it is based on the "interaction" between the world and the simulator. On the other hand, when the simulator's state changes from "dangerous" to "safe" world by shooting at the wolf, then it can be considered an internal event, since no other interaction was needed for the

farmer to begin shooting.

Since we have been talking about events though, there must be some sort of transformation which causes events in the simulator. In the simulator, transformations which lead from a lawful state to another allowed lawful state (lawful transformations), can be considered the "rationalization" of the agent's "beliefs" of the world. For if beliefs comprise states of the simulator, a lawful transformation of the simulator would detail how an agent moved from one set of beliefs to another.

The change of a state in an effector is also an event. When the farmer's finger moves from the "ready to shoot" state (finger on the trigger but not pressed down) to the "shot" state (finger pressed down on the trigger), there is a change in the properties of the farmer's effector. The moving from the "ready to shoot" to "shot" state could be considered as the act of shooting.

Table 1 is a summary of the ontological constructs discussed and the symbols we will use to define agent behaviour.

Table1: Ontological constructs used in the definition of agent concepts		
Ontological Construct	**Definition (from Wand & Weber, 1995)**	**Symbol**
Thing	The elementary unit in the BWW ontology. A composite thing may be made up of other things (composite or primitive).	T
Property	A property is modelled via a function that maps the thing to some value. A property that is inherently a property of an individual thing is called an intrinsic property. A property that is meaningful only in the context of two or more things is called a mutual property.	Intrinsic: p(T) Mutual: p(T1,T2)
Attribute	An attribute is a human representation of a property of a thing.	A_p
State	The vector of values for all property functions of a thing.	$s(T)= A_{p1}(T)...A_{pn}(T)$
State space	The set of all possible states	S
Transition Law	The rules governing the changes of state over time	$L: S \rightarrow S$
State law	Restricts the values of the property functions of a thing to a subset that is deemed lawful because of natural or human laws.	L_T
Event	A change of state of a thing. It is effected via a transformation.	$e(T)=<s_1,s_2>$
Transformation	A mapping from a domain comprising states to a co-domain comprising states.	$g(s) = s'$
Lawful Transformation	Defines which events in a thing are lawful.	
Stable State	A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment.	$L(S)=S$
Unstable State	A state that must change into a new state.	$L(S)\neq S$

3.2.2 Formal Ontological Description of Intelligent Agent Behaviour

Now that some of the BWW ontology has been explained, we can begin to define intelligent behaviour.

Let W be the agent's world, Sim be the simulator, E be the effector, and let "→" mean "cause". According to our model, a change in the "world" can affect a change in the

simulator, which may cause a change in the effector. This in turn, will lead to a change in the world.

More formally:

Event (W) $\rightarrow$ Event (Sim) $\rightarrow$ Event(E) $\rightarrow$ Event(W)

The way a change in one thing causes a change in the other can be modelled ontologically as follows: assume an event occurs in W. If as a result, a mutual property of W and Sim changes, then Sim might reach an unstable state which will cause it to change again. This new change might cause a mutual property of Sim and E to change. Thus, a change in W will be propagated to a change in E via Sim. How Sim and E will change states, depends on their transition laws. Specifically, for a given thing, an event starting in an unstable state is determined by the transition law of the thing:

$e(T) = \langle s(T), s'(T) \rangle$ where $s'(T) = L_T(s(T))$ or, briefly :: $e = \langle s, L(s) \rangle$ where e, s, L all relate to the same thing.

The propagation of changes can be modelled as follows:

$e(T1) = \langle s_1(T1), s_2(T1) \rangle$

As mentioned previously, when a thing changes states it is called an event and a state is comprised of attributes, so for an event to occur a set of the thing's attributes must change. Let us say that C(e) denotes the set of attributes that change in an event e. A change will propagate from a thing T1 to a thing T2 iff there exists a mutual property p(T1,T2) such that $A_p \in C(e(T1))$. Specifically, for our model of agents, the propagation of changes can be described below. However, we skip the concept of "sensing" the environment, since as mentioned in section 3.1 we assume that the receptor will accurately interpret signals from the environment:

(1) An event e(W) occurs.

(2) For e(W) $\exists p(W,Sim) \in C(e(W))$ and the state of Sim after e(W) is unstable bringing about an event e(Sim): $e(W) \rightarrow e(Sim)$.

(3) For e(Sim) $\exists p(Sim,E) \in C(e(Sim))$ and the state of E after e(Sim) is unstable bringing about an event e(E): $e(Sim) \rightarrow e(E)$.

(4) For e(E) $\exists p(E,W) \in C(e(E))$ bringing about an event e(W): $e(E) \rightarrow e(W)$

For the agent to achieve its goal, its effector has to be given the "correct command" (namely, a change of a mutual property) by the simulator to bring about the right change in the world. The ability of the simulator to do this is modelled by its transition law.

The above description uses ontological concepts to model an agent's behaviour. However, computer and cognitive science researchers refer to the actions of intelligent agents using different terms. Therefore, we seek to link our ontological description of agent behaviour to concepts previously used in the agent literature. Table 2 is an overview of the notation used throughout the thesis.

Table 2: List of Notation

Notation	Explanation
e(W)	Event in the world
e(Sim)	Event in the simulator
e(E)	Event in the effector
A(W, Sim)	Mutual attribute between the world and the simulator
A(Sim, E)	Mutual attribute between the simulator and the effector
A(E, W)	Mutual attribute between the world and the effector
p(W, Sim)	Mutual property between the world and the simulator
p(Sim, E)	Mutual property between the simulator and the effector
p(E, W)	Mutual property between the world and the effector
C(e(W))	A set of attributes that change when e(W) occurs
C(e(Sim))	A set of attributes that change when e(Sim) occurs
C(e(E))	A set of attributes that change when e(E) occurs
p(Sim)	An intrinsic property of the simulator
A(Sim)	An intrinsic attribute of the simulator
S_W	State space of the world
$L_W(s) \neq s$	Unstable state of the world

3.2.3 Ontological Description of Intelligent Agent Concepts

The model in section 3.2.2 uses the two concepts, simulator and effector, to illustrate the intelligent agent system. However, this is not usually how computer and cognitive science researchers refer to the actions of intelligent agents. Therefore, the model must be linked to concepts previously used in the literature. Below we will define terminology that was shown to

be used in the agent field by Arazy and Woo (2002) and show how they relate to the above model.

3.2.3.1 Perception
p(W,Sim)

A perception is a mutual property between the world and the agent's simulator, which means that when a perception changes in the world, it affects the state of the simulator. It is possible that something can occur in the world without the agent perceiving it, however, it will be impossible for the agent to know about that event. For example, if the farmer sees the wolf tracks, he perceives the wolf tracks, since it can change his model of the world.

3.2.3.2 Learning
e(W) such that $\exists A(W,Sim) \in C(e(W))$ and, the new state of Sim is unstable (leading to further changes of Sim).

Ontologically, learning is described as a transformation of the state of the world and a previous state of the simulator into a new state for the simulator. This means that learning is a change in perceptions. Since the change is due to a lawful transformation, an agent can only learn if the change adheres to the laws of the simulator, or if the end state of the transformation is lawful. For example, when the farmer sees the wolf tracks he learns that there is a wolf at the farm. Since that prior to seeing the tracks he did not think there was a wolf at the farm, we can say that once he saw the tracks his model of the world changed.

3.2.3.3 Belief
A specific p(Sim)

As discussed previously, a belief is a property of the agent's simulator. Since, the simulator is the agent's model of the world, the state of that simulator (or its collection of properties) is the agent's world view. Changes in beliefs though are governed by laws or assumptions. When we were speaking earlier of the change in the farmer's model of the world we were talking about changing beliefs. By perceiving the wolf tracks, the farmer gained the belief that a wolf was at the farm. Also, there is a delineation within beliefs between wants and expectations. Since both of these are types of beliefs they are both changed by perceptions, however, changes in wants must be consistent with the goal to lead to a "lawful" change.

Going back to our farmer example, if the farmer has the goal to kill the wolf, he will want to know where the wolf will be by the time he can pull the trigger. Based on the perceptions of the wolf's speed and pattern of movement, the farmer may guess that the wolf will be at the barn door by the time he can shoot at it, therefore the farmer will want to shoot at the barn door. If the farmer were to change his mind and only want to scare the wolf away, he will want shoot at the side of the barn instead. So his belief, "best location to shoot", changed from "barn door" to "side of barn" because his goal changed.

Now let us say that the wolf changed its speed. The farmer would then calculate that by the time he was ready to shoot the wolf, it would be at the side of the barn instead of the barn door. This change, in expectation of "future location of the wolf", was based entirely on the change in the world and did not incorporate the goal.

The main difference between the wants and expectations is that the transformation of the want property is linked to the goal. Therefore, the want will not be changed unless the agent is trying to achieve a goal, and when it is changed, the state that the simulator is in will be, or will be equivalent to, the goal. However its expectations can change even when it has satisfied its goal.

Using the ontologically refined representation for intelligent behaviour developed in the section 3.2.2, the difference between a want and an expectation is if e(Sim) such that $\exists$ A(Sim)$\in$ C(e(Sim)) and if e(Sim) leads to a goal state in the simulator then A(Sim) is a **want** otherwise its an **expectation**.

It is also important to note that since wants and expectations are beliefs they do not have to be correct. Expectations can be wrong and the desired value of wants may not lead to the completion of the goal.

3.2.3.4 Resources

p(E,W)

A resource is a mutual property of the effector and the world. Without resources the agent would not be able to affect the world. In the case of the farmer, his resource is the trigger of the gun which can change the state of the world. For the agent to interact with its environment, it must use (change) its resources.

3.2.3.5 Action

e(E) such that $\exists A(E,W) \in C(e(E))$

An action can be seen as a lawful event or lawful transformation of the effector. As stated before, an event is a change from one state to another. A state is a representation of a collection of properties and some of the properties/attributes of the effector can be classified as resources. So when the resource of farmer (the trigger) changes state, the farmer is performing the action of shooting. Because an action is a lawful event (due to the lawful transformation), certain property values need to be present before the event can actually take place.

3.2.3.6 Capability

p(E) such that $\exists s \in S\ L(s) \neq s,\ A_p \in C(s, L(s))$

Capabilities are intrinsic properties of the effector but are themselves tied to actions. If an agent has a resource but the value of an intrinsic property (capability) is not correct, the action can not take place. Going back to the gun example, to fire the gun the trigger has to be pressed down but suppose that the farmer does not have enough energy to pull the trigger. In this case, the "energy to pull the trigger" would be considered a capability since even though the agent has access to the resource it does not have the capability to perform the action.

3.2.3.7 Procedure

p(Sim,E)

A procedure is defined as a mutual property between the agent's simulator and the effector. This means that when a procedure is activated, or changed, it may lead to a lawful change in the effector's state (an action). When a procedure is activated the agent has the intention of performing a specific set of actions. When the farmer sees the wolf he may intend to shoot it, and he will want to perform the actions to carry out that procedure. It is possible that the agent will want to perform an action but may not be able to do so.

3.2.3.8 Reasoning

e(Sim) such that $\exists A(Sim,E) \in C(e(Sim))$

How do agents pick the procedures they wish to enact? According to the model, an agent will activate its procedures if a lawful transformation can occur between its simulator's previous state and its new state. Put another way, if the mutual property of the simulator and

effector changes, reasoning has occurred. This means that specific conditions need to hold before the agent will reason that a procedure needs to be activated. For example, suppose our farmer has two procedures, "scare the wolf" and "shoot the wolf", both of which will achieve his goal of keeping himself and the sheep safe. To choose between which procedure to undertake, the farmer needs to know which procedure will achieve his goal. If the farmer knows that the wolf will not come back if properly scared, he may simply want to scare the wolf. However, if the same wolf has been bothering him repeatedly no matter what he does, he may want to shoot the wolf. Reasoning is the mechanism which makes it possible for the farmer to choose between shooting and scaring the wolf.

3.2.3.9 Goal

$s \in S(Sim)$, $L_{Sim}(s)=s$

A goal is a set of the *stable* states of the agent's simulator. If the simulator is agitated in some way it will not stop thinking of new scenarios until it thinks it has reached its "goal" scenario. Once the goal scenario has been achieved, the agent will try to put the corresponding procedures into actions. Our farmer is trying to achieve his goal of making him and his sheep safe.

Below, in Table 3, there is a summary of the ontological construct to agent concept mapping.

Table 3: Agent Concepts and Definitions		
Intelligent Agent Concept	**Proposed Ontological Definition**	**Notes**
Perception	**p(W,Sim)** mutual property between the world and the simulator	If something occurs in the world without the agent perceiving it, it will be impossible for the agent to know about that event.
Learning	**e(W) such that $\exists A(W,Sim) \in C(e(W))$ and, the new state of Sim is unstable (leading to further changes of Sim).** a change in the mutual property of the world and the simulator and a following event (lawful transformation) in the simulator	The mechanism which enables the world to change the simulator. An agent can only learn if the change is aligned to the laws of the simulator.
Belief	**A specific p(Sim)** intrinsic property of the simulator	Changes in beliefs are governed by laws or assumptions L(Sim). But may not reflect the reality of the world
Resource	**p(E,W)** mutual property of the world and effector	For the agent to interact with its environment, it must alter (change, use) its resources.
Action	**L_E, e(E) such that $\exists A(E,W) \in C(e(E))$** lawful transformations of the effector and a change in the mutual property of the world and the effector	Certain properties of the effector (capabilities) need to be present before the event can take place.
Procedures	**p(Sim,E)** a mutual property between the simulator and effector	It is possible that the agent will want to perform an action but may not be able to do so.
Reasoning	**L_{sim}, e(Sim) such that $\exists A(Sim,E) \in C(e(Sim))$** a lawful transformation of the simulator and possibly a change in the mutual property of the simulator and effector	Specific conditions need to hold before the agent will reason that a procedure needs to be activated.
Goals	**$s \in S(Sim)$, $L_{Sim}(s)=s$** a stable state of the simulator	If the agent is agitated the simulator will try to find a procedure which will lead to the "goal" scenario.
Capability	**p(E), $\exists s \in S_W$ $L_W(s) \neq s$, A_p and $\exists A(E, W) \in C(e(E))$** an intrinsic property of the effector which changes as the result if certain transformations in the effector occur.	To perform certain actions (use resources) the agent must have some internal capabilities.

3.2.4 Implications of Proposed Conceptual Framework

In this Chapter we have defined the basic aspects of an intelligent agent, a system which takes in feedback from the world and produces reasoned actions to achieve its goals. By conceptualizing the intelligent agent as a system, we were able to identify its components and processes. By using the BWW ontology, we then were able to find the interrelationships between the concepts. Since the conceptual framework defines all the modes of interaction (mutual properties) between the agent components and the world as well as define those agent components, we now have a framework which incorporates the minimum concepts needed to talk about an agent which interacts with, models, and acts upon the world. We can say this because with only these concepts we can explain how a change in the world leads to an action by the agent. A perception in the world changes, which may lead to the agent learning a fact or opinion and incorporating that into its beliefs. If the new belief leads the agent to determine that its goal will not be achieved, the agent may reason that it needs to change its procedures. This may lead to actions being taken by the agent using/changing its resources if it has the required capabilities to do so.

When speaking of an intelligent agent that acts upon the world, we must identify the intrinsic and mutual properties of the simulator and effector, the changes (events) of the mutual properties of these things, as well as the simulator's stable states. The names given to these aspects of the simulator and effector (learning, goal, belief, etc.) are only useful for communication purposes and can be changed. However, using the BWW ontology, we have shown the concepts must be present for a modeller to say that they are using intelligent agents.

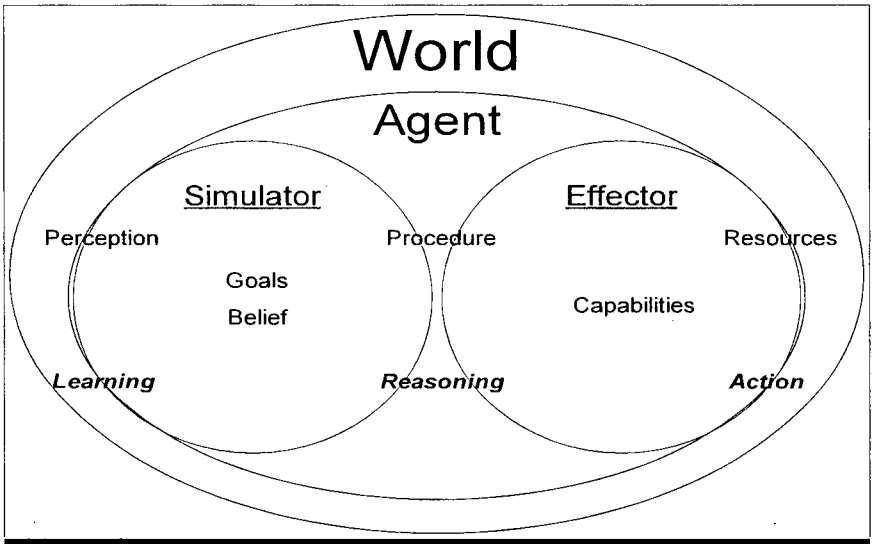

Figure 8: Representation of Intelligent Agent Concepts

Figure 8 shows graphically the different concepts discussed and how they relate to the world, simulator, and effector. The italicized concepts are those which are dynamic concepts and they are directly related to the concept above it; perception is tied to learning, reasoning is tied to procedures and actions are tied to capabilities. The framework denotes the simulator's dynamics (learning and reasoning), intrinsic attributes (beliefs and goals) and mutual properties or ways the simulator interacts with the other entities (perceptions and procedures). This is also done for the effector which, like the simulator, has dynamics (action and reasoning), intrinsic properties (capabilities), and mutual properties (resources and procedures).

Compared to Miller's (1978) and Swaminathan et. al's (1998) work, Figure 8 shows a more general and complete model of an intelligent agent. We can see that in Miller's model there is no goal. The agent can act in the environment and even associate actions with certain behaviours but we do not know why it does this. In Swaminathan et. al' s work we see that even though their model contains some of the same concepts as those in Figure 8, it focuses not on actions but on agent's receiving and sending messages. Since the agent is limited to these "actions" the concept of capabilities is not discussed.

Even though the models discussed are not the same as the one in Figure 8, the nine concepts used have already been discussed in Arazy and Woo (2002). It may be said that these concepts may have been arbitrarily picked. However, using systems theory to determine the components of the agent system, and investigating the processes of these components using the BWW ontology, we were able to pick only the concepts needed to describe a change in the world

leading to action being taken by an agent. Also the BWW ontology has shown precisely how these different concepts interrelate with one another. We have provided a set of concepts which, when "grouped" together, explain intelligent agent behaviour.

So we now have a conceptual framework which defines the intelligent agent terminology and also shows the interrelations between the different concepts.

Chapter 4: Visual Representation of the Agent Model

With the concepts defined in the previous chapter, we can describe intelligent agent action in the environment by saying that; perceptions affect the beliefs of the agent which can then lead to the agent trying to achieve a specific goal. The agent then goes through a set of procedures which will, according to reasoning, lead to achieving its goal. It will then try to turn those procedures into their respective actions if it has the capabilities to do so.

Writing down all of the procedures, beliefs, goals, etc. in a table, or in pseudo code form, would be cumbersome, especially when modelling the interaction between different types of agents. However, there is a way to show these interactions in graphical form. We propose that a graphical representation of the conceptual framework will aid modellers in designing their agent systems.

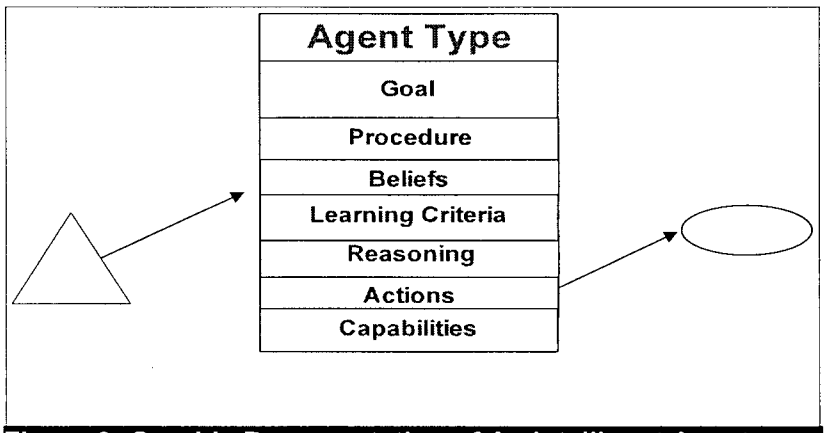

Figure 9: Graphic Representation of An Intelligent Agent

All of the concepts in Figure 9 are familiar from the previous Chapter except Type which can be described as the agent's "name". A type has its own unique goals, procedures, beliefs, reasoning, actions and/or capabilities. An agent has the ability to autonomously change into other types of agents as specified by the modeller.

It should also be noticed that the concepts, resource and perception, are missing from Figure 9. They still can be depicted using the graphical representation but will be discussed later.

Figure 9 also introduces the idea of a learning criteria. When applying an earlier version of the framework, we had decided that even though learning was an aspect of an intelligent agent, its workings would be too complex to show in a conceptual model. As we progressed, however, we found that it was vital to show some form of the learning behaviour in the model. The learning criteria is a way to show how the agent "labels" stimulus from the environment for the purpose of learning. Using the ontological model created in section 3.2, we can say that the learning criteria are state transition laws of the simulator.

According to a survey of computational learning (Angulin, 1992), a learning system is comprised of a target concept, examples, and labelled examples. Target concepts are what the agent is trying to learn, in this case a belief. Our farmer has the belief that "all wolves are dangerous", this would be a target concept. Direct (or indirect) observations of wolves acting dangerously, or not, would be the examples. According to Angulin, the learning algorithm needs labelled examples that are given as "good" or "bad" to come to an accurate conclusion.

Ontologically, "good" means that according to the simulator's transition law the change in the mutual property between the world and the simulator leads to the value of the property of the simulator corresponding to the target concept to be set to "true" (the target concept is thought to be true). A bad example would be when, according to the simulator's transition law, the change in the mutual property between the world and the simulator leads to the value of the simulator's property to be set to "false".

If the farmer sees wolves chasing down people and farm animals, this would constitute a positive acknowledgement of the target concept ("good"). If the farmer saw a pack of wolves leave his farm animals alone without any discouragement from him, then this would be negative to the target concept ("bad"). By dealing with these examples, the learning algorithm can create an approximate hypothesis about the target concept. Depending on his experience and information, the farmer may come to believe that all wolves are dangerous, or not. This framework is able to incorporate different computational learning techniques such as classifier learning systems[1] and q-learning[2] since these techniques use the same type of learning

1 "Classifier systems are rule based systems that learn by adjusting rule strengths from environmental feedback and discovering better rules" (Sen & Weiss, 1999)

2 Q-learning is a learning algorithm that tries to find a set of actions which maximizes an agents future payoffs in all states of the environment (Sen & Weiss, 1999)

methodology (Sen & Weiss, 1999). The learning criteria gives the user the ability to set when an example can be labelled as positive or negative for the agent. Anything more detailed would limit the ability of the modeller to specify the learning algorithm.

Perceptions and resources in the model are displayed as either agent or system variables. System variables are properties that make up the state of the world and are shown in the proposed framework as ovals, while agent variables refer to agent properties that are not part of its effector or simulator and are shown in the framework as triangles. For example, an agent's height is definitely an attribute of the agent but not necessarily of the simulator nor the effector. In the representation, the difference between the resources and perceptions is shown by the agent's relation to the variables. If an interaction arrow goes from the variable to the agent then it is a perception. However, if the interaction arrow goes from an agent to a variable then the variable is a resource.

If a researcher wanted to see how a customer responded to various sales and price information and wanted to build an agent-based simulation, Figure 10 is an example of the how the researcher would model her customer agent.

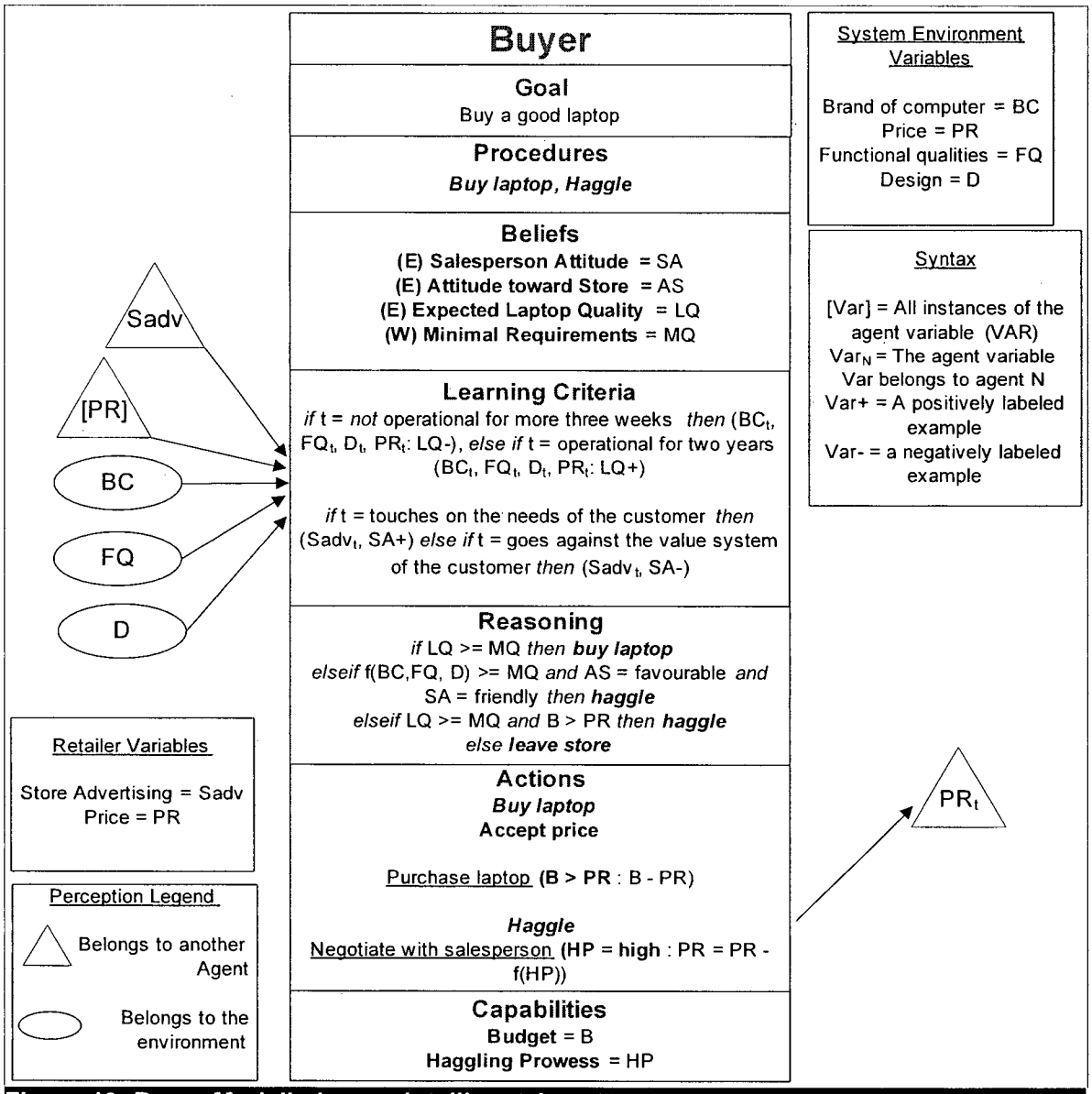

Figure 10: Buyer Modelled as an Intelligent Agent

In Figure 10 the agent learns which prices, designs, brand name and functional features can lead to a high quality computer while which ones will not. It then looks at those variables in the current laptop to decide on an expected quality of the laptop.

As mentioned previously, the triangles are agent variables while the circles denote system or "world" variables. Specifically, Sadv is the store advertisements which have an effect on the consumer's attitude toward the store. PR refers to the price of the laptop, BC represents the computer brand, FQ is the computer's functional quality, or specifications, and D is the

physical design of the computer. Each of these variables affect the belief of the expected laptop quality.

Along with the distinction between agent and system variables there are also distinctions between specific and aggregate variables. Aggregate variables are shown as [variable] while specific variables do not have brackets. For example, the customer takes into account store prices of **all** retailers ([PR]) to make a decision about the expected laptop quality.

In the beliefs section of the model above, each belief is also denoted as a want or an expectation. As stated before, the main difference between wants and expectations is that one is tied to the goal while the other is not. For example, the attitude of the salesperson (AS) is really an expectation of what the salesperson's real attitude is and has nothing to do with the goal of getting a good laptop. However the minimum requirements (MQ) needed for the laptop is a want, since it is the least that the buyer wants to have for the laptop to be "good". Ownership of a variable is denoted by a subscript therefore BC_t is the brand name of computer "t".

The last bit of syntax to explore is in the learning criteria. As mentioned previously, the learning criteria makes it possible for the agent to label "good" and "bad" examples of a concept. In Figure 10, the agent learns to pair a specific brand name, a set of functional requirements, design, and price (BC_t, FQ_t, D_t, PR_t) with a good laptop quality (LQ+) if the laptop is functional for more than two years or bad laptop quality (LQ-) if the laptop does not work after three weeks of purchase. It is important to note that the + and – symbols do not necessarily mean good or bad in a subjective sense but are linked to the belief that they are learning. For example, if our farmer were learning which animals were dangerous to his sheep, the learning criteria may look like this:

$$\textit{if}\ \text{prey}_{\text{animal}} = \text{sheep}\ \textit{then}\ (\text{type}_{\text{animal}}, \text{size}_{\text{animal}}\text{: dangerous+})$$

This says that if the farmer sees an animal hunting sheep then he will take note of the type and size of the animal. So dangerous+ does not mean "less dangerous" or that the agent likes danger, but that it is learning to positively associate the variables with the belief that there is danger.

Moving from the design of a specific agent, we will now show how agent interaction can be shown in the representation. Below is an example of a generic interaction between two agent types.

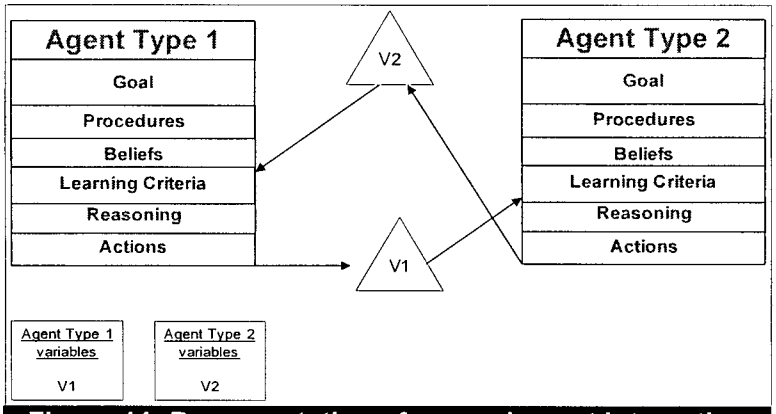

Figure 11: Representation of general agent interaction

Interaction in Figure 11 is shown by changing agent variables. The diagram shows that agent type 1 is "interacting" with agent type 2 and vice versa because they both can perform actions that affect their counterpart's perceptions.

Another important aspect to the modelling of intelligent agents is the use of sub-types and transitions. According to Wooldridge (2002), intelligent agents are adaptable computer systems, therefore it should be possible for them to change type. Subtypes are also useful for modelling, because information is not repeated needlessly in the diagram.

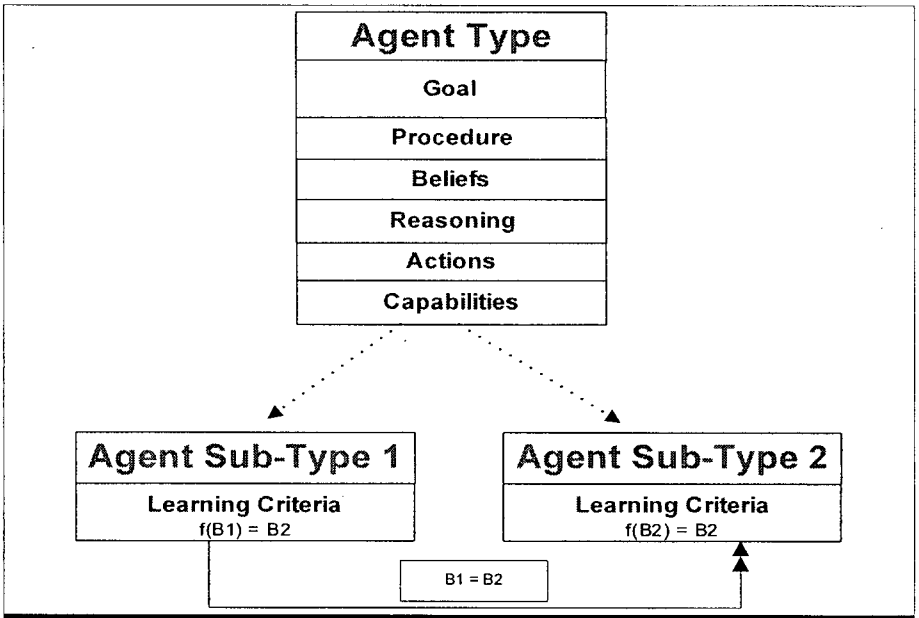

Figure 12: Representation of Agent Sub-Types and Type Transitions

The double-headed arrow shows the transition from one type to another. There is also a trigger which will state the conditions for when the agent will change type. The dotted arrows show that the top agent type (Agent Type) has two subtypes, Agent Sub-Type 1 and Agent Sub-Type 2. These subtypes have everything that their parent types have but differ specifically with what is outlined in the model, in this case the learning criteria for each subtype is different.

4.1 Generic Implementation of Conceptual Framework

The graphic representation is useful since it gives researchers the ability to visualize the model. However, the representation discussed in the previous section is the design phase of an agent-based system and according to Shehory and Strum (2001), a design model should be implementable. Since the representation is based on the concept of an agent, it should be implementable in an agent programming environment. The platform we first chose was Netlogo which was developed as a teaching and research tool, and has been used to build physical and social science simulations. The aim of the platform's authors was to make an easy to understand environment for the construction of complex models (Tisue & Wilensky, 2004).

> [Netlogo is a] programmable modelling environment for simulating natural and social phenomena. Modellers can give instructions to hundreds or thousands of independent "agents" all operating concurrently (Wilensky, 1999).

The environment merges object-oriented programming languages (Java) with agent specific commands and operations (Tisue & Wilensky, 2004). Just like in object-oriented languages, each "object" in Netlogo can have a unique set of variables.

Since the language was made to easily represent agent-based models, the graphic representation should be easily converted into a Netlogo program. Below, we will show how each part of the graphic representation agent can be operationalized using the Netlogo code. We will first start with the static elements of the agent system and then go on to the dynamic processes of intelligent agents. In Arazy and Woo (2002), the researchers split the various agent concepts in the literature into static and dynamic components. Using their taxonomy it can be shown that in the conceptual framework; type, perceptions, beliefs, procedures, and capabilities are static elements, while learning, reasoning, and acting are dynamic elements. We will start with the static elements and then move on to the dynamic elements.

4.1.1 Type

The type of an agent in the graphic representation is the name of an agent with a specific set of goals, procedures, beliefs, etc. In Netlogo this is called a "breed". Breeds are a classification of agents which can be called on later. An agent has a breed and can change breeds later on if so programmed (Wilensky, 1999). Breeds are defined in Netlogo as `breeds[x y]`. In the above code there are two types of agents; x's and y's.

4.1.2 Perceptions

Just like in the representation, Netlogo differentiates between system and agent variables. In Netlogo there are three kinds of variables; global, patch, and agent. The first two, global and patch, deal with the agents' environment while an agent variable belongs to an agent type. A global variable is a "system" variable, it is "one value and every agent can access it" (Wilensky, 1999). A patch variable is a variable that belongs to a spot on the "physical space" of the Netlogo environment in the display area which acts as a visual representation of the world. For example, if we were simulating our farm example, a patch variable could be "amount_of_grass". The proposed representation does not differentiate between location (patch) and system variables so we will look at how to work with and declare only system and agent variables. If we continue simulating the farm, a system variable could be sunrise_time or the time at which the sun will rise. It would be declared as `globals[sunrise_time]`. Agent variables are declared separately. For example if animals were to be simulated as agents on the farm, each one could have an age. Therefore, there would be an agent variable called age, declared as `turtles-own[age]` (turtle is Netlogo syntax for a generic agent).

If we only wanted to simulate the sheep and the sheepdogs on the farm, according to section 4.1.1 we would code `breeds[sheep dog]`. However sheep and sheepdogs may have very different variables, for instance, the dogs may have a variable called "house_time" which is a variable for how many times the animal will sleep in the house. It is unlikely that the sheep would ever have anything above zero. Rather than give the agent a variable it will never use, Netlogo can give different breeds, different variables. So our house_time variable can be declared as `dog-own[house_time]`. In Netlogo, breeds get their own variables and any variables that were declared for generic agents. The only way to modify and identify agents in Netlogo is by their breed name and their variables.

4.1.3 Beliefs, Procedures, and Capabilities

According to our conceptualisation of an intelligent agent, there are three things that are modified as the agent interacts with its environment, its beliefs, procedures, and capabilities. What this means is that along with the agent variables discussed previously, beliefs, procedures and capabilities also have to be defined as "agent variables" in the Netlogo sense. Going back to our farm example, sheepdogs are used to herd sheep into pens. If we simulated the behaviour of sheepdogs on the farm, each dog would have a "belief" about how many sheep it is supposed to herd into a pen, and two procedures it could follow, ***herd sheep*** (bring more sheep into the pen), or ***close the gate***. Its capabilities would be "authority to herd the sheep" and the "energy to move around". The Netlogo code for this would be `dog-own[age #_sheep herd close_gate authority energy]`. So beliefs, procedures, capabilities and agent variables are declared all at once.

Netlogo does not keep track of what is a belief, procedure, or capability, they are shown in Netlogo code only as generic variables of the agent. This means that the reader of the code has to interpret which variables are procedures, beliefs, or capabilities. However these variables can be identified since beliefs, procedures, and capabilities do not belong in certain places. For example, beliefs do not belong in any "action" code. So beliefs and capabilities can be identified based on where they show up in the code. The same can be said for procedures as well, but they also have another distinguishing feature, all procedures are boolean. Since Netlogo does not require users to type variables before use, the only way to know if a variable is boolean is to track its value throughout the program.

As stated, the previous concepts were static elements of the agent. Now we will deal specifically with the three dynamic sections of the proposed agent system; learning, reasoning and acting, characterized by the code below with the reserved words italicized:

```
to run_simulation
   learn
   reason
   act
end
```

The above Netlogo code block is called a procedure. Any dynamic action that needs

to be called in Netlogo has to be written as a "procedure" and is prefaced by "to". We will see more procedures in the next sections.

4.1.4 Learning

The procedure called "learn" contains all of the learning functions of the agents in the simulation. The Netlogo code would be:

```
to learn
   ask agent_type1 [1learning]
   ask agent_type2 [2learning]
   ...
end
```

In Netlogo the ask command is used to tell an agent, or groups of agents, what to do. For example, the code above asks all agent_type1's to execute the "1learning" procedure. The modeller then must specify the procedure, 1learning, or how agent_type1 learns.

The learning criteria used in the representation does not deal directly with a specific kind of learning but gives the agent the criteria for labeling examples that can guide the learning processes. How the agent goes about creating these labels is left to the modeller, however the representation can still be used as short hand to limit the amount of coding needed. To help show this we will give an example of one type of learning technique, q-learning. In section 5.4.1 we will give a more detailed view of what q-learning is, but for now, we will only use it to demonstrate the versatility of the learning criteria.

Suppose we created a farm simulation where the farmer wanted to find a spot of land (locationX) where the proportion of well fed flock (WFF) is 80% or higher. The learning criteria for the agent would be if WFF > 0.80 then locationX+ else locationX- or "if the proportion of well fed flock is above 80% then the grazing location is considered good". The Netlogo code for learning criteria is below. This is the actual mathematical implementation of the q-learning algorithm, however the parts that concern us for now are in bold.

```
to 1learning
      ifelse WFF > 0.8 [set reward 5 + 2 * WFF] [set reward 0]
      set step 0

      while [step < (range_of_strat)][
           ifelse step = locationX[
                set accum (1 - ex) * reward
```

```
            ]
            [set accum ex * (item step strat_score / range - 1)]

            set q_table replace-item step strat_score ((1 - r) * item step
            strat_score + r * accum)
            set step (step + 1)
      ]
end
```

Now if another simulation was created where the farmer needed to determine the best time of the week to sell his eggs, the learning criteria would be "if MAX(egg_profit) then tsell+ else tsell-". Step by step we will go through how to transform this statement into Netlogo code. First we will show the q-learning code, abstracting the parts that are relevant to the learning criteria.

```
to 1learning
      set reward objective function
      set step 0

      while [step < (range_of_strat)][
            ifelse step = belief_to_be_learnt[
                set accum (1 - ex) * reward
            ]
            [set accum ex * (item step strat_score / range - 1)]

            set q_table replace-item step strat_score ((1 - r) * item step
            strat_score + r * accum)
            set step (step + 1)
      ]
end
```

Since in q-learning the reward is trying to be maximized in the Netlogo code, we would only need to make sure that the reward was equal to the variable egg_profit. And the belief to be learnt would be what time to sell, or tsell.

```
to 2learning
      set reward egg_profit
      set step 0

      while [step < (range_of_strat)][
            ifelse step = tsell[
                set accum (1 - ex) * reward
            ]
            [set accum ex * (item step strat_score / range - 1)]

            set q_table replace-item step strat_score ((1 - r) * item step
            strat_score + r * accum)
            set step (step + 1)
      ]
```

```
end
```

There is little difference between the two code blocks other than the bold parts, even though the things being learnt are very different.

4.1.5 Reasoning

Just like "learn", "reason" is a procedure that specifies how different agent types choose the actions they want to initiate. It also follows the same basic structure as the learn procedure. However, since reasoning is generally less complicated than learning, extra procedures usually are not needed.

```
to reason
   ask agent_type_1 [
       ifelse belief1 != x [
           set procedure1 true
       ]
       [set procedure1 false]
   ]
   ask agent_type_2 [
       ifelse belief2 != y [
           set procedure2 true
           set procedure3 false
       ]
       [set procedure2 false
       set procedure3 true]
   ]

end
```

Notice that the procedure variables are given true or false as a value. This is done so that the program can track the instance where a procedure is enacted, but its actions cannot be performed. If a procedure is "true" but its corresponding actions have not occurred, the modeller can then say that the agent has the intention to perform the procedure but does not have the capabilities to do so. If procedures were implemented as Netlogo procedures, it would be harder to track unfulfilled intentions.

So just as in the graphic representation, reasoning uses the beliefs of the agent to decide which procedure needs to be enacted. For instance if we were modelling the sheepdogs we discussed previously, we would create a reasoning function saying "if #_sheep >= #_sheep_in_pen then ***close gate*** else ***herd***". This means that if the number of sheep that are supposed to be in the pen is greater than or equal to those in the pen, the dog will choose to close

the gate, otherwise it will herd more sheep. The Netlogo code would be

```
to reason
  ask dog [
      ifelse #_sheep >= #_sheep_in_pen [
          set close_gate true
          set herd false
      ]
      [set close_gate false
       set herd true]
  ]
end
```

4.1.6 Actions

Acting is the last dynamic process in intelligent behaviour and is also implemented as a Netlogo procedure:

```
to act
    ask agent_type1 [
        if procedure1 = true and capability1 = some_value[
            set agent_variable x
        ]
        if procedure2 = true and capability1 != some_value[
            set agent_variable2 y
        ]
    ]
end
```

It can be that an action can occur only if the agent has the requisite capabilities. If not, then the agent has the intention of performing the action (its corresponding procedure is true) but can not do so. To make the code more concrete, let us go to the sheepdog example again. Figure 13 is the graphic representation of the sheepdog's actions.

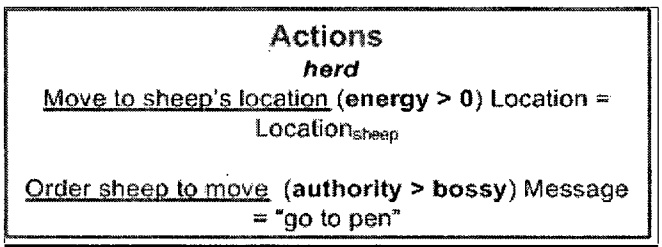

Figure 13: Representation of sheepdog's actions

Figure 13 is saying that to engage in the herd procedure, the dog must first move to the sheep's location if it has the energy to do so and then must give the sheep the "message" to go to the pen, if it has the authority to do so. The Netlogo code translation of this is:

```
to act
      ask dog [
           if herd = true [
               if energy > 0 [
                   set location location-of sheep
                   if authority > "bossy" [
                       set message "go to pen"
                   ]
               ]
           ]
      ]
end
```

4.1.7 From Representation to Implementation

Now that the basics of the translation have been shown, we will go through a complete translation by outlining a problem, developing an intelligent agent representation for it and then creating the code for implementation in Netlogo. Moving away from the farm, suppose we have a race with two kinds of runners, amateur runners who exercise for fun and health reasons, and professional runners who compete in events all around the world. Now suppose there was a city marathon where anyone could enter. We want to model a simulation where agents would race against each other so as to understand how amateur runners in a race would act if there were no, some, or very many professional runners. We first learn from the literature that professional runners will gauge their own strength as well as try to outpace their opponent. However amateur runners, caught up in the competition, will run fast, even at the beginning of the race to try and keep up with their opponent. This means the runner would be wasting energy that could be used at the end of the race.

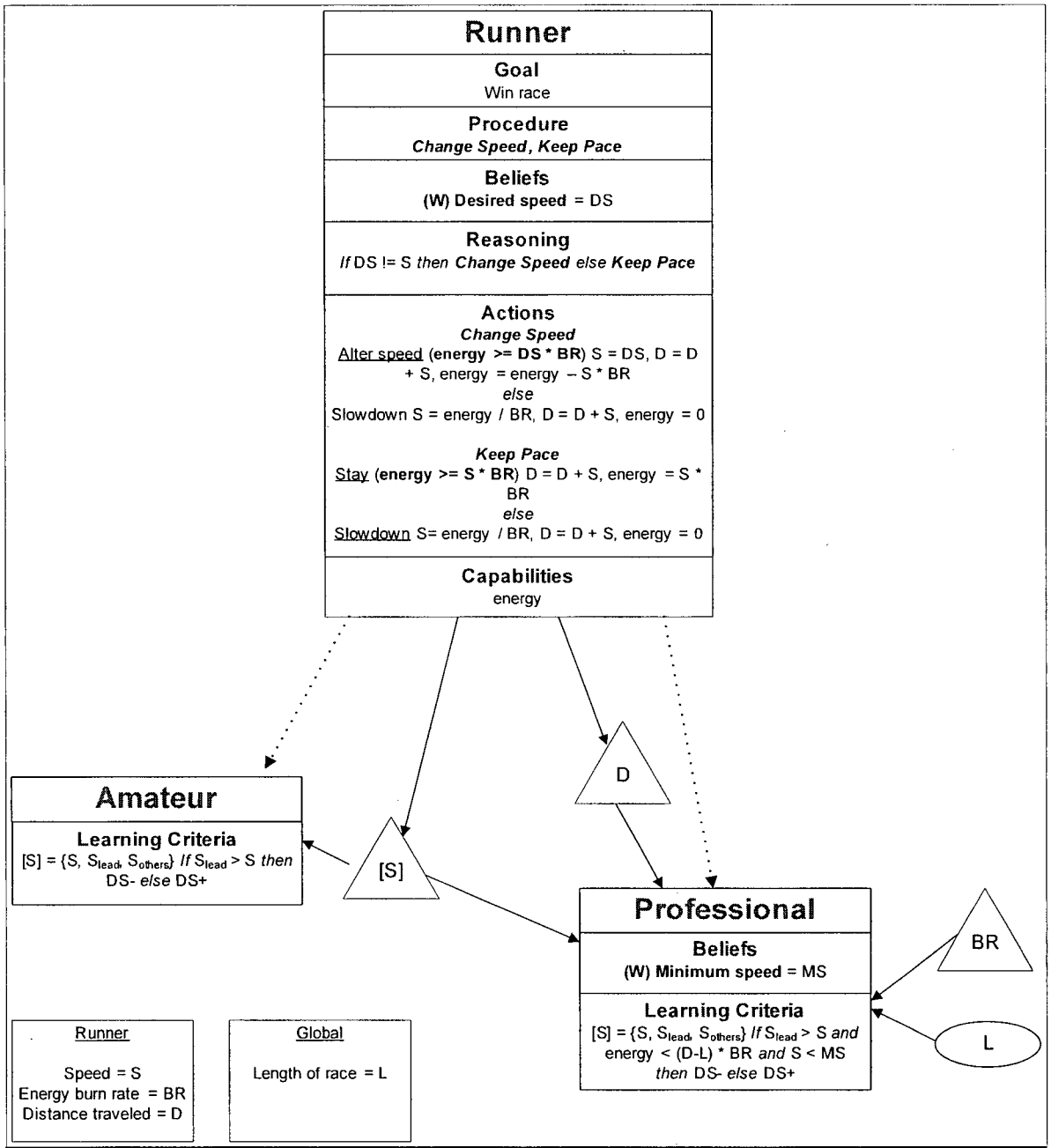

Figure 14: Representation of Runners' Marathon

Figure 14 shows that there is one agent type, the runner, with two subtypes, the amateur and the professional. There are also some global and agent variables. We will begin the translation of the representation by outlining the global variables in Netlogo.

```
globals [L lead ex r C]
```

Along with the global variable L, shown in the representation, we also included another variable called "lead". This will keep track of which agent leads the race. The ex, r, and C variables are used in the learning algorithm. Now we turn to the agent variables. Since there is only one type of agent we will use the generic, "turtles" identification.

```
turtles-own [ D S DS MS BR energy change_speed keep_pace type DSscore DSprob
reward step probtotal accum]
```

We include the professional agent's beliefs, because in Netlogo there is no way to separate variables for different subtypes, only types. We have now declared all procedures, beliefs, variables and capabilities for the runner agents. We have included the "type" variable which will make it possible for us to distinguish between amateur and professional agents. We have included the DSscore, DSprob, reward, step, probtotal, and accum variables which are used in the learning algorithm. All beliefs which are to be learned by the agent must be accompanied by a score variable if q-learning is being used, in this case DSScore. Also each "score" variable must have a corresponding probability distribution and the reward variable is used to decide which value of the belief the agent learns.

Now that the static elements are in place, we can develop the dynamic part of the program. First we use the basic "go" procedure outlined previously but with a twist.

```
to run_simulation
   lead = turtle with [max D]
   learn
   reason
   act
end
```

We have added a function which makes sure that the runner in the lead (the runner with the most distance) is identified. We now must outline the learn procedure in the simulation. The code for the learning function looks like this:

```
to learn
     ask turtles with [type = "amateur"] [
          update-DSScoreA
          update-prob
          bestspeed
     ]

     ask turtles with [type = "professional"] [
          update-DSScoreP
          update-prob
          bestspeed
```

```
    ]
end
```

This procedure can not be fully derived from the representation for several reasons. First these procedures are specific to q-learning and may not be used in other learning algorithms. Also, there are many ways to conceptualize the q-learning functions and they do not have to necessarily be split into these different functions.

The update-DSScore function is the heart of the q-learning algorithm. This procedure takes in information from the environment and learns which action is the best to take. The syntax in the learning criteria box can be used to derive part of this code.

```
to update-DSScoreA
     set step 0
     set reward S - S-of lead

     while [step < (L / 4)][
          ifelse step = DS[
              set accum (1 - ex) * reward
          ]
          [set accum ex * (item step DSScore / (L /4) - 1)]

          set DSScore replace-item step DSScore ((1 - r) * item step DSScore
          + r * accum)
          set step (step + 1)
     ]
end
```

The term L / 4 is used in the procedure to give the range of options that the agent goes through. Therefore, the agent will experiment between moving 0 units per turn up to a quarter of the length of the race per turn. This is specific to the implementation and does not appear in the representation. Also the r and ex variables are parameters of the learning algorithm and are too specific to be specified in the representation. The update-DSScoreP procedure is exactly the same as the block of code above except the reward function would be

```
set reward S - S-of lead + energy - ((D - L) * BR) + S - MS
```

The update-prob and bestspeed procedures can not be directly derived from the representation but they do follow a pattern which makes it possible to automatically write their code when the update-DSScoreA and update-DSScoreP procedures are defined.

Now moving on to the reasoning procedure, since all agents have the same reasoning in the representation then it is possible to write one function.

```
to reason
      ask turtles [
         ifelse DS != S [
             set change_speed true
             set keep_pace false
          ]
          [set change_speed false
           set keep_pace true
          ]
       ]
end
```

The syntax found in the reasoning box is very straightforward and is easily translated into the Netlogo code. The syntax gives a condition where one procedure is activated and when it is not. However when implementing the code, it is not only important that a procedure is activated (set to "true") when its condition holds, but that it is also deactivated (set to "false" when its condition is not true. Lastly, the act procedure needs to be defined. Again this procedure is the same for each subtype therefore there needs to be only one procedure.

```
to act
      ask turtles [
           if change_speed = true[
               ifelse energy >= DS * BR [
                  set S DS
                  set D D+ S
                  set energy energy - S * BR
               ]
               [set S energy / BR
                set D D + S
                set energy 0
               ]
           ]
           if keep_pace = true [
               ifelse energy >= S * BR [
                  set S DS
                  set D D+ S
                  set energy energy - S * BR
                ]
               [set S energy / BR
                set D D + S
                set energy 0
               ]
           ]
       ]
end
```

The syntax of the representation makes it possible to create this code. The representation tells the programmer what actions are attached to which procedures. So a coder can say what should happen if a specific procedure is true (if the agent reasons it wants to perform a procedure).

4.2 Visual Programming Environment

Section 4.1 has shown that the representation is easily translated into a Netlogo program. Therefore we propose that developing a modelling/programming environment for intelligent agents would be feasible. Non-technical users could automatically create multi-agent based simulations using the graphic representation. It has been shown by the business press' response to automatic database generation in Visio (Plotkin, 1999), that this functionality in a modelling program would be greatly appreciated. Below is a mock up of the environment using Visio.

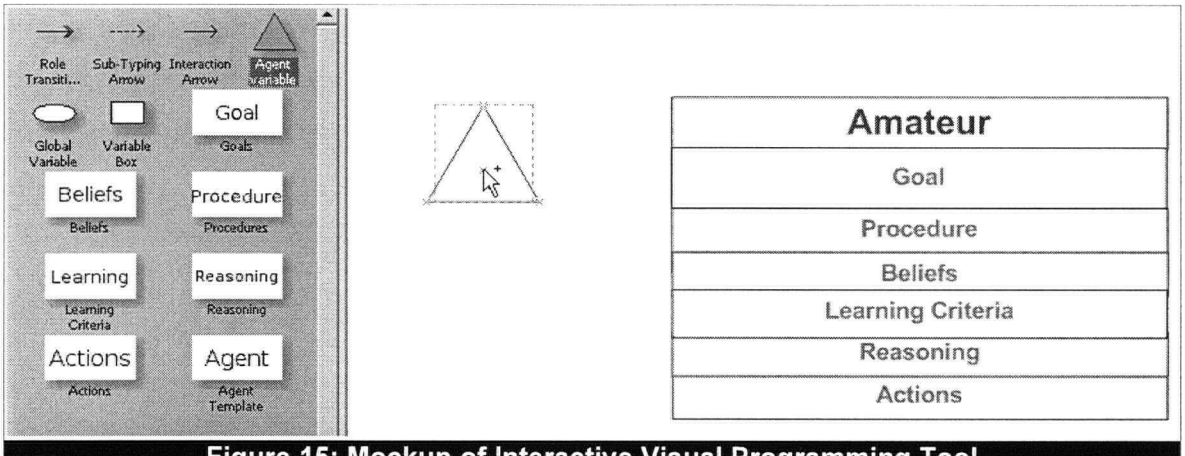

Figure 15: Mockup of Interactive Visual Programming Tool

Figure 15 shows the proposed modelling environment which would involve the constructs in the previous sections. As users drag and drop constructs onto the environment and fill in the missing information under the specific agent concept headings, they can provide the names of the beliefs, procedures and capabilities of the agent which would be converted into programming code. Modellers could also generate the code for agent reasoning using the environment. Finally, the environment could automatically generate code to declare both global and agent variables by having the user fill in the "variable box" construct.

However there are some aspects of the simulation that would not be generated by the environment. The specifics of the learning algorithm would not be fully generated using the representation and would need to be specified by the user. The user would also have to specify any monitoring (e.g. number of agent types) or operation (e.g. number of steps in a "while loop") variables. Also, since the representation does not show the order of operation for types, this

would need to be added to the program by the user. If all of one agent type needs to perform a task before the rest of the other agent types can properly perform their duties, then this code would have to be entered by the modeller. Another potential problem could be the issue of sub-typing. If using a language that does not allow sub-typing of agent types, the user would have to decide how different subtypes would be identified. A sub-type variable could be created for the agent type, but it is also possible that the user would want to identify sub-types based on the value range of a belief or capability. Therefore the method of sub-typing would have to be specified by the modeller.

Like the work done by van der Zee and van der Vorst (2005), we hope that this framework will make it possible to make coherent workable simulations without needing the expertise to code complicated algorithms.

Chapter 5: Application of the Model

As stated before, the design of a system must be implementable. To test the usefulness of the model we will apply the representation to a practical problem using intelligent agents. The research area we chose was ACE simulation, discussed in Chapter 2. Many of the computational agents used in the area of economics are very simple (Tesfatsion, 2002). One area where more intelligent agents may be needed is competitive pricing strategy research. In this area, firms do not simply react but must also engage in strategic thinking.

5.1 Competitive Pricing and Game Theory Overview

Narasimhan (1988) proposed a scenario on competitive pricing strategies between two firms. In this scenario two or more firms set a promotional price for a product. Each brand has a loyal segment which always buys a specific brand, and there is also a "switcher" segment that will buy whichever brand is cheapest. Narasimhan developed a game theory model where players had to lower their price to attract a large market share but also needed to make a profit. The game theory model of the scenario came to the conclusion that a Nash equilibrium would have to involve mixed strategies. In game theory, a mixed strategy is when the actions of the players are random but follow a main distribution. For example, a retailer could decide to randomly price its goods with a mean of $6.00 and a standard deviation of $0.30. This would make it highly unlikely that it would charge $2.00 or $15.00 for its product but the exact price

would not be known to its opponents.

There are two main theories about the rationale behind mixed strategies. The first, proposed by Von Neumann and Mogenstern, states that a player engages in mixed strategies to conceal its strategy from the opponent. However, one criticism is that Von Neumann and Mogenstern only looked at mixed strategies in a zero-sum game (Reny & Robson, 2004). Therefore, if a player can adjust its behaviour to take advantage of its opponent's known strategy it will gain a significant advantage in the game. This explanation makes no sense in a non-zero sum game, where some cooperation may be advantageous to the players (Schelling, 1960). In this scenario it would be counter-productive to wilfully deceive the other player in the game since the knowledge of their strategy may lead to the optimal outcome for both players.

However, Harsanyi (1973) states that mixed strategies can still occur in this situation. The modified rationale for mixed strategies is that, there is incomplete information between the players in the game. Therefore, if two players are trying to gain the most they can in a given situation, but do not have the ability to fully coordinate, each must guess the other player's action.

This "uncertainty principle" is shown in the paper by Aumann and Branderburger (1995) where they investigate the conditions where a Nash equilibrium can occur in a mixed strategy situation. In their discussion they introduce the idea of the players having an interactive belief system. Players have actions (strategies), which are rationally chosen and based on conjectures (beliefs) of what the other players in the game will do. Since there is uncertainty as to what their opponents will do, the players' conjectures have the ability to shape how the game is played out and if it reaches a Nash equilibrium or not.

5.2 An Empirical Study of Competitive Pricing Strategy

Researchers Choi and Messinger (2005) were unsure whether individuals would truly pick the mixed strategy behaviour under time and cognitive constraints. A competing hypothetical outcome for the Narasimhan game was that retailers would perform Edgeworth cycle pricing. An Edgeworth pricing strategy is where players undercut each other until they reach a point where it is no longer profitable for them to price lower, at which point they both raise their prices.

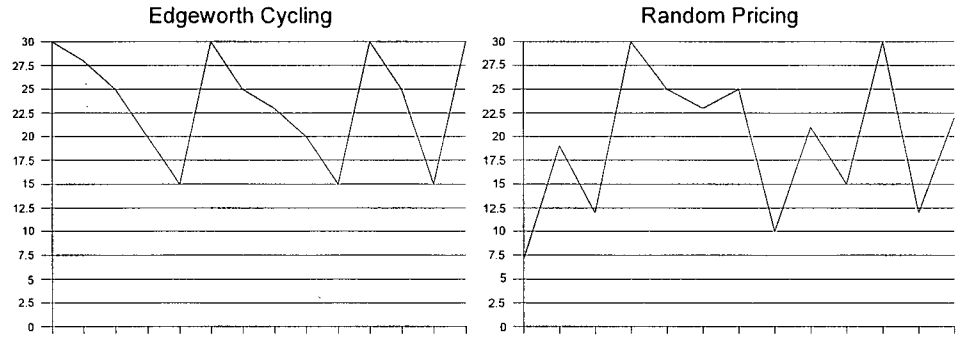

Figure 16: Random vs. Edgeworth Cycle Pricing

To test their hypothesis, the researchers designed an empirical study with human participants. Each of the subjects were paired with each other and played thirty "rounds" as retailers competing against one another in the Narasimhan scenario. The subjects were paid based on how much profit they accumulated over time and not on how many times they "won" the switcher market. The cost of each good sold was $15.00 while the subjects could only price up to $30.00.

The study had three experimental conditions; the first group had symmetrical loyal groups and no preference for either brand in the switcher market. This means that each subject had 30% of the market which he/she could rely on to always buy from them. Also the switcher market would simply buy from the subject who priced the lowest. The second experimental group also had an unbiased switcher market but the loyal segments of the market were uneven. Retailers with Brand 1 had a 40% loyal market while retailers with Brand 2 only had a loyal segment of 20%. The third experimental group had a symmetrical loyal group but the switcher market had a preference for Brand 1 with $d = 3$. This meant that the subject assigned to Brand 2 had to price $3.00 lower than Brand 1 to gain the switcher market.

The researchers found that the Narasimhan mixed strategy could accurately predict most of the prices in the games, however they did find that there was serial correlation in the data, meaning that price at time T was somewhat dependent on price at $T - 1$. The researchers found that 78 out of 128 trials had some serial correlation in prices, however the assumption of independence between the rounds is central to the idea of the Nash equilibrium found in the Narasimhan game.

To see what strategy could be contributing to this, the researchers tested to see if the Edgeworth cycle pricing pattern could predict the changes in prices. They first tested the null

hypothesis that Edgeworth cycling had no effect on price adjustments, which was rejected. They then tested the null hypothesis that Edgeworth cycle pricing could explain all of the price adjustments, which was also rejected. The researchers then concluded that Edgeworth cycling has a partial effect on price adjustments. The adjusted R^2 of the tests shows that out of all the treatments, Edgeworth cycle pricing explains Treatment 1 the most.

However we do not know why people engaged in partial Edgeworth cycling behaviour. To gain a better understanding of the reasons for these results we proposed creating a simulator to study the interactions between the two retailers and to find the conditions under which the behaviour in the study occurs.

5.3 Conceptual Framework of the Competitive Pricing Simulator

The fact that there was a random strategy and a discernible pattern in the pricing game scenarios conducted by Choi and Messinger is very similar to what Aumann and Branderburger (1995) noticed in their rational and "irrational" games. However other things also seemed to be involved in the experiment, such as learning and competitive aggressiveness that will be hard to implement in a game theoretic model, but were also hard to control for in the experimental setting. Another method needs to be used to gain a better understanding about how mixed strategies come about and if they really are conscious randomizations or clashes between differing conjectures in the competitive pricing strategies.

An intelligent agent based simulation would aid researchers to understand this problem. We can compare the simulation to the game theoretic results since the concept of intelligent agents fit very nicely with Aumann and Branderburger's interactive belief system model. We can say this, since intelligent agents must reason about strategies (actions) based on their beliefs of what will happen in the future (conjectures), just like in Aumann and Branderburger's model. We also can compare the results from the simulation to the empirical data by making the agents mimic behaviour. To do this though we must understand how agents develop their strategies.

According to Vidal and Durfee (1996), agents can have three broad levels of strategy when interacting in a market place; strategy 0, 1, and 2. Strategy level 0 players only use their "personal" past experience to learn how to win the game, while Strategy level 1 players incorporate opponents' actions into their thinking. Strategy level 2 players, like level 1 players

think about their opponents, however they model their opponents as level 1 players. What this means is that while a level 1 player will only predict its opponent's behaviour by assuming that its opponent only takes into account its own actions when making decisions, a level 2 player will assume that its opponent is at least partially basing its behaviour on the agent's own thinking.

Figure 17 shows the graphic representation of the system developed to simulate the Narasimhan game.

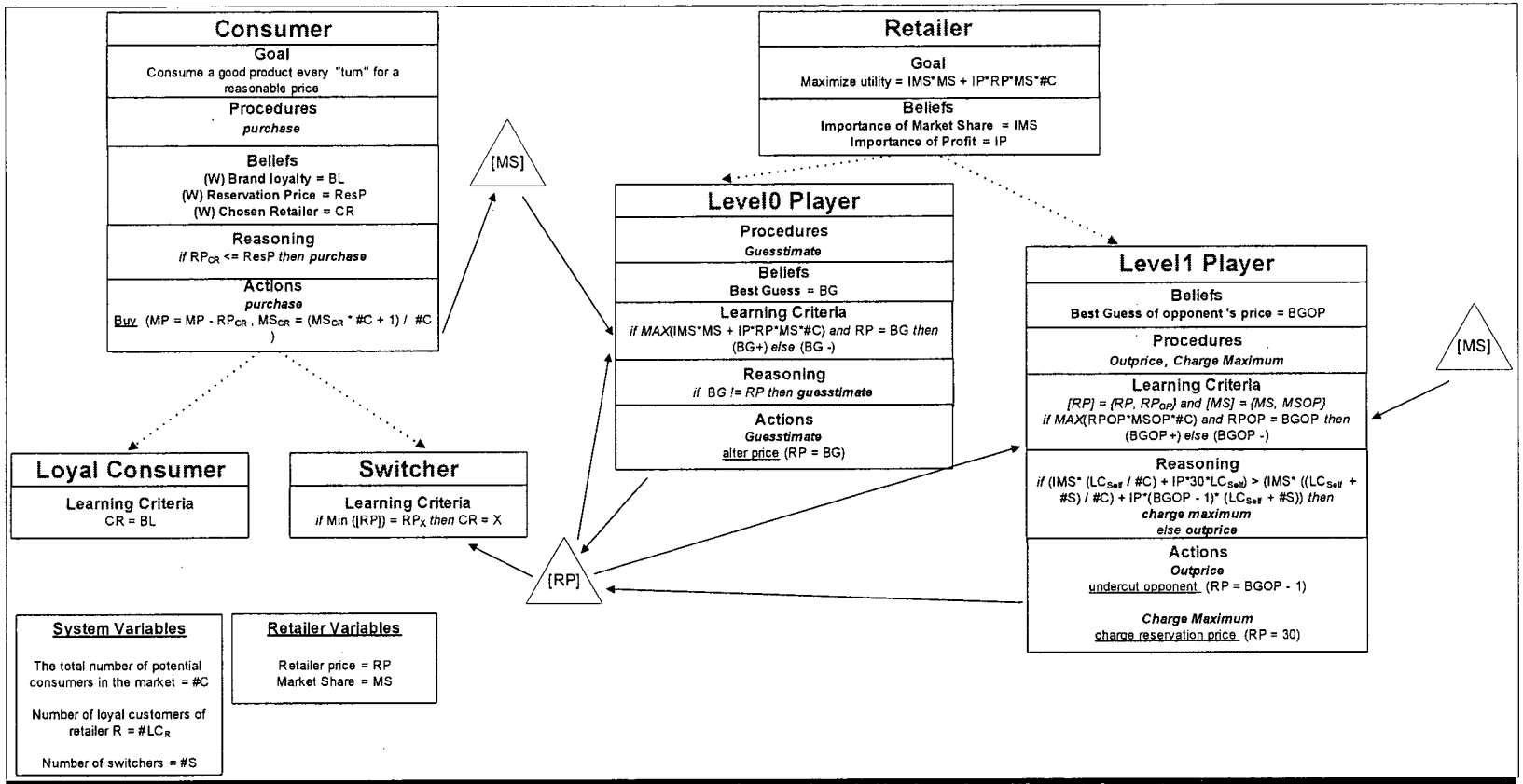

Figure 17: Narasimhan's Pricing Game Modelled as Intelligent Agents

In Figure 17, it can be seen that there are two main agents, retailers and consumers. Consumers and retailers come in two sub types, loyal consumer and switcher, and level0 and level1 players. The consumers are very simple, just like in Narasimhan's model and follow a predictable pattern determined by the retailers' actions. The retailer's goal is to maximize its utility (a mixture of revenue and market share) and the agents interact through the RP, or retailer price, variable given to each retailer. By manipulating the utility functions and subtypes of the competing firms the researcher should have the tools to understand the development of Edgeworth cycles in the game.

5.4 Implementing the Representational Model

Once the agent framework was created, we implemented the model on two different platforms. First we used the agent-based platform Netlogo to see if the representation could be translated into the code. And secondly, due to the limitations of Netlogo for the study, Dr. Dan Putler implemented an earlier version of the conceptual framework of the simulation using the R language. However, before we detail how this occurred we must first discuss the learning algorithm used in the study.

5.4.1 Q-Learning

The algorithm we used in the study was q-learning which is a reinforcement technique which helps agents learn how to maximize a specific function (Sen and Weiss, 1999). The possible actions that an agent can take is each given a "score", the higher the score, in relation to the scores of the other possible actions, the more likely the agent is to choose it. Therefore if the score for each action is the same, the agent has an equal chance of choosing any action. As the simulation progresses, the agent changes the propensity of its actions based on how much of a "reward" its past actions have given it. Therefore, if a specific action gives a consistently high reward, it will be more likely to be performed by the agent. The function works exactly the same when learning chess or which route is the fastest on a highway, the only differences are the objective function, perception, learning conditions, and the "belief" being learned.

One of the reasons we picked this algorithm was because it was used in ACE literature (Tesfatsion, 2005). Below we will show a formal explanation of the q-learning

algorithm and discuss the assumptions behind the technique.

First, let us define the agent's choice of actions as the set "AC". AC contains Z_{hj} number of elements within it. Each of these elements has a q-score which is contained in the set q_{ji} with i ranging from 1 to Z_{hj}. As stated previously, the q-scores are used to determine if an agent will choose a specific action in set AC. This can be calculated by finding the choice probability of the action at a specific time (T) or $p_{ji}(T)$. The equation below shows how $p_{ji}(T)$ is calculated.

$$p_{ji}(T) = \frac{\exp\left(q_{ji}(T) / C_{Hj}\right)}{\sum_{m=1}^{Z_{Hj}} \exp\left(q_{jm}(T) / C_{Hj}\right)}$$

It can be seen in the equation that the choice probability of an action is a ratio of all q-scores in the set q_{ji}. If the specific q-score for the action is high it will have a higher choice probability than if it is low. The variable p_{ji} (T) for i is always between 1 and 0. If $p_{ji}(T) = 0.5$ then there is a 50% chance that the corresponding action/belief will be picked by the agent. Along with the q-score, another major determinant of the choice probability is C_{hj}, or what is also called the cooling factor (Tesfatsion, 2005). The cooling factor makes sure that if the q-score is very large it will not "swamp" the rest of the actions and so it can be considered as a way of minimizing the effect of the learning the agent has received over time.

Since we are dealing with a learning algorithm, these q-scores are updated periodically. Below is the equation for the updated q-score or $q_{ji}(T+1)$.

$$q_{ji}(T+1) = \left[1 - r_{Hj}\right] \cdot q_{ji}(T) + r_{Hj} \cdot \text{Response}_{ji}(T)$$

This equation introduces the idea of a learning rate or r_{Hj}. A learning rate is a way to change how fast the agent learns over time. If r_{Hj} is high, then the agent will discount its previous learning but incorporate what it has recently learned ($Response_{ji}$ (T)), however if the recency effect is low then the agent will slowly incorporate new data into its q-score.

Before we move on though, we must show how the $Response_{ji}$ (T) is calculated.

$$\text{Response}_{ji}(T) = \begin{cases} [1 - e_{Hj}] \cdot \text{Reward}_{ji'}(T) \text{ if } i = i' \\ e_{Hj} \cdot q_{ji}(T) / [Z_{Hj} - 1] \text{ if } i \neq i' \end{cases}$$

There are two parts to the above function, one for i' and the other for i. The variable i' refers to the action being performed at time T while the variable i refers to the other actions. For example, if the retailer in our simulation had the choice of charging $5.00 or $10.00 for its item, then the agent would have two q-scores for each "action". If the agent had just priced its item at $5.00, i' would refer to the q-score for $5.00.

The agent takes into account the "reward" it has received for performing its most recent action (in our example, charging $5.00 for the product) and all of the other q-scores are updated to "keep up". We also introduce another variable in this equation called the experimental parameter or e_{Hj}. This variable affects how much the agent will deviate from performing its "optimal" choice. If e_{Hj} is high, the reward will be discounted and the other q-scores will have a higher "response" value. However, if it is low, the agent's non " i' " q-scores will not change.

As mentioned previously, there are some assumptions in the learning algorithm. Since the algorithm is a optimization technique, modellers should only use it if optimizing agents make sense in their models (Tesfatsion, 2002). Also, the technique does not distinguish the range and space of the different actions. For example, if a retailer found that at time T $6.00 was optimal but at time T+1 it was not, it would randomly pick a price based on q-scores. This is fine in a set of actions which are completely discreet but if we deal with actions that are continuous, or ranked, then the concept of range may be useful. We may want our retailer to first pick the space "around" $6.00 before it chooses an action that is "far away".

5.4.2 Netlogo Implementation

We used the platform discussed earlier, Netlogo, to implement the system. There were many other available agent development platforms, such as Swarm (Hiebeler, 1994) and Repast (Collier, 2002). However, Netlogo is one of the simpler platforms and seemed to fit the purposes of the study nicely.

In section 4.1 we discussed how to turn the representation into a Netlogo program. The same approach was taken with this simulation. In Figure 17, the system variables are #C (the

number of consumers), $\#LC_R$ (number of loyal consumers for each retailer R) and #S (the number of switchers). In Netlogo the code is:

```
globals [
    #C
    #S
    #LC1
    #LC2
]
```

Globals can also be defined in the interface of the system. To give the user more freedom we incorporated the number of consumers in slide rule interfaces.

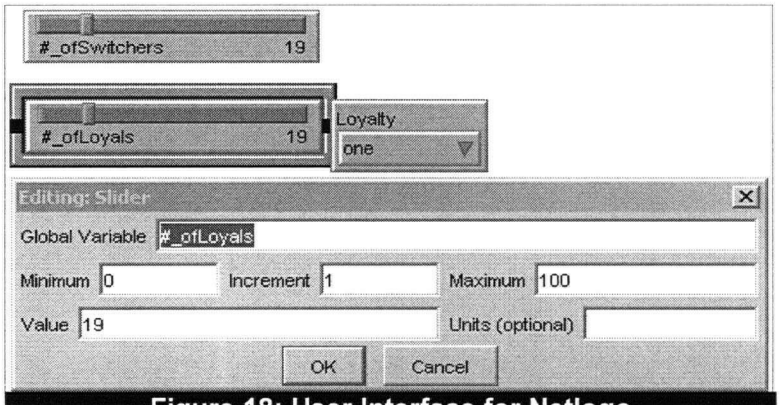

Figure 18: User Interface for Netlogo

In Figure 18, #S is #_ofSwitchers (the number of switchers in the market) and $\#LC_R$ is #_ofLoyals (the number of loyal customers in the market). #C is the sum of the loyal and switcher consumers. All interface variables are considered by the Netlogo environment to be global variables.

Other than the system variables discussed, global variables can also include "operational" variables that might be used to provide information about the program, e.g. the number of turns that the simulation has completed.

Once global variables had been declared, we continued to define the different agents, retailers and consumers. We then gave the program a means of progression. The system had to know which agents to call on and in what sequence the agent tasks had to be implemented. For this purpose, we used the generic progression suggested in section 4.1.3:

```
to run
   learn
   reason
```

```
    act
end
```

The definition of these procedures, and the entire Netlogo code, can be found in Appendix A. In the appendix, the code that is bold is code that can not be generated directly from the representation.

5.4.3 R Implementation

We did not originally plan to use the R language to create another version of the simulation. However when we first created the simulation using Netlogo, we found that the platform did not support many of the statistical analysis tools that were needed to make sense of the data.

We used R to implement the system because it is already used to analyse real world economic data (Francisco & Spyro, 1999) and therefore contains all the tools needed to analyse the simulated data from an agent-based economy. This unforeseen circumstance however gave us the opportunity to see if the graphic representation could be used to implement an agent-based system in a platform that was not originally designed to support agent concepts. R is an object-oriented language that is integrated with statistical tools and data analysis techniques. R was designed to have the flexibility of a full programming language like perl, but includes useful data analysis tools like SAS (R Core Development Team, 2004).

In R, all objects have attributes (variables, classes, etc.) and perform functions by calling "methods". Objects can come in many forms; lists, vectors of variables, and mathematical functions are all types of R objects.

The R implementation was done by Dr. Dan Putler who has extensive knowledge of R. He used an earlier version of the above representation of the system to implement the agent system in this object-oriented language. Since R is an object-oriented language, the modeller must make all of the needed objects that will be part of the simulation. One of the first objects created was the "game" object. Below is the game object creation code:

```
nGame <- function(gameEnv, playerA, playerB, plots=FALSE, pnts=FALSE)
```

A simulation consists mainly of the nGame function which consists of an environment (or market) and two players. The other two arguments are graphic display options.

Another difference between the R implementation and the Netlogo one is that the

consumer agents were conceptualised as a percentage of an overall "market" variable rather than as individual agents. This was done to more closely simulate the scenario used in the Choi and Messinger experiment. The only intelligent agents in the simulation, and the study, were the two retailers. The object type game environment is as follows:

```
gameEnv <- list(rounds, nCust, capA, capB, minPri, maxPri,priIncr, cost)
```

The attribute "rounds" is the number of times the simulation will run and nCust is the number of customers in the simulation. The attribute capA is the share of loyal customers for Firm A. Therefore if capA = 0.3 that means that 30% of the market is a loyal segment of Firm A and capB is the share of firm B. The switcher segment is what is left after A and B's loyal segments are accounted for. The min and max Pri are the minimum and maximum prices that the retailer can set, and priIncr is the increment that the retailers can set prices, so if priIncr was equal to one, it would mean that the retailers can only lower or increase their price in increments of a dollar.

As stated previously, the consumers are not agents in the sense that they are in Netlogo, but are an abstraction in an object, however retailers are real agents. Below is the R code for the retailer agent sub type, Player 0.

```
Player0 <- list(learn=qLrnLvl0, lParam=c(0.8, 0.8, 0.65), objective=utility,
oParam=0.75)
```

This agent has four attributes. The first is the type of learning (learn) and the learning algorithm parameters (lParam) that the agent uses. The next variables are the agent's utility function (objective) and the corresponding goal parameters (oParam). Reasoning was implemented into the learning algorithm and actions were implemented in the simulation mechanics.

Other attributes of the agent, like price and market share, were implemented as "global" variables that would be manipulated by specific agents.

The R implementation of the model did not directly follow the model of agent reasoned action as outlined in Section 3.2.2. However, our conceptual framework of an intelligent agent did provide a way to implement these agent concepts in an object-oriented platform.

5.5 Results from the Simulation

As stated in Section 5.2, the reason for creating the simulation was to find the conditions for when partial Edgeworth cycle pricing behaviour occurs. To do this we had to run the simulation in various configurations. Before we could do that however, we had to set the common scenario for our various simulations.

We decided that we would have sixty-three consumer agents; twenty-one as switchers, and forty-two as brand loyals, evenly split between the two retailers. We then set the parameters for the learning function.

As mentioned in Section 5.4.1, q-learning involves a number of parameters that need to be set by the modeller. The retailer agents in our simulation had to determine, in a reasonable number of turns, the best action against an opponent who constantly set the same price. If one person is pricing constantly, at say 18, the best strategy is to price at 17. We wanted to make sure that the agent could learn the best strategy in a few turns, but would still experiment if the situation changed. Using trial and error, we found that the parameters that gave us the best results were 0.65 for the experimental factor, 0.8 for the learning rate variable, and 16 for the cooling factor.

Now that we had a common setting, we tested our first hypothesis which was that agents who cared more about market share would be more likely to engage in Edgeworth cycle behaviour. Also, according to literature, we believed that the different strategy levels proposed by Vidal and Durfee would have an effect on pricing, however, we were not sure what it would be. We first tested strategy level 0 players competing against each other, but to test our hypothesis we ran two simulations, one where the agents' goal function was weighted towards profit and the other with the function weighted towards market share.

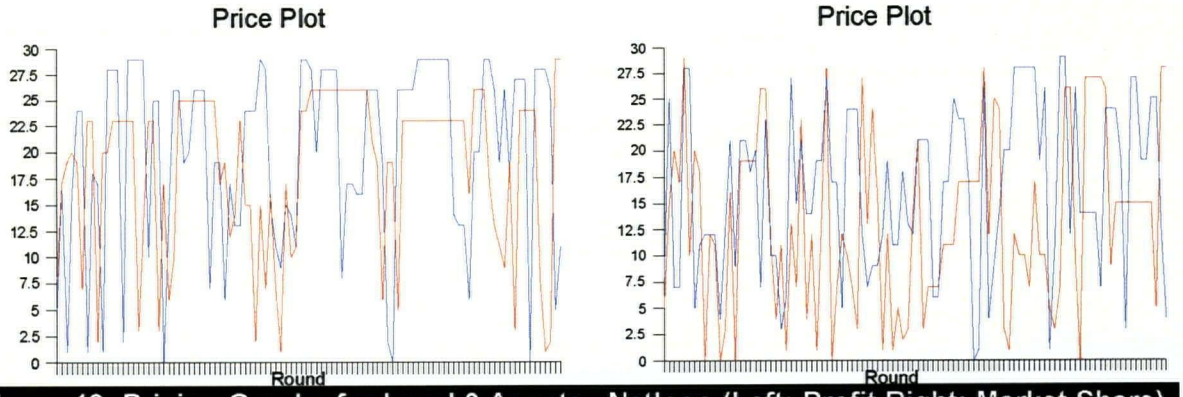

Figure 19: Pricing Graphs for Level 0 Agents– Netlogo (Left: Profit Right: Market Share)

As shown in Figure 19, we found very little difference between the two graphs. In both simulations the agents seem to randomly set prices, and stay at the same price for extended periods of time. Therefore our first hypothesis, that agents who care about market share will try to chase the switcher market using Edgeworth cycle pricing, was false. To see if strategy levels would change pricing behaviour, we ran two simulations with strategy level 1 retailers, again varying the weighting of market share and profit in the agents' goal functions.

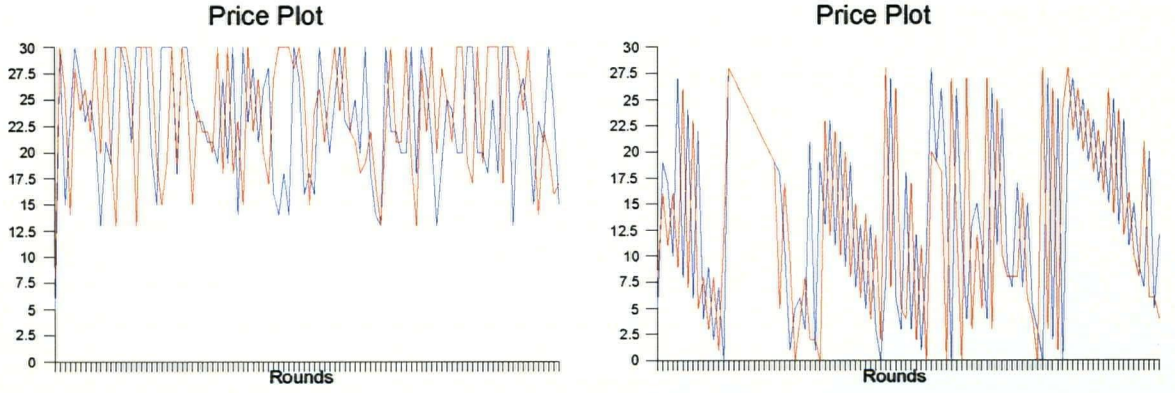

Figure 20: Pricing Graphs for Level 1 Agents– Netlogo (Left: Profit Right: Market Share)

It can be seen from Figure 20, that the agents do engage in Edgeworth cycling behaviour, no matter what the weightings of their goal functions may be. Therefore we can say that the Strategy level of the agent is the deciding factor for Edgeworth cycling behaviour in the Narasimhan pricing game. To engage in a price war the retailers must be trying to model each other's actions.

As we mentioned previously, in section 5.4.3, we also created an R implementation

to better analyse the data. Since we implemented the consumer agents differently in the R representation, we had to change the cooling factor to 0.8 to create the same realistic behaviour. Along with this change, we added the capability for researchers to put two strategy level types in the same game, meaning that we can create a game with a strategy level 0 player and a strategy level 1 player, as shown in Figure 21.

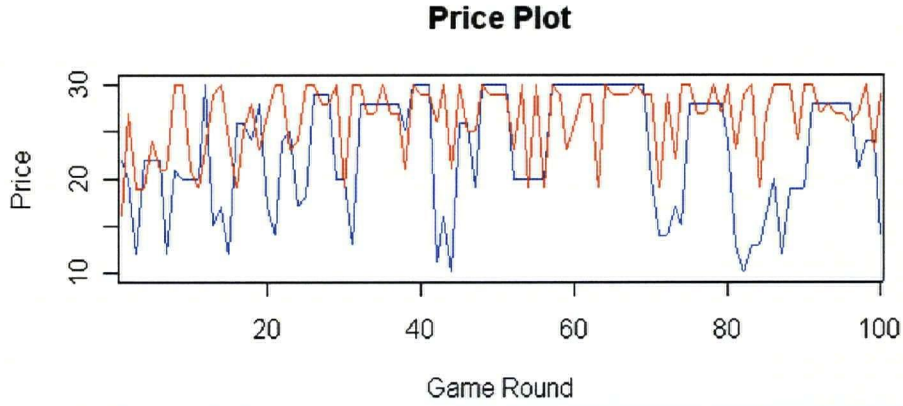

Figure 21. Pricing Graph with Strategy Level 0 and 1 Players– R

5.6 Results from the Application of the Conceptual Framework

The test has shown that the representation can be used either as a way to implement intelligent agents in an agent implementation environment or an object-oriented programming environment. This is useful because it validates that the representation does not only work in programming environments tailored to agents but in other programming languages as well.

The simulation provided opportunities for the marketing researchers to explore certain aspects of the problem that were previously intractable, learning models. As stated before, initially the focus on the investigation was on how different objective functions (goals) could affect behaviour, a mainstay of traditional economic thinking. However, the simulation found that the learning models are what significantly changed how the agents interacted. Also, the application was even better than expected since the representation was used to introduce agent-based concepts into a language that was not initially made to develop intelligent agents.

Preliminary results from the simulation are still being analyzed, but what we have found so far has proven very interesting. One of the first hypothesises was that Edgeworth cycling would occur if one player was trying to attain market share and that pure randomization would occur when both agents were only interested in profits. However this was not the case. We

found that strategy level 0 players would randomize then indirectly show colluding behaviour, while strategy level 1 players competing against each other in the price game would engage in price wars, resulting in Edgeworth cycle pricing patterns. We now must compare our research to that of Choi and Messinger's and see if the graphs generated by the simulation match the pricing behaviour in the empirical study. If so, we can truly say that the results generated by the simulations are consistent with those created by human beings playing the same game.

Chapter 6: Conclusion and Future Research

The study began by asking how intelligent agents could be used to model the complex situations in social science. We analysed the current work on agent-based simulations and models by investigating various ACE market simulations which used agents that learned from their environment. We decided a conceptual model of intelligent agents was needed to aid researchers to communicate and compare their models. In our investigation, we identified the high level components needed to describe an intelligent agent using systems theory and the BWW ontology to guide us through the process of analysing the components of the intelligent agent system. We then discussed the implications of the model and provided a brief look at the modelling rules and benefits of a deeper ontological analysis of the model.

We also provided a graphic representation of this conceptual model, showed how the representation could be implemented in an existing agent development platform, and presented the beginnings of a visual interactive programming tool for intelligent agent development. We then tested the graphic representation by using it to develop a multi-agent simulation to solve a marketing problem. The test proved successful and not only did the simulation provide some early surprising results, but the graphic representation was able to guide the marketing researcher in implementing the simulation on agent-based and object-oriented platforms. Therefore, we propose that our intelligent agent framework can be used to graphically model complex phenomena in the social sciences.

The other contributions of this study are; more formally defined and theoretically grounded agent concepts, and a definition of the interrelationships between these concepts and how they relate to the intelligent agent.

As more non-technical users start to see the value of agent-based modelling, more sophisticated tools, like agent conceptual modelling, are needed to aid in the creation of these

systems.

Also, by analysing our framework it is possible that we could derive rules of intelligent agent behaviour and modelling. For example, to accurately say that a thing in the environment is an agent there must be:

1) A mutual attribute of the agent and the environment that is changed by an external event.
2) A mutual attribute of the agent and the environment that is changed by an internal event.
3) An external event that puts the agent into an unstable state and eventually leads to an event in the world.

What this means is that for a thing to be modelled as an agent it must be able to 1) monitor part of its environment, 2) change part of its environment, and 3) react to a change in the environment.

We can also see how the simulator and effector interact by using the conceptual model. For example, we can say that the beliefs of the agent that are modelled must be used in its reasoning. This must be true because when the agent reasons (changes its procedures), it is put into an unstable state leading to changes in the intrinsic properties of the simulator (the agent's beliefs).

A further analysis of the model can lead to other interesting results. For instance, using our definition of a goal and how agents deal with the environment we can define an achievable goal. An achievable goal of the agent would be a stable state of the agent's simulator that represents a state of the world where 1) there is at least one joint attribute of the world and the effector in the state of the world, and 2) the world is in a stable state. What this means is that the agent must have control over something in the environment that will lead to its goal. The second statement is related to the three rules stated above. If an agent reacts to the environment, the environment must always be in the "goal" state for the agent to have achieved its goal. If the goal state of the agent represents an unstable state of the world then once the agent has changed the world to a goal state, the environment will move to another state, and its goal will no longer be achieved.

Another aspect of intelligent agent behaviour that we can define is interaction. Since agents can only affect the world (agent or global variables) we can say that there is no direct contact between one agent's effector and another agent's simulator. Interaction only occurs when one agent changes another agent's perceptions.

Another definition can be the discussion of what occurs when an agent's simulator

changes. If the simulator's state moves from stable to stable by an external event, it means that the agent is monitoring the environment. If the agent's simulator changes from stable to unstable then the agent is making a decision. If the agent's simulator changes from unstable to unstable then it would be an internal event and would represent the agent learning something new using the information it has received from the environment.

As mentioned previously, this is only a small sample of what can be accomplished by deeply analysing the model. However, at a cursory glance we can see that many of the intelligent agent concepts used in literature can be explained using the semantics of this model.

However, the current study does have some limitations. Only one problem in one research area was modelled. It is possible that intelligent agents are not useful for modelling a non-marketing, or a non-competitive strategic problem. Also, the conceptualisation of intelligent agents was tested by creating an agent-based simulation, but agents can also be used to perform tasks in the real world (Wooldridge, 2002). It is possible that the representation may not have the tools to accurately model the concept of a dynamic artificial intelligence that acts in the real world.

Therefore, one avenue of future research could study if the proposed representation can help develop agents to perform routine (Kitts & LeBlanc, 2004) or dangerous (Turner, 1998) real world tasks. These tasks are different from the simulated environment tasks because agents need to operate within dynamic environments which may contain elements that even the programmers are not aware of.

Another possible research study could be the development of a modelling/ programming environment. Non-technical users could automatically create multi-agent based simulations using the graphic representation, as discussed in Section 4.2. Not only would this help non-technical users develop coherent agent simulations, but because of its standard language it could be used as a powerful communication tool.

References

1. Ackoff, R.L. (1971). Towards a System of Systems Concepts. *Management Science*. Vol. 17 (11), pp. 661–671.

2. Angulin, Dana (1992). Computational Learning Theory: Survey and Selected Bibliography. *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing.*
3. Arazy, Ofer and Woo, Carson C. (2002). Analysis and design of agent-oriented information systems. *The Knowledge Engineering Review*. Vol. 17 (3), pp. 215-260.
4. Aumann, R.J. and Brandenburger, A. (1995). Epistemic conditions for Nash equilibrium. *Econometrica* Vol. 63, pp. 1161–1180.
5. Axelrod, Robert (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press. Princeton, NJ .
6. Ben Said, Lamjed; Bouron, Thierry and Drogoul, Alexis (2002). Agent-based Interaction Analysis of Consumer Behavior. In *Proceedings of the First Joint Conference on Autonomous Agents and Multiagent Systems*. pp. 184–190.
7. Bertalanffy, Ludwig von (1968). *General Systems Theory: Foundations, Development, Applications*. George Braziller. New York, NY.
8. Bunge, M. (1977). *Treatise on Basic Philosophy. Vol. 3, Ontology I: The Furniture of the World*. Reidel, Boston, Massachusetts.
9. Bunge, M. (1979). *Treatise on Basic Philosophy. Vol. 4, Ontology II: A World of Systems*. Reidel, Boston, Massachusetts.
10. Casti, John (1989). *Paradigms Lost: Tackling the Unanswered Mysteries of Modern Science*. Avon Books. New York, NY.
11. Choi, Sungchul and Messinger, Paul R. (2004). *Edgeworth Cycles and Fairness in Competitive Promotional Strategies: An Experimental Study of Mixed Strategy Pricing.*
12. Churchman, C.W. (1979). *The Systems Approach*. Dell Publishing. New York, NY.
13. Collier, Nick (2002). *RePast: An Extensible Framework for Agent Simulation.* http://tinyurl.com/3lp2k.
14. Conte, Rosaria; Gilbert, Nigel and Sichman, Jamie Simão (1998). MAS and Social Simulation: A Suitable Commitment. In *First International Workshop MABS'98 Proceedings*. Paris, France. pp. 1–9.
15. Drogoul, A.; Vanbergue, D.; and Meurisse, T. (2002). Multi-Agent Based Simulation: Where Are The Agents? *Proceedings of MABS'02* (Multi-Agent Based Simulation), Bologna, Italy, July 2002. LNCS, Springer-Verlag.

16. Foil, C. Marlene and Huff, A. Sigismund (1992). Maps for Managers: Where are we? Where do we go from here?. *Journal of Management Studies.* Vol. 29 (3), pp. 265–285.
17. Francisco, Cribari-Neto and Spyros, G. Zarkos (1999). R: Yet another econometric programming environment. *Journal of Applied Econometrics*. Vol. 14, pp. 319-329.
18. Harsanyi, J.C. (1973). Games with randomly disturbed payoffs: A new rationale for mixed-strategy equilibrium points. *International Journal of Game Theory* .Vol. 2, pp. 1–23.
19. Hiebeler, David (1994). The Swarm Simulation System and Individual-based Modeling. In *Proceedings of Decision Support 2001*, Toronto, Canada.
20. Holland, John H.; Holyoak, Keith J.; Nisbett, Richard E. and Thagard, Paul R. (1986). *Induction: Processes of Inference, Learning and Discovery*. MIT Press. Cambridge, MA.
21. Kim, Byung-In; Graves, Robert J; Heragu, Sunderesh S. and St. Onge, Art (2002). Intelligent Agent Modeling of an Industrial Warehousing Problem. *IIE Transactions.* Vol. 34 (7), pp. 601–612.
22. Kirman, Alan P. and Vriend, Nicolaas J. (2001). An ACE Model of Price Dispersion and Loyalty. *Journal of Economic Dynamics and Control*. Vol. 25, pp. 459–502.
23. Kitts, Brendan and LeBlanc, Benjamin (2004). Optimal Bidding on Keyword Auctions. *Electronic Markets*. Vol. 14 (3), pp. 186-201.
24. Lakoff, G. (1987). Cognitive models and prototype theory. In Neisser, U. (Ed.), *Concepts and Conceptual Development: Ecological and Intellectual Factors in Categorization.* Cambridge University Press. Cambridge, MA.
25. Miller, James Grier (1978). *Living Systems*. McGraw-Hill. New York, NY.
26. Nadoli, Gajanana and Biegel, John E. (1993). Intelligent Manufacturing Simulation Tool (IMSAT). *ACM Transactions on Modelling and and Computer Simulation*. Vol. 3 (1), pp. 42–65.
27. Narasimhan, C. (1988). Competitive Promotional Strategies. *Journal of Business.* Vol. 61, pp. 427-450.
28. Parunak, H. Van Dyke; Savit, Robert and Riolo, Rick L. (1998). Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. In *First International Workshop MABS'98 Proceedings.* Paris, France. pp. 10–25.
29. Plotkin, D. (1999). "Jack of All Trades". *Intelligent Enterprise*. San Mateo: Vol. 2 (7), pg. 52.

30. R Development Core Team (2004). *An introduction to R.* The R Foundation for Statistical Computing. http://www.r-project.org/.
31. Reny, P. J. and Robson, A. J. (2004). Reinterpreting mixed strategy equilibria: a unification of the classical and Bayesian views. *Games and Economic Behaviour.* Vol. 48 (2), pg. 355.
32. Schelling, T.C., (1960). *The Strategy of Conflict.* Harvard Univ. Press, Cambridge, MA.
33. Sen, Sandip and Weiss, Gerhard (1999). Learning in Multiagent Systems. In Weiss, Gerhard. (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, MA.
34. Shehory, Onn and Sturm, Arnon (2001). Evaluation of modeling techniques for agent-based systems. *Proceedings of the Fifth International Conference on Autonomous Agents.*
35. Swaminathan, Jayashankar M.; Smith, Stephen F. and Sadeh, Norman M. (1998). Modeling Supply Chain: A Multiagent Approach. *Decision Sciences.* Vol. 29 (3), pp. 607-632.
36. Tesfatsion, Leigh (2002). Agent-Based Computational Economics: Growing Economies from the Bottom Up. *Artificial Life*, Vol. 8 (1), pp. 55-82.
37. Tesfatsion, Leigh (2005). Agent Based Computational Economics: A Constructive Approach to Economic Theory. Forthcoming in Judd, K.L. and Tesfatsion, L. (Eds.) *Handbook of Computational Economics.* Handbooks in Economics Series, North-Holland.
38. Tisue, S., and Wilensky, U. (2004). NetLogo: Design and Implementation of a Multi-Agent Modeling Environment. *Proceedings of Agent 2004.*
39. Turner, Roy M. (1998). Context-mediated behavior for intelligent agents. *Int. Journal of Human-Computer Studies.* Vol. 48, pp. 307–330.
40. van der Zee, D.J. and van der Vorst, J.G.A.J. (2005). A Modeling Framework for Supply Chain Simulation: Opportunities for Improved Decision Making. *Decision Sciences.* Vol. 36 (1), pp. 65-95.
41. Vidal, J.M. and Durfee, E.H. (1996). The impact of nested agent models in an information economy. In *Proceedings of the Second International Conference on Multiagent Systems.*

42. Wand Y. and Weber R. (1990). An ontological model of an information system. *IEEE Transactions on Software Engineering.* Vol 16. pp. 1282–1292.
43. Wand Y. and Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*. Vol. 3, pp. 217 – 237.
44. Wand, Yair and Weber, Ron (1995). On the deep structure of information systems. *Information Systems Journal*. Vol. 5, pp. 203–223.
45. Wilensky, U. (1999). *NetLogo (and NetLogo User Manual)*, Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/
46. Wooldridge, M. (2002). *Introduction to Multi-Agent Systems*. John Wiley and Sons Limited. Chester.

Appendix A: Netlogo Code

```
globals[
      turns
      r
      ex
      C
      range
]


breeds [
      consumer
      retailer
]


consumer-own [
      CR
      BL
      Purchase
]


retailer-own [
      name
      RP
      MS
      MP
      turnprofit
      charge_max
      outprice
      guesstimate
      IMS
      accum
      IP
      step
      probtotal
      rand
      op_score
      op_prob
      opponent
      BGOP
      strat_score
      strat_prob
      cost
      BP
      strategy
]


to learn

      ask consumer [
            ifelse brandloyalty != "none"[
                set CR one-of retailer with [name = brandloyalty-of myself]
            ]

            [set CR min-one-of retailer [RP]]
```

```
      ]

      ask retailer with [strategy = level0] [
           update-score
           update-prob
           bestchoice
      ]

      ask retailer with [strategy = level1][
           update-op-score
           update-op-prob
           bestguess
      ]
end


to reason
      ask retailer with [strategy = "level1"] [
           ifelse ((IMS * (count consumer with [brandloyalty = name-of myself]
           / count consumer) + IP * 30 * count consumer with [brandloyalty =
           name-of myself]) > (IMS * ((count consumer with [brandloyalty =
           name-of myself] + count consumer with [brandloyalty = "none"]) /
           count consumer) + IP * (BGOP - 1) * (count consumer with
           [brandloyalty = name-of myself] + count consumer with [brandloyalty
           = "none"])))[

               set outprice false
               set charge_max true
               set guesstimate false
           ]
           [
               set outprice true
               set charge_max false
               set guesstimate false
           ]
      ]

       ask retailer with [strategy = "level0"] [
           ifelse BP != RP [
               set guesstimate true
           ]
           [set guesstimate false]
       ]
end


to actions
      ask retailer [
           if charge_max = true[
               set RP 30
           ]
           if outprice = true and BGOP > 0  [
               set RP (BGOP - 1)
           ]
           if outprice = true and BGOP <= 0 [
               set RP 0
           ]
           if guesstimate = true [
               RP = BP
           ]
      ]
```

```
      ask consumer [
           if purchase = true [
               set MP-of CR (MP-of CR + RP-of CR - cost-of CR)
               set turnprofit-of CR (RP-of CR + turnprofit-of CR - cost-of CR)
               set MS-of CR (((MS-of CR * count consumer) + 1)/ count
               consumer)
           ]
      ]
end


to update-score
      set step 0

      while [step < (range)][
           ifelse step = RP[
               set accum (1 - ex) * (IMS * MS + MS * (RP - cost) * count
               consumer)
           ]
           [set accum ex * (item step strat_score / range - 1)]

           set op_score replace-item step strat_score ((1 - r) * item step
           strat_score + r * accum)
           set step (step + 1)
      ]
end


to update-prob
      set step map [exp (? / C))] strat_score
      set probtotal sum step
      set strat_prob map [(exp (? / C))) / probtotal] strat_score
end


to update-op-score
      set step 0

      while [step < (range)][
           ifelse step = RP-of opponent [
               set accum (1 - ex) * (MS-of opponent * (RP-of opponent - cost)
               * count consumer)
           ]
           [set accum ex * (item step op_score / range - 1)]

           set op_score replace-item step op_score ((1 - r) * item step
           op_score + r * accum)
           set step (step + 1)
      ]
end


to bestchoice
      set rand random-float 1
      set step 0
      set probtotal 0
      while [step < (range)][
           ifelse step = 0 [
               if rand <= item step strat_prob [
                   set BP 0
```

```
            ]
        ]
        [
            set probtotal (probtotal + item (step - 1) strat_prob)
            ifelse step = range - 1 [
                if rand > item step strat_prob and rand <= 1 [
                    set BP step
                ]
            ]
            [
                if rand > probtotal and rand <= (item step strat_prob) +
                probtotal
                [
                    set BP step
                    set step range + 1
                ]
            ]
        ]
        set step (step + 1)
    ]
end


to update-op-prob
    set step map [exp (? / C))] op_score
    set probtotal sum step
    set op_prob map [(exp (? / C))) / probtotal] op_score
end


to bestguess
    set rand random-float 1
    set step 0
    set probtotal 0
    while [step < (range)][
        ifelse step = 0 [
            if rand <= item step op_prob [
                set BGOP 0
            ]
        ]
        [
            set probtotal (probtotal + item (step - 1) op_prob)
            ifelse step = range - 1 [
                if rand > item step op_prob and rand <= 1 [
                    set BGOP step
                ]
            ]
            [
                if rand > probtotal and rand <= (item step op_prob) +
                probtotal
                [
                    set BGOP step
                    set step range + 1
                ]
            ]
        ]
        set step (step + 1)
    ]
end
```

Appendix B: R Code

```
nGame <- function(gameEnv, playerA, playerB, plots=FALSE, pnts=FALSE) {

    #INITIALIZE PROPENSITIES, CHOICES AND OUTCOMES
    posPrices <- seq(gameEnv$minPri, gameEnv$maxPri, gameEnv$priIncr)
    Qa <- matrix(rep(NA, gameEnv$rounds*length(posPrices)),
      nrow=gameEnv$rounds, ncol=length(posPrices))
    Qb <- Qa
    priceA <- rep(NA, gameEnv$rounds)
    profitA <- rep(NA, gameEnv$rounds)
    shareA <- rep(NA, gameEnv$rounds)
    priceB <- rep(NA, gameEnv$rounds)
    profitB <- rep(NA, gameEnv$rounds)
    shareB <- rep(NA, gameEnv$rounds)
    playerA$myCap <- gameEnv$capA
    playerB$myCap <- gameEnv$capB
    gameEnv$swtchSeg <- swtchSeg # append the switcher seg size to the game
env

    # THE GAME ROUNDS
    # The first round is different from the others
    # Here is my initial shot at the first round
    Qa[1,] <- rep(1/length(posPrices),length(posPrices))
    Qb[1,] <- Qa[1,]
    priceA[1] <- round((gameEnv$minPri+0.4999) +
      (gameEnv$maxPri-gameEnv$minPri)*runif(1))
    priceB[1] <- round((gameEnv$minPri+0.4999) +
      (gameEnv$maxPri-gameEnv$minPri)*runif(1))
    switchA <- swtchSeg*(as.numeric(priceA[1]<priceB[1]) +
      0.5*as.numeric(priceA[1]==priceB[1]))
    shareA[1] <- gameEnv$capA*as.numeric(priceA[1]<=30) + switchA
    profitA[1] <- gameEnv$nCust*(priceA[1]-gameEnv$cost)*shareA[1]
    shareB[1] <- gameEnv$capB*as.numeric(priceB[1]<=30) + (swtchSeg-switchA)
    profitB[1] <- gameEnv$nCust*(priceB[1]-gameEnv$cost)*shareB[1]
    experA <- data.frame(myPri=priceA[1], myPrf=profitA[1], myShr=shareA[1],
      otrPri=priceB[1], otrPrf=profitB[1], otrShr=shareB[1])
    experB <- data.frame(myPri=priceB[1], myPrf=profitB[1], myShr=shareB[1],
      otrPri=priceA[1], otrPrf=profitA[1], otrShr=shareA[1])
    # The second to the terminal rounds
    for(i in 2:gameEnv$rounds) {
        learnA <- playerA$learn(playerA, gameEnv, experA, as.vector(Qa[(i-
1),]))
        Qa[i,] <- learnA$Qval
        priceA[i] <- learnA$price
        learnB <- playerB$learn(playerB, gameEnv, experB, as.vector(Qb[(i-
1),]))
        Qb[i,] <- learnB$Qval
        priceB[i] <- learnB$price

        switchA <- swtchSeg*(as.numeric(priceA[i]<priceB[i]) +
          0.5*as.numeric(priceA[i]==priceB[i]))

        shareA[i] <- gameEnv$capA*as.numeric(priceA[i]<=30) + switchA
        profitA[i] <- gameEnv$nCust*(priceA[i]-gameEnv$cost)*shareA[i]

        shareB[i] <- gameEnv$capB*as.numeric(priceB[i]<=30) + (swtchSeg-
switchA)

        profitB[i] <- gameEnv$nCust*(priceB[i]-gameEnv$cost)*shareB[i]
```

```
        experA <- data.frame(myPri=priceA[1:i], myPrf=profitA[1:i],
          myShr=shareA[1:i], otrPri=priceB[1:i], otrPrf=profitB[1:i],
          otrShr=shareB[1:i])

        experB <- data.frame(myPri=priceB[1:i], myPrf=profitB[1:i],
          myShr=shareB[1:i], otrPri=priceA[1:i], otrPrf=profitA[1:i],
          otrShr=shareA[1:i])
    }
}

qLrnLvl0 <- function(player, gameEnv, experience, Qval) {
    posPrices <- seq(gameEnv$minPri, gameEnv$maxPri, gameEnv$priIncr)
    cool <- player$lParam[1]
    recency <- player$lParam[2]
    exprmnt <- player$lParam[3]
    response <- (1-exprmnt)*rep(1/(length(posPrices)-1),length(posPrices))
    lastPrice <- experience$myPri[nrow(experience)]
    response[posPrices==lastPrice] <- exprmnt*player$objective(player,
      experience[nrow(experience),])
    Qval <- (1-recency)*Qval + recency*response
    probs <- exp(Qval/cool)/sum(exp(Qval/cool))
    price <- min(posPrices[as.numeric(cumsum(probs)>runif(1))==1])
    return(list(Qval=Qval, price=price))
    }

expOpPriLvl1 <- function(player, gameEnv, experience, Qval) {
    posPrices <- seq(gameEnv$minPri, gameEnv$maxPri, gameEnv$priIncr)
    cool <- player$lParam[1]
    recency <- player$lParam[2]
    exprmnt <- player$lParam[3]
    response <- (1-exprmnt)*rep(1/(length(posPrices)-1),length(posPrices))
    lastOpPrice <- experience$otrPri[nrow(experience)]
    response[posPrices==lastOpPrice] <- exprmnt*player$opObj(player,
      experience[nrow(experience),])
    Qval <- (1-recency)*Qval + recency*response
    probs <- exp(Qval/cool)/sum(exp(Qval/cool))
    opPrice <- min(posPrices[as.numeric(cumsum(probs)>runif(1))==1])
    lowPrice <- opPrice - gameEnv$priIncr
    lowShare <- gameEnv$swtchSeg + player$myCap
    lowProfit <- gameEnv$nCust*(lowPrice-gameEnv$cost)*lowShare
    experLow <- list(myPri=lowPrice, myPrf=lowProfit, myShr=lowShare,
      otrPri=opPrice, otrPrf=gameEnv$nCust*(lowPrice-gameEnv$cost)*(1-
lowShare),
      otrShr=(1-lowShare))
    hiPrice <- gameEnv$maxPri
    hiShare <- player$myCap
    hiProfit <- gameEnv$nCust*(hiPrice-gameEnv$cost)*hiShare
    experHi <- list(myPri=hiPrice, myPrf=hiProfit, myShr=hiShare,
      otrPri=opPrice, otrPrf=gameEnv$nCust*(hiPrice-gameEnv$cost)*(1-hiShare),
      otrShr=(1-hiShare))
   if(player$objective(player, experLow) >= player$objective(player,
experHi)) {
        price <- lowPrice
        }
    else price <- hiPrice
    return(list(Qval=Qval, price=price))
    }

utility <- function(player, experience) {
    util <- experience$myPrf*player$oParam +
```

```
      14*experience$myShr*(1-player$oParam)
    return(util)
    }

opUtility <- function(player, experience) {
    util <- experience$otrPrf*player$oParam +
      14*experience$otrShr*(1-player$oParam)
    return(util)
    }


    #Agent and game environment specifications
    Player0 <- list(learn=qLrnLvl0, lParam=c(0.8, 0.8, 0.65),
      objective=utility, oParam=0.75)
    class(Player0) <- "player"

    Player1 <- list(learn=expOpPriLvl1, lParam=c(0.8, 0.8, 0.65),
      objective=utility,  oParam=0.75, opObj=opUtility)
    class(Player1) <- "player"

    base <- list(rounds=30, nCust=1, capA=0.3, capB=0.3, minPri=10, maxPri=30,
      priIncr=1, cost=15)
    class(base) <- "gameEnv"
```