Architecture of a Prediction Economy

by

Jason W. Carver

S.B., Computer Science M.I.T., 2006

S.B., Management Science M.I.T., 2006

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 23, 2008

Author ______________________________
Department of Electrical Engineering and Computer Science
May 21, 2007

Certified by ______________________________
Thomas W. Malone
Patrick J. McGovern Professor of Management
Director, MIT Center for Collective Intelligence
Thesis Supervisor

Accepted by ______________________________
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Architecture of a Prediction Economy

by
Jason W. Carver

Submitted to the
Department of Electrical Engineering and Computer Science

May 23, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A design and implementation of a Prediction Economy is presented and compared to alternative designs. A Prediction Economy is composed of prediction markets, market managers, information brokers and automated trading agents. Two important goals of a Prediction Economy are to improve liquidity and information dispersal. Market managers automatically open and close appropriate markets, quickly giving traders access to the latest claims. Information brokers deliver parsed data to the trading agents. The agents execute trades on markets that might not otherwise have much trading action. Some preliminary results from a running Prediction Economy are presented, with binary markets based on football plays during a college football game. The most accurate agent chose to enter 8 of 32 markets, and was able to predict 7 of the 8 football play attempts correctly. Source code for the newly implemented tools is available, as are references to the existing open source tools used.

Thesis Supervisor:
Thomas W. Malone

Title:
Patrick J. McGovern Professor of Management
Director, MIT Center for Collective Intelligence

Prediction markets are a market-based method for eliciting an aggregate belief from a group. In these markets, people invest in event shares which return a profit if the investor correctly predicted the outcome of the event. As a result of many traders interacting, the market generates and market price. This price can be used to estimate the average of the participants' belief of the state of the world, which can be very useful as an aggregate prediction of an event's probability.[1,2] Here, I discuss an extension to these markets: a Prediction Economy[3]. A Prediction Economy is composed of prediction markets, market managers, information brokers and automated trading agents. Here, I describe an architecture for a Prediction Economy, I discuss some alternative choices we considered over the course of the design, and I detail the actual implementation of the system that we built.

Design

This Prediction Economy was built from three software components: the marketplace itself, automated traders, and a market manager/information broker. The last two were built as a single entity as a matter of convenience. The marketplace shows the definition of the featured event and its payoffs. The marketplace also graphs the price history and helps match buyers and sellers. The automated traders are independent agents created by human traders. These agents can execute trades in the marketplace with each other and other humans. The market manager opens and closes market claims for specific events. It automatically executes these actions when new event claims become available or the outcome of an event has been discovered. The

1 Adams, Christopher, Learning in Prediction Markets, August 2006. Available at SSRN
2 Wolfers, Justin and Zitzewitz, Eric, "Interpreting Prediction Market Prices as Probabilities" (May 2006). CEPR Discussion Paper No. 5676
3 Thanks to Thomas Malone for conceptualizing the Prediction Economy and walking me through the details.

information broker provides agents with semantic details related to the currently trading markets.

There are several marketplace options available. We chose to use Zocalo in this implementation of a Prediction Economy. Zocalo[4] is open source which means we can make the changes required for a Prediction Economy. The most significant necessary change is the API for agent interaction. Most markets assume that all traders are human and can easily use an HTML interface. While it is not difficult for agents to use such interfaces, the agents tend to be much more brittle to changes in the UI. Also, the HTML interface loads a lot of data that will be ignored by the agent, needlessly slowing down the server and the execution of the trade. In the interest of reducing long-term maintenance cost and speeding up interaction, we added an RPC interface for agent interaction.

The trading agents themselves are intended to be built by the traders. We designed the system so that there are as few requirements on the agents as possible. The only requirement is that they interface using a common RPC communication interface. By reducing the design constraints, we encourage as many traders as possible to be involved in the process, because they can implement agents using whatever languages and environments they desire. We implemented several trading agents to demonstrate the concept and provide a framework for future traders to extend.

The market manager was tasked with opening and closing markets at the appropriate time. We could not find an existing system designed to do this so we built our own, called the NewsServer. In order to open and close markets in the marketplace, the market manager has a semantic notion of the market topics and outcomes. With these details, the market manager can

4 Details and source code for Zocalo can be found at: http://zocalo.sourceforge.net/

open and close markets at appropriate times with the correct outcome. This enables an important feature that would otherwise be difficult: short lifespan markets. Using the market manager means that humans do not have to try to precisely time the opening and closing of markets. If a human was required for every step, then running a long series of very short markets would be costly in manpower and prone to error. Instead, the market manager can easily run one-minute or shorter markets with high precision and low cost. In addition to managing the markets, the market manager keeps agents informed about which markets it has opened. We chose to avoid keeping a registry of agents in the market manager, which informed the communication decisions. These decisions are detailed further below.

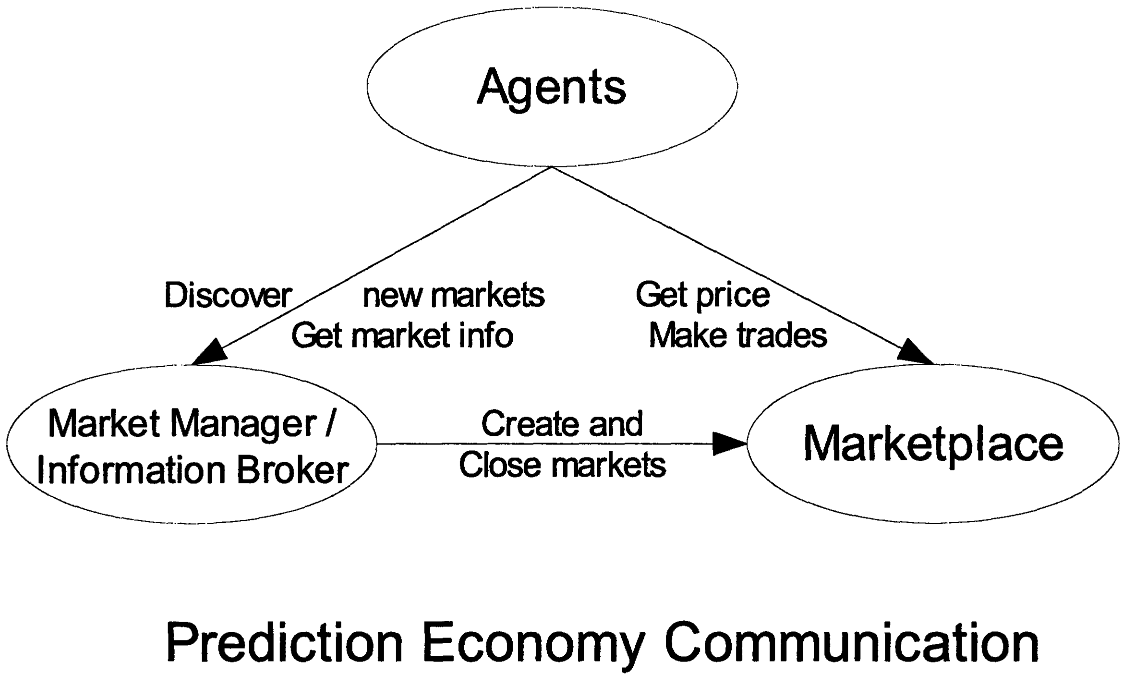
The NewsServer also served the role of the information broker, which is the last main component of a Prediction Economy. The information broker provides relevant details to the agents about the currently open market, in a semantically parsed way. Without an information broker, agents would have to parse unstructured, natural language sources. We avoided a significant difficulty of designing the information broker by only testing it with predetermined events, so all of the information could be transcribed and parsed ahead of time. In a real-time setting, which is more realistic for a prediction market, the information broker would have to have access to a stream of the latest data and parse it with the help of a human or advanced artificial intelligence techniques.

The choice to make the information broker aware of situational semantics had important implications. It expedited transmission of data to the agents and simplified the processing necessary to respond to new data. With semantic data, the agents do not have to parse natural language text to infer information. Instead, they know the exact format of the data coming in, and what it means in context of the relevant market. The disadvantage of this situation is that new code is required for the information broker to handle each new event context. This new code is necessary for the information broker to understand and transfer the new semantic data. Then, agents have to recompile with access to this new code, which defines a library of semantic structure. This process is costly in each new event context. Without solving the general problem of AI, however, understanding new event contexts will continue to be a fundamentally laborious task regardless of the design decisions made here.

Combining the information broker with the market manager may seem like an odd choice. It is important to note that bundling the broker with the manager does not preclude the Prediction Economy from having other information brokers. On the other hand, the complete absence of information brokers would bring the system to a halt, because the manager and agents would have no information with which to make decisions. A dedicated, separate primary information broker is one option to prevent this kind of system halt. However, there are no cases where the market manager can achieve any of its goals independently of the primary broker. This strongly motivated the combination of the two. In this combined setup, the primary information broker distributes the most basic of information to the market manager and agents, but leaves the room for other information brokers to offer premium services with more in-depth data. Supplying the most basic information is critical for agents, which would basically be unable to trade without it. Even if the agents are making poor trades because their information is too basic, they serve to add incentive for more informed players to enter the market and correct the bad bets. In the words of Robin Hanson, "The sheep bring out the wolves." All this motivated the choice to include the primary information broker in the market manager.

All the components of a Prediction Economy are independent: they can be turned on and off independently, and can be run in different locations as long as there is a network connection between them. Because of this distributed nature, it became important to consider the data flow and which components initiate communication. For example, would the market manager contact agents when it opens a new market, or would the agents have to ask the market manager repeatedly if there was any new information? Would the marketplace keep the market manager up to date on the current market price, or would the agents talk directly to the marketplace? We made the following design choices: the marketplace never initiates and the agents always initiate. This marketplace choice was motivated by the idea that a marketplace should be a meeting place, not a service provider. Also, it reduces the requirements for a market to be a part of a Prediction Economy, which encourages competition among marketplace providers. The agents, on the other hand, take the opposite tactic: they never receive any incoming requests. Of all the components, agents are expected to be the least stable. Their owners may bring up or take down the agents at a whim, and change their Internet address regularly. Incoming communications would require the market manager to keep track of which agents have been created, which are currently active and where they are located. Instead, the agents poll the market manager to find out about market updates and the latest semantic information, and the agents must poll the marketplace to find out the current price or to make direct trades. See figure 1 for more details of the communication flow.

FIGURE 1 – The elements of the Prediction Economy initiate communications in the direction of the arrows: the agent initiates all communications and the marketplace initiates none.



Marketplace Alternatives

When locating a marketplace to use in the Prediction Economy, it was very important to us to use a marketplace that was available as open source. We require open source for a couple of reasons. First, we want to encourage public experimentation and extension of this work, which is only possible if all the components are open source. Second, we had to extend the marketplace in order to enable agent trading, which requires access to modify the source. There are two other marketplace packages that are available as open source: Usifex and Idea Futures. Both options have inactive developer communities as of this writing. The website that uses the Usifex[5] marketplace is getting zero trader traffic, and the home page refers to a claim that should have

5 http://www.usifex.com/if/source.html

expired four years ago. Idea Futures[6] by Consensus Point is a bit more recently updated, with the latest update published two and a half years ago, in August 2005. Consensus Point also builds proprietary software on top of Idea Futures and licenses it, so it is likely that if there were any recent improvements, that they went into the proprietary version. The page devoted to open source Idea Futures is bare, and references a demo that does not load. The software may warrant further investigation, though, just based on the list of users: Foresight Exchange (a respected, popular prediction market), Siemens, General Electric, etc.

Agent Location Alternative

Agents are currently hosted by the trader who built them. However, another option we considered was to host the agents at the marketplace. This has several advantages, including intimate market access, access to agent code, agent/human distinction, and lower trader cost for hosting agents.

Agents having close integration with the marketplace have several advantages. They are likely to be able to discover price changes more quickly. They are able to execute trades more quickly. They are more likely to be able to successfully make multiple trades that depend on each other, as in many arbitrage trades. All of these are beneficial to the agents, perhaps even giving agents an unfair advantage over human traders.

If the marketplace hosts agents, it has access to the agent code. This provides some opportunities to share the wealth of agent success. After some fixed period of time, the market could release agent code to improve everyone's understanding of the market by a forced open

6 http://ideafutures.sourceforge.net/

source strategy.

When the marketplace hosts agents, it knows which trades are coming from the hosted agents and can easily distinguish them from the trades coming from the standard web site interface. This information would be of use to the marketplace, the traders and the agents. Knowing when agents perform well, on which markets and in what situations is a largely unexplored area. Unfortunately, there is no way to guarantee that the data is fully accurate, as mentioned in the section below: Human and Agent Tracking. Even without hosting agents, the marketplace could track the same basic information by discriminating between trades using the HTML and RPC interfaces, although it is easier for humans to impersonate agents in that scenario.

Finally, having agents hosted on the marketplace reduces the hosting cost to traders, thus implicitly lowering barrier to entry. Unfortunately, it also increases the barrier to entry in other ways. A hosted solution would typically require agents to be written in a particular programming language. Direct interaction with the server might require more sophisticated understanding of the market server internals. Hosting agents on traders' machines, is not even particularly expensive given that all traders would already need a networked computer to be on the market anyhow. Although their personal computer might experience more downtime or network loss, this doesn't keep them from entering the market, it just means they might miss a few opportunities here and there.

Hosted agents are dangerous; a poorly-written or malicious agent could take down the server. Additionally, Prediction Economies include more than one marketplace, so it would mean either hosting the agents on every available marketplace or having RPC communication anyway, as if they were hosted by the trader. Finally, the closeness of the agent to the programmer tends to lead to improvements in rapid prototyping that would be more difficult with remote hosting. In the end, the benefits of hosting did not seem to outweigh the costs at this stage, but this was far from a definitive result.

Human and Agent Tracking

We discussed the benefit of capturing and providing information about the origination of trades, in other words: which trades were executed by humans and which were executed by agents. Human traders might have added incentive to trade in markets dominated by agents, because they believe they have access to information that automated traders simply cannot access or comprehend. Alternatively, agents may benefit from knowing whether there are any humans trading. If they are designed to take advantage of common human trading patterns, the agents would want to stay out of markets without any humans.

It is incredibly difficult to reliably track the difference between agents and humans. If either wanted to obscure their identity, there are typically ways to do it. Say there was an agent that wanted to appear to be human. It could log in through the website as if it were human. A common preventative measure is to use a CAPTCHA[7]. A CAPTCHA is a question that is easy for humans to respond to, but hard for agents to respond to. A common implementation is to have a very warped word drawn as an image on-screen, which the user has to respond to by typing out the word. This is hard enough for computers to answer that it is a significant deterrent. However, in the marketplace environment, a human could respond to the CAPTCHA

7 Luis von Ahn, Manuel Blum and John Langford. Telling Humans and Computers Apart Automatically. *Communications of the ACM, Volume 47, Issue 2 (February 2004).*

on login, and then the agent could trade from there. So the web site would have to have a CAPTCHA on every trade page which is a significant inconvenience and cost in trade execution time. Barring this inconvenient design, it is fairly easy for an agent to impersonate a human.

Conversely, if a human wanted to impersonate an agent, one could write an agent that is actually just an interface for the human to trade on the back end. At the press of a button on the agent program, the agent would execute a trade that seems to be coming from an agent. We considered a "reverse CAPTCHA," but it turns out to be even easier to circumvent. A reverse CAPTCHA would be something that is easy for a computer to calculate quickly, but is tough for a human. The simple counter-strategy to this is to have the agent program calculate and respond quickly without waiting on the human. So it becomes a sort of hybrid interface, responding as an agent when necessary and allowing the human to make the actual trade decisions.

We were deterred from building any human versus agent tracking, but it would not be difficult to add in a few lines to the marketplace that tracks whether the request came in over RPC or HTML. Although industrious users might be able to spoof one side or the other as described above, a basic tracking mechanism might still prove useful, especially if the cost of spoofing is high enough that it can be assumed that few users are trying it.

Communication Alternatives

There are a wide variety of options available for remote communication between the components in a Prediction Economy. Many of them are slight variations on each other, so a general strategy was to stick with the simplest available option.

SOAP and REST are alternatives to RPC. SOAP originally stood for *Simple* Object Access

Protocol. Despite the name (which is now dropped), SOAP is the least simple of the alternatives. It is incredibly extensible, allowing designers to do everything they need and more. Object models, complicated structures, and message specific processing instructions are available, but they come at a cost. The many features and extensions dramatically increase the learning curve for understanding how to communicate with the marketplace server, and building a simple working prototype. This cost of simplicity would discourage traders from building agents, which was a major reason we chose RPC over SOAP.

REST is essentially the equivalent of having the agent use the human user interface (UI) for trading. This certainly has its advantages: the marketplace does not need to do any alterations, and all of the 'specification' for the interface is the HTML source. The downside is that the human UI is typically constantly changing. Changes in UI are much easier for humans to adapt to than for agents. If a single field on the HTML page changes names, it would typically require a change in the agent code. This is okay with one or two agents, but with an economy full of agents, every UI change becomes a significant public cost. Over the course of our own project, the UI went through two changes: one minor and one major. If we had been using REST, we would have been forced to implement changes to all our agents both times. Because the RPC interface was distinct, the agents kept on trading without interruption. Another disadvantage of REST is that you lose the most simple way to achieve human versus agent tracking, because the server cannot tell the difference between a human visiting the page and an agent.

Java Remote Method Invocation (RMI) was another option we considered. It was quickly thrown out, because it would limit the trader to use exclusively Java for programming the trading agents. Although both marketplace and agent were built on Java in this implementation of the

Prediction Economy, there is little reason to believe they will all be in the future.

Security Issues

There was little focus on the security in this design; this project was aimed at getting the first fully functional reference implementation. In a real-money market, system security would naturally be of paramount importance. The marketplace itself must of course be designed with secure accounts, but with a distributed system like this, there are many entry points of concern.

Notable areas of potential attack are: phishing, packet sniffing and trojan horses for gaining root access to the market manager or marketplace. This is far from an exhaustive list, but an obvious place to start. In a phishing attack, a malicious user impersonates either the marketplace or market manager in order to trick the user into sending the username and password to the malicious user. In packet sniffing, a sophisticated user could listen in to the Internet traffic going between the elements of the Prediction Economy to extract sensitive information, including passwords. Finally, a malicious user might be able to gain root access, say by tricking the operator of the market manager, the marketplace, or a trusted information broker into installing a trojan horse virus. In this case, or if a malicious user were able to otherwise impersonate any of these components, then he could wreak havoc by creating false markets, by closing markets prematurely or with the incorrect outcome, or by tricking agents traders into placing bad trades. There are well-understood safeguards that can help prevent these things, although they are typically not foolproof. These safeguards include using certificates, a web of trust, RSA encryption for all communication, and of course a recently-updated virus scanner.

Implementation

In order to understand how this high-level design was implemented, one must necessarily know the context of the markets being traded. The Prediction Economy was implemented with prediction markets offering claims on football plays during a recorded college football game. In these markets, traders take up opposing positions that pay off depending on if the offense attempted to pass or run. We did not run markets in plays that were ambiguous or had neither outcome.

These markets were particularly interesting for two reasons. First, they predict the intent of just a couple people: the quarterback and offensive coordinator. We are curious if standard prediction market theory extends to market topics based on information that is intentionally hidden and held by so few people. Second, the markets are interesting because they have very short lifespans: the market claim is opened and resolved on the order of minutes or less. The question of accuracy in markets of intent prediction remains open, but we do show that the implementation can handle these short-lifespan markets.[8]

Communication

The interaction between elements of a Prediction Economy are critical to its function. When the economy is first being built, as in our case, a single entity is likely to be managing every component except the agents. Agents should be able to come and go as they please, even being able to crash without disrupting the economy. So we force all agent communications to be initiated by the agent: it is required to contact the NewsServer and Zocalo in a standard client-server type model. If any other components were required to initiate communication with the

8 We did not test short-lifespan markets in real time, as the information broker was not built for live feeds.

agents, it would mean keeping a database of current, live agents in each component, which seems to be an unnecessary overhead. Additionally, all requests are kept as single atomic actions and internally transactional without any state between calls. This provides added simplicity and protection against faulty agents.

We use an RPC model accessible across platforms and languages called xml-rpc. This interface was chosen for a couple reasons. Xml-rpc is portable – it can be used in many different programming language contexts. It is well supported – the Apache group, which maintains xml-rpc written for Java[9], is respected for its thoroughness and innovation. So xml-rpc gives traders the necessary freedom to create agents however they want, and the reliability of knowing the library is consistent and well-built.

Marketplace: Zocalo

Zocalo is a fully functional marketplace server, so our requirements for implementation were minimal. However it did not have any interface for agent interaction. So we added in an interface[10] using xml-rpc that allows contact from both the NewsServer and the agents. The interface for the agents includes procedures to deposit cash, make trades and get the current market price. The interface for the NewsServer includes procedures to create and close markets. These interfaces are logically separated, but technically are part of the same RPC server.

Implemented Agents

The agents did not have to meet any specification besides being able to use xml-rpc to talk to the NewsServer and Zocalo. So all of the implementation details are different from one agent

9 Documentation, source and binaries available at: http://ws.apache.org/xmlrpc/

10 Thanks to Chris Hibbert for helping me incorporate the xml-rpc code with Zocalo.

to the next. The only limitations on the number of agents are the ability for the NewsServer to handle the requests and whatever limitations Zocalo has on the number of possible traders (neither of these upper bounds were tested). In the course of developing and testing the system, we developed several reference implementations of different agents. The notable ones include a rule-based agent and an artificial neural network (ANN) agent.

Both trading agents, the rule-based and the neural network agents, analyzed the details about the previous play and predicted the outcome of the next one. Using this prediction, they made a single bet of a fixed number of shares in the market, based on a parameter set at agent creation time. In testing, the agents were initialized with several seconds of delay between each other and with a slow refresh rate, so that their trading activity would be staggered regularly throughout the testing rather than having all of them trade immediately at the creation of the market.

The rule-based agent used a base prediction about the likelihood of a pass, and the distance to the first down to modify this base expectation in each play. The base pass likelihood was calculated using the 2007 NFL season from the combined statistics of the Washington Redskins and the New York Giants, which came to 54%. The rule-based agent used the following rule: if the first down was less than 5 yards away, it reduced the likelihood of a pass from 54 by a uniform random deviation from 0 to 10. Otherwise, the rule-based agent would increase the expected probability of a pass by the same random deviation. Then it would place a single trade. If the market is being run with an automated market maker (AMM), then the agent trades a predetermined number of shares towards its belief about the probability (buy if the market price is lower than the expected probability, and vice versa). If the market is being run without a market maker, it will place buy orders in a predetermined number of shares just below its belief about the expected probability, and place the same number of sell orders just above the expected probability.

The artificial neural network[11] (ANN) agent was built on top of the open source library Joone[12]. Joone provides all the underlying neural network structure and training code, which we then extended for the agents to take in football data and predict the likelihood of a pass. The agent was trained using data from the first quarter of a 2007 Redskins versus NYG game. It considered three inputs: which down the play is on, how many yards are left till the first down, and how many yards are left till the goal. It is trained so that the output of 1 means pass, and the output of 0 means run.

I tested the output of the neural network against the plays in the BCS college game between West Virginia University and Oklahoma University. One ANN agent was designed to make a trade in every market no matter its confidence. It traded in the direction of the neural network output (ie~ output >= 0.5 means pass and output < 0.5 means run). This agent was accurate in predicting the play call 62.5% of the time, across all 32 relevant plays. If instead, the agent only places trades when it is more confident, the accuracy goes up significantly. Using this scheme, the agent only enters the market if the neural network output is within an epsilon of 0 or 1. After testing against epsilon values of 0.3, 0.2, 0.1, 0.08, 0.06, 0.05, 0.03, and 0.01, I found that the agent made predictions at a maximum accuracy of 87.5% when using a value of 0.08. That agent only traded if the neural network output was less than 0.08 or more than 0.92, which was true in 8 of the 32 markets.

11 Stuart Jonathan Russell, Peter Norvig. Artificial Intelligence: A Modern Approach, Prentice Hall, 2003.
12 Joone project details and code can be found at: http://sourceforge.net/projects/joone/

The market should benefit from having several neural network agents with multiple epsilon settings in the market. High epsilon agents would add more liquidity with less accuracy, and low epsilon agents would add more accuracy with less liquidity.

Potential Agents

The possible strategies of agents are limitless. Many options were considered, and there is every reason to try as many of them as is feasible. The only limit was in our resources to produce them. In particular, I unfortunately did not get a chance to implement the following types of agents: a momentum trader, a technical trader, a support vector machine trader, and a long-shot anti-bias trader. These traders were briefly considered, but set aside for the completion of the Prediction Economy.

The momentum trader is a simple agent that watches for price changes and trades on top of them in the same direction. The rationale is that many players buy a little bit of prediction market "stock" at a time, slowly testing the stiffness of the market. So if the momentum trader detects a new pattern of increasing prices, it will buy as soon as possible, and sell when the momentum ends.

The technical trader can use all sorts of ways to identify patterns, but a particularly promising one is to use an entropy-based pattern matching system as described by Jenson et al[13]. It works to detect patterns on-the-fly and uses an entropy measure to keep track of how often that pattern has occurred in the past and whether it seems to be recurring. Using this information, the agent could keep track of common trading patterns and trade before it expects a jump (or drop) in

13 Jenson, Boley, Gini, Schrater. *Rapid on-line temporal sequence prediction by an adaptive agent* International Conference on Autonomous Agents, 2005

price. It would take into account the profitability of the pattern in addition to the entropy to help decide whether it is worth remembering and/or trading. This trader is essentially a super momentum trader in the sense that it would naturally learn momentum trades, but also other kinds of patterns. In fact, in smaller markets, it might even be able to adapt to the trading patterns of the individual traders. Those traders likely have exploitable trading patterns, which the technical trader can profit from.

A support vector machine (SVM) agent analyzes the play data using an SVM, which is a robust, accurate non-linear classifier. SVMs are much like artificial neural networks (ANN), but are less prone to over-fitting[14]. Where ANN algorithms often form local minima, the SVM has a global minimum. This more advanced technique should mean that where an ANN does something akin to remembering old plays, SVMs learn how to classify plays in a more general sense.

Finally, a long-shot anti-bias trader would look for markets in which there are extreme prices and trade in a way to make them even more extreme. Human traders have a documented long-shot bias[15]. This means that the market is distorted at extreme prices. As the market nears 100, it will tend to undershoot the 'true' probability because of the traders on the long-shot side of the trade. Conversely, a market with a price near 0 will be over-confident. So a long-shot anti-bias trader would buy very high percentage trades, and sell very low percentage trades. This simple technique is a bit dangerous: it is very expensive to be wrong when the prices are that extreme, and there is no way of knowing how many other traders are also trying to take

14 Nello Cristianini, John Shawe-Taylor. *An Introduction to Support Vector Machines,* Cambridge University Press, 2000.

15 M. Ali, *Probability and utility estimates for racetrack bettors,* Journal of Political Economy **85** (1977), pp. 803–815.

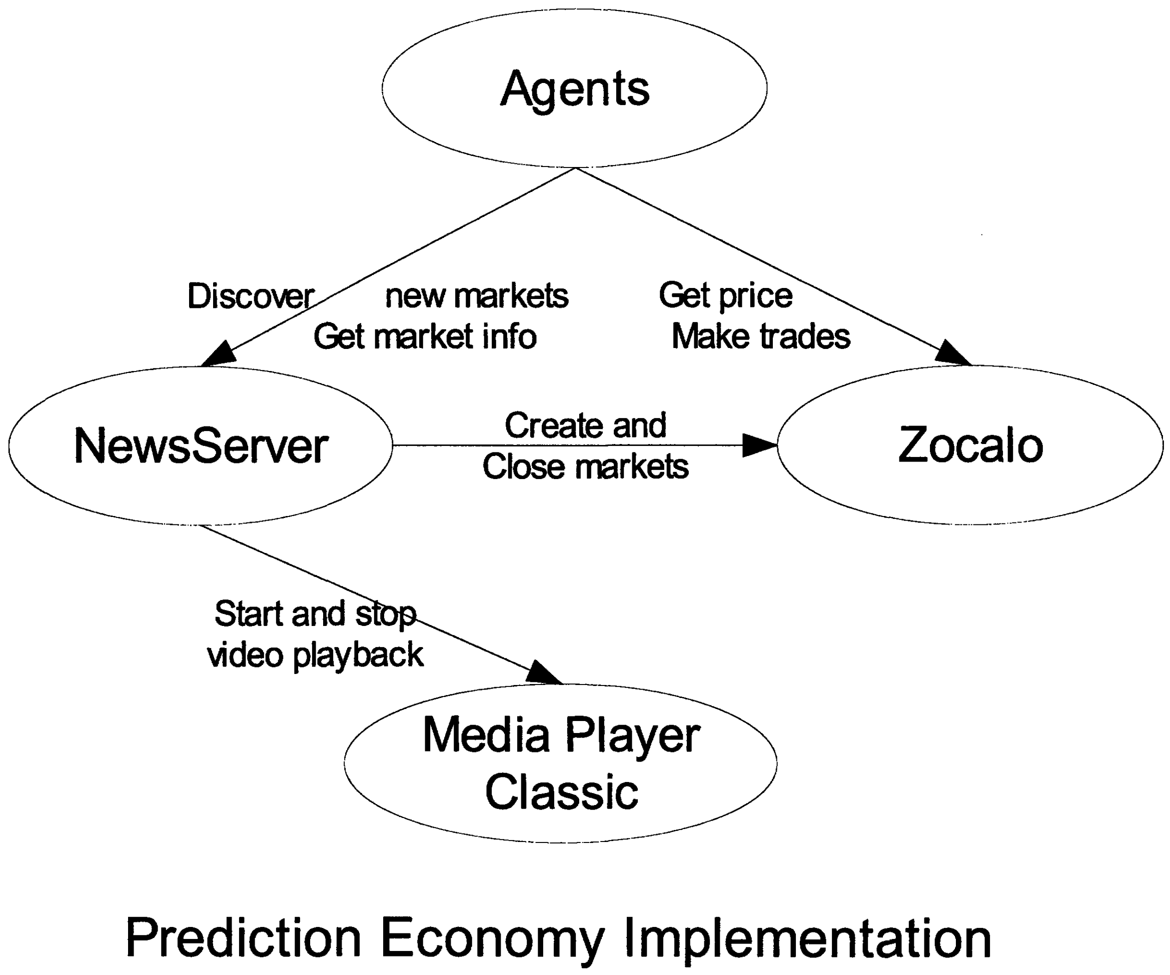
advantage of a long-shot bias. So the long-shot anti-bias trader would have to monitor the long-shot bias in many markets over time, and trade less aggressively if the bias appears to be reducing. The bias would naturally reduce as other traders adopt similar strategies.

Market Manager and Information Broker: NewsServer

The NewsServer had the dual distinct roles of market manager and information broker. In these roles, the NewsServer is in charge of opening and closing market claims and providing relevant semantic details to the agents, respectively. The market manager implementation is fairly straight-forward. It simply needs to know when new events start, when current events end, and how the event concluded. The only detail that is not discussed earlier in the design is exactly how the NewsServer gets access to the necessary information. Implicitly, it is like the other agents: it requires access to basic information from the information broker. Because the NewsServer integrates the roles of market manager and information broker, it has direct access to the necessary football play data. These data are then used to determine when and how to open and close markets.

In testing, the NewsServer also showed a video of a football game, acting as a kind of information broker to the human participants. The video gave humans the relevant and necessary football play information required to trade on the markets alongside the agents. In this fabricated setting, the NewsServer knew the outcome of events ahead of time, because the games were previously recorded and transcribed. The mechanism by which the NewsServer collects and parses this information will be domain-specific. The details of collection are likely to be uninteresting to all but those in that particular domain, so I will not detail the collection or parsing of football play information. The added role of the NewsServer as an information broker for human traders results in a slightly updated communication structure, shown in figure 2.

FIGURE 2 – Prediction Economy components as implemented. Zocalo and Media Player Classic[16] are available as open source packages; the NewsServer and the agents are newly built.



In our implementation, the NewsServer, agents and media player were all on the same computer, while Zocalo was hosted separately. However, all of the communication conduits represented as arrows in figure 2 work over TCP/IP. They can easily all be spread across four (or

16 Media Player Classic is a sub-project of: http://sourceforge.net/projects/guliverkli/

more) different computers.

Because Prediction Economies provide aggregated predictive judgments, they have little practical value if event outcomes are already known (as when replaying past events). Given that only future events are of interest, information brokers like the NewsServer must ultimately collect real-time information and report the semantically parsed data to the agents and marketplace. We largely sidestepped this issue because we were preparing to run experiments and wanted repeatable results. The NewsServer knew the whole series of events, but only gave out information about those events to the agents and market at the appropriate time.

The semantic system works by sharing an object model. The NewsServer defines a class that describes a current situation – in this case a football play. During specific plays, the NewsServer generates an object of that class and serializes it into a character string which is then transmitted upon agent request. The agent is compiled with access to the class, so it can un-serialize the data and interpret the situation, acting on the data however it desires. If a new event context is required, the NewsServer defines a new class, and the agent recompiles with access to the class so that it can properly un-serialize the new data sent by the NewsServer.

Lessons Learned

We learned many lessons in the implementation and testing of this system. This section takes a brief pause from describing system specifics to extrapolate the lessons learned to more general situations. There were numerous benefits from decentralization of the components, the Darwinian AI environment, and the new concepts of a market manager and information broker.

Decentralization

Decentralization was crucial for system stability. If the agents had been hosted on the server, or otherwise running in the same process as the other compenents, then a single failure in one of many agents could bring the whole system to a halt. As built, the agents can start up and exit on the fly without affecting the NewsServer or Zocalo. This is very natural if you consider the human analogy: if a trader's browser or computer crashes, it should not shut down the marketplace.

Another effect of decentralization is that the components can be separately maintained and run across the world. For example, if an agent or the NewsServer needs an update, there is no need to take down the marketplace. This allows for decentralized, decoupled development which means faster progress.

Finally, because an agent can simply be hosted on a home computer, someone with basic skills can experiment with running their own agent. They do not need to sign up or interact with an unfamiliar host. They can build one from scratch, or start even further ahead by downloading a superclass model that handles all of the major work for interacting with the NewsServer and Zocalo. This dramatically reduces barriers to entry.

Darwinian AI

By bringing together the benefits of decentralization and markets, we ended up with a simple way to rank and weed out AI designs. Simply pitting agents against each other and humans with a limited budget means that the agents that underperformed (or did not handle the capital risk well) eventually run out of money and are removed from the market. It does not take much analysis to determine that the removed AI were moving the market in the wrong direction, and the owner has a strong disincentive to reintroduce it to the market. Conversely, agents that do well have more capital to move the market . We can use the funds in an agent's account as a proxy measurement of its success. The take-away is that having natural selection processes built into your system is a convenient and effective way to test many alternatives.

Market Management

Managing the markets automatically seems to be a fairly new concept in the field of prediction markets. While it was convenient for the humans that would otherwise have to open and close markets, automation turned out to be even more important. It enabled a brand new kind of market that was previously infeasible. Automation was critical to run the high-tempo markets we dealt with, because the human cost and error rate would otherwise be prohibitive. Also, in the context of experiments where we needed to show media to participants with high precision, the market manager was crucial for automating the display process. Automated market management did not just make these features easier; it made them possible. Perhaps automation of other functions in a Prediction Economy would enable other new features that we have not yet dreamed of.

Information Broker

Understanding the information broker and how it interacts with other components is crucial to understand the Prediction Economy as a whole. The key role of the information broker is to provide semantically tagged information. If no brokers existed at all, the Prediction Economy would come to a halt due to information starvation. Understanding natural language is still a "hard" AI problem, although there are many techniques for it. So having at least one broker that accomplishes this task and passes on the semantic data to other agents greatly simplifies the task of both trading agents and the market manager. In addition, a well-built information broker should be able to parse and send new data quickly. This is very useful for agents, because a quick response to new information can be the difference between profit and loss. If there were not even a single information broker, the agents and market manager would be starved for information and be unable to perform even their most basic and important tasks. This seems to support the idea that information is the lifeblood of a Prediction Economy. Proper information capture and transfer makes the difference between a non-functioning economy, a poorly functioning economy, and an efficient economy.

Contributions

During the course of the project, I designed and demonstrated a working Prediction Economy with interacting of agents, markets, market managers, information brokers and human traders. I have demonstrated a working environment with more than 11 human traders and 6 agent traders interacting. I provided the necessary improvements to the marketplace software, Zocalo, to enable it to interact in the Prediction Economy. I designed and implemented ground-up versions of a market manager/information broker (the NewsServer), a rule-based trading agent, and two variations on an Artificial Neural Network-based agent. All of these are licensed as open-source, and are available for download[17]. Finally, we ran preliminary tests on this framework, getting promising results on successful agent strategies.

17 Search sourceforge.net for Prediction Economy, or contact jasoncarver@alum.mit.edu directly

Appendix A – Setup Instructions

NewsServer Setup

Confirm that the relevant NewsServer parameters are set up:

```
        public static final boolean manualAdvance = true; //must click to show next play, otherwise will only pause briefly
        public static final boolean agentOnlyTrading = false; //does not show video or wait long between rounds
        public static final int groupID = 333; //start with 3 next
        private static final int hours = 0;
        private static final int min = 31;//5;
        private static final int seconds = 50;//15;
        private static final String gameVideoFile = null; //for DVD's
//      private static final String gameVideoFile = "C:\\path\\to\\video\\file.mpg";
        private static final List<FootballPlay> defaultGame = FootballGameFactory.getWestVirginiaVsOklahoma2008();
        public static final boolean decoupleFromServer = false;
        public static final int serverPort = 8080;
```

ManualAdvance should be on for experimentation (it can be useful to turn off for agent-only trading). GroupID should be incremented every time: Zocalo requires a unique market name, so the groupID is appended to all market names. The time parameters reference the starting time of the football game. Feel free to set it to the middle of a game, it will just skip the markets from the part leading up to the specified time. The game video file should be set to null for DVD's (typically the only way to legally view media because of restrictive copyrights). The defaultGame specifies what transcribed data will be used by the NewsServer. If you want to test without interacting with the marketplace server, set decoupleFromServer to true. ServerPort is the port referenced by agents contacting the NewsServer.

Media Player Classic (MPC) Setup

Install MPC from mpc2kxp6490.zip at:
http://sourceforge.net/project/showfiles.php?group_id=82303&package_id=84358

Install the codecs from ffdshow_rev1803_20080120_xxl.exe at:
http://sourceforge.net/project/showfiles.php?group_id=173941

Open MPC and go to View -> Options
Go to Player->Formats and select Quicktime file
Go to Player->Web Interface and check "Listen on port:"
Fill in the port number: 13580

You should be able to double-click a .mov file and have it open in MPC now. Also, if you run MPC on the same computer as the NewsServer, it will automatically be detected, then the NewsServer will control it to display the transcribed football plays.

Note about MPC conflict: I installed some Microsoft additions for Windows (Silverlight, Direct

X, and various patches) just before an experiment, and the NewsServer could no longer connect to MPC. The patches appear to have made port 13579 unavailable for service. That port is the default server port for MPC. MPC did not complain about the port being taken, and the Java code was just saying connection refused. The system was returned to full functionality by changing the MPC server to listen on port 13580.

Preparing the Experiment

There is no automated tracking of agent performance, so write down the balance of agents (currently agent1 through agent6). Optionally re-fund the agents up to 10,000 Zocalo dollars each. Fund the NewsServer sufficiently. It needs funds to endow the automated market maker. For 32 plays, I fund as much as 10,000 per play, so it should have a balance of at least 320,000. Close any open markets on Zocalo to avoid confusion for new traders.

Running the Experiment

Running a game market is as simple as starting up all these processes in this order:

1. NewsServer
    1. Increment groupID
    2. run main()
2. AgentFactory
    1. Confirm relevant agent code in main()
    2. run main()
3. Media Player Classic
    1. Insert DVD
    2. Navigate to relevant Title/Chapter

The NewsServer has a UI with further instructions. Note that the NewsServer will exit on exceptions, the most common of which is that there was already a market on Zocalo with the same name (which usually means you forgot to increment the group ID). Look for error messages in the Eclipse console to diagnose the problem.

Appendix B – Code Excerpts

Several relevant elements of the code are included below. For complete and up-to-date code, look for Prediction Economy on Sourceforge.net.

NewsServer – Excerpts

```
package edu.mit.cci.predecon.news.server;

//imports...

/**
 * NewsServer
 * @author Jason
 *
 * The NewsServer serves two important roles: it is the market manager,
 * opening and closing markets at the appropriate time. Also, it is an
 * information broker, providing relevant semantically parsed data to
 * trading agents who request it. In this case, it also serves an unusual
 * role: displaying video of the relevant event to human traders.
 */
public class NewsServer {

	public static final boolean manualAdvance = true; //must click to show next play, otherwise will only pause briefly
	public static final boolean agentOnlyTrading = false; //does not show video or wait long between rounds
	public static final int groupID = 333; //start with 3 next
	private static final int hours = 0;
	private static final int min = 31;//5;
	private static final int seconds = 50;//15;
	private static final String gameVideoFile = null; //for DVD's
//	private static final String gameVideoFile = "C:\\path\\to\\video\\file.mpg";
//	private static final String gameVideoFile = "C:\\Torrents\\NFL-
New_York_Giants_vs_Washington_Redskins-20071216\\Giants_vs_Redskins-20071216--a.mpg";
//	private static final String gameVideoFile = "C:\\Documents and
Settings\\Jason\\Desktop\\Giants_vs_Redskins-1stQtr.mov";
	private static final List<FootballPlay> defaultGame = FootballGameFactory.getWestVirginiaVsOklahoma2008();
	public static final boolean decoupleFromServer = false;
	public static final int serverPort = 8080;

	/**
	 * @param args
	 */
	public static void main(String[] args) {

		//RPC Server
		final NewsServer me = new NewsServer();
		try {
			me.run();
			singleton = me;
		} catch (XmlRpcException e) {
			e.printStackTrace();
			return;
		} catch (IOException e) {
			e.printStackTrace();
			return;
		}
```

```
            //Shutdown hook
          Runtime.getRuntime().addShutdownHook(new Thread() {
        public void run() {
          me.shutdown();
        }
      });

                  //UI
                  NewsUI ui = new NewsGUI();
                  me.setUI(ui);
                  ui.setType("Football");
                  ui.show();

                  //Choose Game / Group (statically)
                  List<FootballPlay> game = defaultGame;

                  boolean success = true;

                  //Show video
                  if(agentOnlyTrading){
                          me.pauseWithMessage("Press OK to begin the AGENT ONLY experiment at
"+hours+":"+min+":"+seconds+".",
                                          true);
                          Util.silentSleep(2*1000);
                  } else if(gameVideoFile == null){
                          ui.notifyUser("Start up Media Player Classic and open the relevant DVD.\nThen press OK to begin the
experiment at "+hours+":"+min+":"+seconds+".",true);

                          //skip to appropriate section
                          if(!MediaPlayerClient.goTo(hours, min, seconds)){
                                  success = false;
                                  ui.notifyUser("failed to contact media player to seek to time",true);
                          }

                  } else {
                          Util.openFileWithDefaultApplication(gameVideoFile);

                          //allow Media Player Classic time to initialize, keep trying if it isn't yet ready
                          boolean mpcLoaded = false;
                          while(!mpcLoaded){
                                  Util.silentSleep(2*1000);
                                  mpcLoaded = me.pauseWithMessage("Press OK to begin the experiment at
"+hours+":"+min+":"+seconds+".",
                                                  true);
                          }

                          //skip to appropriate section
                          if(!MediaPlayerClient.goTo(hours, min, seconds)){
                                  success = false;
                                  ui.notifyUser("failed to contact media player to seek to time");
                          }
                  }

                  //Start timer loop
                  if(success)
                          me.runExperiment(game);

                  Util.out("Shutting Down News Server");
                  System.exit(0);
          }

          private void runExperiment(List<FootballPlay> game){
                  lastMarket = "";
                  int lastPlayEnd = Util.hmstosec(hours, min, seconds);

                  for (int i = 0; i < game.size(); i++) {
```

```
FootballPlay play = game.get(i);

//skip past this play if the start time is past the event
int nextPlayEnd = play.getSecondsIntoVideo() - 6;
if (nextPlayEnd - lastPlayEnd <= 0)
    continue;

//open market BEFORE showing the play in video
String marketName = null;
if(i < game.size() - 1){
    nextPlay = game.get(i+1);

    if(nextPlay.isTradeable()){

        //prepare to create market on Zocalo
        marketName = generateMarketName(play);
        marketCurrentlyOpen = true;
        lastMarket = marketName;

        //create market on Zocalo
        if(!createMarket(marketName, generateMarketDescription(play))){
            Util.out("Failed to create market "+marketName);
            marketCurrentlyOpen = false;
            break;
        }

        //after this statement, polling agents would expect the market to exist on
Zocalo
        currentPlay = play;
    }

    //show next play outcome (if tradeable, it will be starred out)
    ui.showPlayPreview(nextPlay);
}

//show last play outcome locally on NewsServer
ui.showPlay(play);

//show video
if(!waitBetween(lastPlayEnd, nextPlayEnd))
    continue;

//pause for market trading
if(i < game.size() - 1){

    if(nextPlay.isTradeable()){

        //pause for trading
        pauseForTrading(false);

        //reset video to proper time
        int plays = Util.sectos(play.getSecondsIntoVideo());
        int playm = Util.sectom(play.getSecondsIntoVideo());
        int playh = Util.sectoh(play.getSecondsIntoVideo());
        MediaPlayerClient.goTo(playh, playm, plays);

        //close old market
        marketCurrentlyOpen = !closeMarket(marketName, nextPlay);
    } else {
        ui.notifyUser("Next play has no market",6000,false);

        currentPlay = null;
    }
} else {

    MediaPlayerClient.sendToMediaPlayer(MediaPlayerCommand.Pause);
```

```
                                        ui.notifyUser("The experiment is now over.",true);
                                        currentPlay = null;
                                }

                                //prepare for next loop
                                lastPlayEnd = nextPlayEnd;
                        }
                }
}
```

Artificial Neural Net Agent

```
package edu.mit.cci.predecon.agents;

//imports...

/**
 * NeuralNetAgent
 * @author Jason
 *
 * Estimates the likelihood of a pass in a football play by training
 * an Artificial Neural Network
 */
public class NeuralNetAgent extends TradingAgents {

        private static final int numInputs = 3;
        private static final int numHidden = (int)Math.ceil(numInputs*1.5);
        private final int tailSize; //how close to 100% (a NN output of 1) does the agent have to be to participate?
        private final int selfConfidence; //if sure, how high will push price?

        private NeuralNet nn;
        private final int buyNumShares;

        public NeuralNetAgent(String username, int tailSize, int selfConfidence) {
                super(username);
                nn = generateNN(numInputs,numHidden,1);
                this.tailSize = tailSize;
                this.selfConfidence = selfConfidence;
                buyNumShares = (selfConfidence - 50)*100;
        }

        private NeuralNet generateNN(int inputCount, int hiddenCount, int outputCount){
//                       Line 1: Create an MLP network with 3 layers [2,2,1 nodes] with a logistic output layer
                NeuralNet nnet = JooneTools.create_standard(new int[]{inputCount,hiddenCount,outputCount},
JooneTools.LOGISTIC);


                double[][] input = getTrainingData();

                double[][] desired = getTrainingResults();

//                       Line 2: Train the network for 5000 epochs, or until the rmse < 0.01
                double rmse = JooneTools.train(nnet, input, desired,
                                6000,     // Max epoch
                                0.01,     // Min RMSE
                                0,        // Epochs between ouput reports
                                null,     // Std Output
                                false     // Asynchronous mode
                );

                Util.out("NN agent: "+username+" trained until the rmse was: "+rmse);

                return nnet;
        }
        private double estimateNewProbability(FootballPlay play){
//                      Line 3: Interrogate the network
                double[] output = JooneTools.interrogate(nn, getNetworkInput(play));

                for (int i = 0; i < output.length; i++) {
                        Util.out("NN "+username+" predicts: "+output[i]);
                }
                return output[0];
        }

        @Override
        protected void gotNewMarket(String newMarket, FootballPlay play) {

                //generate prediction for next play
                double myEstimate = estimateNewProbability(play)*100;

                if(myEstimate < tailSize || myEstimate > 100-tailSize){
                        Util.out("NN agent: "+username+" is extremly confident about play: "+play);
                }
//              return here if you only want to trade high-confidence plays
                else
                        return ;

                //get price
                double price = getMarketPrice(newMarket, "pass");
                if(price == -1){
                        //place straddling bets on the play
                        int spreadAroundEstimate = 5;
                        placeMarketTrade(newMarket, "buy", "pass", (int)Math.round(myEstimate -
spreadAroundEstimate),buyNumShares);
                        placeMarketTrade(newMarket, "sell", "pass", (int)Math.round(myEstimate +
spreadAroundEstimate),buyNumShares);
                }
                else {
                        //this only works for the market maker
                        //buy up to / sell down to desired price
                        int bidToProbability = getHighConfidenceProbability(myEstimate);

                        String tradeType = "sell";
                        if(price < bidToProbability)
                                tradeType = "buy";

                        Util.out("Using info: "+play.toString()+" and price "+price+" to conclude "+tradeType
                                        +" of shares at price "+bidToProbability);

                        placeMarketTrade(newMarket, tradeType, "pass", bidToProbability, buyNumShares);
                }
        }

}
```

Rule-based Agent

```
package edu.mit.cci.predecon.agents;

//imports...

/**
 * Dummy Agent
 * @author Jason
 *
 * Also referred to as Rule-based agent, it makes a very simple guess about football plays.
 */
```

```
public class DummyAgent extends TradingAgents {

	private static final double passBaseline = 54.0;
	private double expectPass;
	private static final int numSharesToBuy = 1000;

	public DummyAgent(String username) {
		super(username);
		expectPass = passBaseline + (Math.random()-0.5)*20;
	}

	@Override
	protected void gotNewPlayData(FootballPlay newPlay) {
		// TODO Auto-generated method stub

	}

	protected void gotNewMarket(String newMarket, FootballPlay play) {

		double myEstimate = expectPass;
		double deviation = Math.random()*10;

		int ytf = play.getYards();
		if(ytf == -1){
			Util.out("yards to 1st was unusable, agent aborting trade");
			return ;
		}
		else if(ytf <= 5)
			myEstimate -= deviation;
		else
			myEstimate += deviation;

		//get price
		double price = getMarketPrice(newMarket, "pass");
		if(price == -1){
			//intended for markets with no Market Maker
			int spreadAroundEstimate = 10 - (int)deviation;
			placeMarketTrade(newMarket, "buy", "pass", (int)Math.round(myEstimate -
spreadAroundEstimate),numSharesToBuy);
			placeMarketTrade(newMarket, "sell", "pass", (int)Math.round(myEstimate +
spreadAroundEstimate),numSharesToBuy);
		}
		else {
			//this only works for the market maker
			//buy up to / sell down to desired price
			String tradeType = "sell";
			if(price < Math.round(myEstimate))
				tradeType = "buy";

			Util.out("Using info: "+ytf+" ytf and price "+price+" to conclude "+tradeType
					+" of shares at price "+(int)Math.round(myEstimate)+" after baseline
"+expectPass);

			placeMarketTrade(newMarket, tradeType, "pass", (int)Math.round(myEstimate),numSharesToBuy);
		}

	}

}
```

Agent Factory – generate all experimental agents to trade

```
package edu.mit.cci.predecon.agents;

import edu.mit.cci.predecon.util.Util;
```

```
/**
 * AgentFactory
 * @author Jason
 *
 * Convenient way to start up many desired agents in tandem
 */
public class AgentFactory {
	public static void spawnDummy(String name, long delay){
		DummyAgent agent = new DummyAgent(name);
		Util.executeAsync(agent);
		Util.silentSleep(delay);
	}
	public static void spawnDummies(String usernameBase, int startingFrom, int numDummies){
		for(int i = startingFrom-1; i < numDummies+startingFrom-1; i++){
			spawnDummy(usernameBase+(i+1), TradingAgents.pollingRate / numDummies);
		}
	}
	public static void spawnNN(String name, long delay, int tailSize, int selfConfidence){
		TradingAgents agent = new NeuralNetAgent(name,tailSize,selfConfidence);
		Util.executeAsync(agent);
		Util.silentSleep(delay);
	}

	public static void main(String ... args){
		//this would be a good place to have some kind of agent UI manager
		//	that tracks running agents, their performance, balance, etc
		int numAgents = 4;
		spawnNN("agent1",TradingAgents.pollingRate / numAgents, 10, 80);
		spawnDummy("agent4",TradingAgents.pollingRate / numAgents);
		spawnNN("agent2",TradingAgents.pollingRate / numAgents, 40, 57);
		spawnDummy("agent5",TradingAgents.pollingRate / numAgents);
//		spawnDummies("agent", 4, 3);
	}
}
```

FootballGameFactory – Transcribed Football Game Data

```
package edu.mit.cci.predecon.news.domain;

//imports...

/**
 * FootballGameFactory
 * @author Jason
 *
 * This class stores the transcribed data for football plays. This data
 * is currently used by the NewsServer to inform agents of current game
 * state, and to open and close markets properly.
 */
public class FootballGameFactory {
	//Next step: have this return a legitimate FootballPlaySeries object.
	//	It would make some things much easier and cleaner, like a whole-game time offset

	//suggested start time: 7:34
	private static LinkedList<FootballPlay> WestVirginiaVsOklahoma2008Plays;
  public static List<FootballPlay> getWestVirginiaVsOklahoma2008(){
	if(WestVirginiaVsOklahoma2008Plays != null)
		return WestVirginiaVsOklahoma2008Plays;
	LinkedList<FootballPlay> plays = new LinkedList<FootballPlay>();

	plays.add(new FootballPlay(WVvOK,Kickoff,
	 OK, 1, 10, yards, 80, 0, 0, Util.hmstosec(0,8,25))); // Play 1
    plays.add(new FootballPlay(WVvOK, Run,
        OK, 2, 5, yards, 75, 0, 0, Util.hmstosec(0,9,0))); // Play 2
    plays.add(new FootballPlay(WVvOK, Run,
        OK, 3, 1, yards, 71, 0, 0, Util.hmstosec(0,9,44))); // Play 3
    plays.add(new FootballPlay(WVvOK, Run,
        OK, 1, 10, yards, 70, 0, 0, Util.hmstosec(0,11,1))); // Play 4
    plays.add(new FootballPlay(WVvOK, PassComplete,
        OK, 2, 4, yards, 64, 0, 0, Util.hmstosec(0,11,43))); // Play 5
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        OK, 3, 10, yards, 70, 0, 0, Util.hmstosec(0,12,30))); // Play 6
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        OK, 4, 19, yards, 79, 0, 0, Util.hmstosec(0,13,36))); // Play 7
    plays.add(new FootballPlay(WVvOK, Punt,
        WVU, 1, 10, yards, 71, 0, 0, Util.hmstosec(0,14,12))); // Play 8
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 2, 5, yards, 66, 0, 0, Util.hmstosec(0,14,42))); // Play 9
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 1, 10, yards, 46, 0, 0, Util.hmstosec(0,15,28))); // Play 10
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 2, 5, yards, 41, 0, 0, Util.hmstosec(0,16,7))); // Play 11
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 1, 10, yards, 34, 0, 0, Util.hmstosec(0,16,42))); // Play 12
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        WVU, 2, 10, yards, 34, 0, 0, Util.hmstosec(0,17,20))); // Play 13
    plays.add(new FootballPlay(WVvOK, PassComplete,
        WVU, 3, 8, yards, 32, 0, 0, Util.hmstosec(0,18,0))); // Play 14
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        WVU, 4, 8, yards, 32, 0, 0, Util.hmstosec(0,19,29))); // Play 15
    plays.add(new FootballPlay(WVvOK, FieldGoal,
        OK, 1, 10, yards, 67, 0, 0, Util.hmstosec(0,19,56))); // Play 16
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        OK, 2, 10, yards, 67, 0, 0, Util.hmstosec(0,20,31))); // Play 17
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        OK, 3, 10, yards, 67, 0, 0, Util.hmstosec(0,21,15))); // Play 18
    plays.add(new FootballPlay(WVvOK, PassComplete,
        OK, 4, 13, yards, 30, 0, 0, Util.hmstosec(0,22,4))); // Play 19
    plays.add(new FootballPlay(WVvOK, Punt,
        WVU, 1, 10, yards, 64, 0, 0, Util.hmstosec(0,22,16))); // Play 20
    plays.add(new FootballPlay(WVvOK, OffPenalty,
        WVU, 1, 10, yards, 22, 0, 0, Util.hmstosec(0,24,44))); // Play 21
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 2, 7, yards, 19, 0, 0, Util.hmstosec(0,25,14))); // Play 22
    plays.add(new FootballPlay(WVvOK, Run,
        WVU, 3, 9, yards, 21, 0, 0, Util.hmstosec(0,25,54))); // Play 23
    plays.add(new FootballPlay(WVvOK, PassIncomplete,
        WVU, 4, 9, yards, 21, 0, 0, Util.hmstosec(0,27,42))); // Play 24
    plays.add(new FootballPlay(WVvOK, FieldGoal,
        WVU, 4, 9, yards, 21, 3, 0, Util.hmstosec(0,28,35))); // Play 25
    plays.add(new FootballPlay(WVvOK, Punt,
        OK, 1, 1, goal, 9, 3, 0, Util.hmstosec(0,29,40))); // Play 26
    plays.add(new FootballPlay(WVvOK, Run,
        OK, 2, 1, goal, 9, 3, 0, Util.hmstosec(0,30,15))); // Play 27
    plays.add(new FootballPlay(WVvOK, Run,
        OK, 3, 1, goal, 11, 3, 0, Util.hmstosec(0,30,38))); // Play 28
    plays.add(new FootballPlay(WVvOK, OffPenalty,

// et cetera...

        WestVirginiaVsOklahoma2008Plays = plays;
        return WestVirginiaVsOklahoma2008Plays;
  }
}
```