CHAPTER 3

Analogies between Systematic Trading and Robotics

This chapter introduces the central concept that underpins the book, namely the direct analogy between robotics and systematic automated trading strategies. This approach proves fruitful for an efficient understanding of the field, and opens the door to a much wider scope of research and development in finance that is naturally suggested by progress in the fields of complexity, self-organization, artificial life, and artificial intelligence. Part Two of the book concentrates on that bridge and suggests some future avenues.

3.1 MODELS AND ROBOTS

Similar to robots thrown into the real world, trading strategies need to survive the complexity of real markets. It is exactly the success of modern developments in robotics and artificial life that have inspired me to apply a variety of such techniques to systematic trading. In this chapter, the groundwork is set for the approach to building trading strategies that exhibit the features of adaptive autonomous agents (AAA). Part One of the book focuses on the autonomous feature and Part Two on the adaptive feature.

An AAA is a physical or software decision-making process that is composed of the following three elements.

1. *Sensors:* Any device that receives information from the external world. Robot: cameras, microphones, positioning devices, speed devices, etc.

Trading strategy: various indicators as well as performance measures of a range of simulated strategies.

2. *Actuators:* Any device by which the agent outputs information and acts on the external world. Robot: wheels, arms, guns, etc. Trading strategy: order management system that ensures current desired market position and emits current desired passive or aggressive orders
3. *Adaptive Control System:* A goal-oriented decision-making system that reads sensors and activates actuators. Robot: feedback and subsumption architecture that achieves optimal foraging behavior under constraint of power utilization and minimal damage to machinery. Trading strategy: feedback and subsumption architecture that achieves optimal profit under constraints of capital utilization and minimal drawdown.

In the above list, *device* is used instead of *mechanism* to draw a further and deeeper analogy with living organisms, which are the most sophisticated AAAs known.

The control system is of course of central importance: It is the brain of the AAA that achieves the required convergence from the current to the desired state. It is a goal-oriented system, in the sense that it has a final task in mind. It tries to achieve that task by balancing between short-term setbacks and long-term rewards, and closes the feedback loop between the real world and the *internal world* of the AAA. By *internal world* is meant the implicit or explicit representation of the real world that the AAA achieves via the input of sensor data into its control system.

3.2 THE TRADING ROBOT

Let us set the stage by introducing a direct parallel between a robot and a trading strategy by describing the flow of information from an observation to an action.

First of all, the trading robot has *sensors* that observe ticks, prices, bars, or any other compressed or uncompressed market event data. It then assembles this data into a representation that is usually a set of *indicators* that are computed via a *preprocessor*.

The indicators act as the first semantic layer of the robot—they filter and pre-interpret the data that is hitting the raw sensors. They can be, as a simple example, moving averages of the data observed. They are passed, along with the current state of the robot, to the *control system* that is its decision-making mechanism.

The control system makes the decision as to the position and orders to have in the market. The decison is enacted into the outside world by *actuators* that are typically a set of order management systems (OMS) that interface between the trading robot and the electronic commerce networks (ECNs) such as exchanges, dark pools, over-the-counter electronic markets, and so on.

The OMS sends the trade orders into the ECNs and manages the outputs from the ECNs. Once the trade is done, the OMS feeds that information back into the decision-making system of the robot.

Finally a *postprocessor* mechanism gathers the relevant data to compute the new state of the robot and makes it ready to observe the next event.

Figure 3.1 illustrates the information flow.

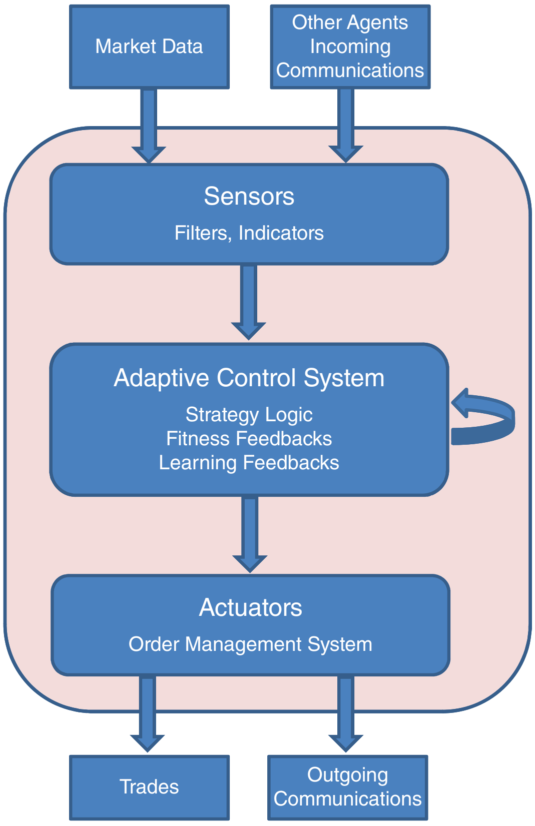

FIGURE 3.1 Block Diagram of Trading Strategy as an AAA

3.3 FINITE-STATE-MACHINE REPRESENTATION OF THE CONTROL SYSTEM

As per the discussion above, once the incoming data has been preprocessed and the relevant indicators calculated, the onus of the decision making falls onto the control system. Once that decision is made, an order is (or is not) generated and passed to the actuator that manages the interface with the external world.

What are the desirable features of a control system? Of course, the first and foremost is its ability to make money! Besides that, the principal feature is *completeness*, meaning a clear mechanism that enables the control system at every point in time to know what the current step should be. This, in itself, helps ensure *recoverability* from faults and hence reduces *operational brittleness*.

Assume the following real-world situation, where an automated trading strategy is connected to an ECN but suddenly the connection is lost for several minutes due to external unforseen circumstances. The strategy is short term, so every minute of data may be relevant to its successful operation. When the connection comes back, what state is the strategy in? Should it be buying? Selling? The control system should be built on the principle that whenever a gap in time or data occurs, it should have a defined plan to proceed with. The same actually holds for the design of the OMS and is discussed in Part Four.

Built-in completeness is key and should be the *central* design pattern for an AAA. Once a structure for completeness has been defined, the AAA stands on a solid basis, and then efforts can be concentrated on the quest for profitability.

This section focuses on the efficient representation of the control system by way of a finite-state machine that ensures completeness of its decision-making process.

Definition 1. Let S be a finite set of symbols representing abstract states, E a possibly infinite set of outside observable events. A finite-state machine $FSM(S, F)$ is defined by its complete set of transitions $F(s_{in}, s_{out}, e)$ with $s_{in}, s_{out} \in S$. The transition functions are such that $\forall e \in E$ and $\forall s_{in} \in S$ and there exists only one s_{out} such that $F(s_{in}, s_{out}, e) = TRUE$ and $F(s_{in}, s, e) = FALSE, \forall s \neq s_{out}$. The FSM, at any point in time, has a current state s_{in}. When a new event e arrives it changes to a potentially different state s_{out}, and that change is completely unambiguous, meaning that only one transition function is true and all others false. The set of transition functions can be represented by an $N * N$-matrix where $N = Card(S)$ is the number of states.

To illustrate how an FSM representation is implemented for a trading strategy, a simple example is discussed here. A set of more complicated real-world trading strategies is presented in the next chapter.

Consider a simple trend-following trading strategy that is always in the market. The strategy is long 1 unit when the price is above a simple moving average of length L, and short when it is below.

Initially, before any position is opened, at least L price update events $P(e_i)$ need to be received to calculate the simple moving average

$$SMA(L, i) = \frac{1}{L}\sum_{j=1}^{L} P(e_{i-j})$$

During that initial time the position of the strategy is zero.

One needs to preprocess the incoming data before presenting it to the FSM. Let $E = \{e\}$ the set of incoming market update events received since inception and $\Pi = \{P(e)\}$ the set of calculated prices. Here the price calculation function $P(e)$ returns the relevant price update depending on the nature of the event received. For example if e is an order book update that does not change the best bid and the best offer then it would be ignored, and the price calculation function would not return a value. On the other hand, trades or changes in the mid-price would be processed and added to Π.

Hence, a necessary and sufficient set of states for this strategy is

$$S = \{INIT, LONG, SHORT\}$$

Define the indicators *COUNTER* and *MA* that are calculated on each new addition P to the set of price updates Π as follows:

$$COUNTER = Card(\Pi)$$
$$MA = SMA(L, COUNTER - 1) \textit{ when } COUNTER > L$$

When an event e is received that yields a price update P, the indicators are recalculated—this is the function of the preprocessor explained above. They are then passed to the strategy's FSM as parameters. The FSM's transitions are explained here and the corresponding matrix representation is shown in Figure 3.2. In that matrix, the initial states are in the columns and the final states are in the rows. For clarity, this particular style of representing the FSMs has been adopted throughout.

The FSM starts in the *INIT* state. It continues in that state until L price updates are gathered. At the next price update P the *COUNTER*

$$
\begin{gathered}
F(INIT, INIT, P) = (COUNTER <= L)\\
F(INIT, LONG, P) = (AND(COUNTER > L)(P > MA))\\
F(INIT, SHORT, P) = (AND(COUNTER > L)(P <= MA))\\
F(LONG, INIT, P) = NIL\\
F(LONG, LONG, P) = (P > MA)\\
F(LONG, SHORT, P) = (P <= MA)\\
F(SHORT, INIT, P) = NIL\\
F(SHORT, LONG, P) = (P > MA)\\
F(SHORT, SHORT, P) = (P <= MA)
\end{gathered}
$$

FIGURE 3.2 FSM Matrix for the Moving-Average Model

indicator becomes $L + 1$. The preprocessor calculates the MA on the previous L price updates (not including this one) and passes the result to the transition matrix. Only the $F(INIT, *, P)$ row is considered by the processor and it consecutively starts computing the three Boolean functions.

The first one, $F(INIT, INIT, P)$ is false because $COUNT > L$. Hence either $F(INIT, LONG, P) = TRUE$ and $F(INIT, SHORT, P) = FALSE$ or vice versa. Notice the complementarity in the strict and nonstrict inequalities that ensure a nonoverlapping partition of the real line $R = \{P > MA\} \cup \{P <= MA\}$. This, in plain English, means that there is no chance for the FSM to fall through the cracks and find itself in an undefined state. This also means that at each price update there is no possibility for the FSM to simultaneously want to transition to two or more different states. There is one and only one transition possible (that of course may be to the same current state). Hence, depending on P, the FSM transitions to either the $LONG$ or the $SHORT$ state. Assume here that the strategy transitions to the $LONG$ state.

At this point, the final job of the control system is to emit a signal to the actuator to execute the buy trade. The OMS takes control and sends an order to the ECN. Once the trade execution confirmation comes back from the ECN, the OMS passes back control to the sensor, which is allowed to open its eye again for the next price update.

The actual details of the process of interacting with the ECN are far more involved and are covered in a later part. The OMS has to embed a finite-state machine of its own design to deal with complications arising from situations of partial fills, disconnects, and the like. Here and in Part Two the idea is to focus on the strategy's core decision making *assuming* that all trade executions are performed without such complications.

When that next price update comes, the process is restarted and the strategy will either remain $LONG$ or switch to the $SHORT$ state. If that is the case, the OMS will have to execute a sale of 2 units.

There are at least two recovery mechanisms possible in the case of a disconnect or any other operational problem that keeps the strategy off-line for a while:

1. The preprocessor takes the next available price P and computes the $COUNTER$ and MA indicators as if nothing happened, thus ignoring the gap.
2. The preprocessor queries a historic external database to fill as much missing data possible, thus repopulating the E and Π sets. Once done, it would use them and the next available P to compute the indicators.

Whichever way the recovery from fault is handled by the preprocessor, the FSM just takes its input and performs its decision-making work. In sum, the FSM embeds the core logic behind the trading strategy and disentangles it from any upstream and downstream operational issues. The FSM is the main element to focus on when researching and designing potentially profitable strategies.

Seeing trading strategies in the AAA context makes the representation of their brains as finite-state machines all the more natural. It puts them into a framework where a direct analogy with robotics can be exploited. It also emphasizes the event-driven nature of trading strategies as opposed to calendar-time driven.

In Part One various simple strategies will be exhibited via their FSMs. The control mechanism there is simply a position size decision-making matrix, based on the current state and the relationship between an indicator and the current price. The sensors observe the current price and the indicators are computed on a history of observed prices and other data. The actuators are not explicit; they will be covered in Part Four when an order management system and its connectivity to the electronic marketplace are described.

The representation of the strategy's core logic by way of a finite-state machine fosters a necessary degree of *discipline* for the design process. The goal is not to complicate that process but to ensure the strategy's stability and recoverability. These essential features are key for reducing the operational and management costs over time, let alone market-related losses. The less time spent on disentangling recovery problems, the more time spent on research and development of profitable strategies.

In order to put the above theory into practice, the next chapter jumps straight into a programmatic implementation of agent-based trading strategies.