STOCK TRADING LANGUAGE

PROGRAMMING LANGUAGES AND TRANSLATORS

PROF. STEPHEN EDWARDS

February 7, 2007

by

Rui Hu (rh2333)

Tom Lippincott (tml2115)

Rekha Duthulur (rd2241)

Matt Chang (yc2345)

Nikhil Jhawar (njj2104)

Problem Statement

There are two types of stock investors who participate in electronic stock trading:-

1. Institutional investors and
2. Individual investors (Retail)

Institutional investors are the big financial firms with their very own IT department which provides customized, proprietary solutions to design and implement their trading strategies based on certain algorithms using general programming languages like C/C++, Java, Perl or Python. However, individual investors do not have the luxury to hire professional programmers neither they have the technical prowess to create programs themselves. They normally have to log on to some trading websites and use the interface that a particular stock trading website offers to conduct their trading. For one, it is very inflexible and inefficient since little automation can be done this way. (Imagine sitting in front of a monitor all day and watch every price movement). Secondly, even if the website provides an array of choices for the investor to choose, the brokerage rate becomes simply unbearable for the small time investor to profitably trade.

Solution: The Stock Trading Language

Our goal is to develop a language for those individual investors involved in stock trading. The functionality of the language specific to the stock market domain arises from the need to aid the retail investor by means of simplicity. Simplicity means that every investor like a small time trader, a professor or a chauffeur who have no idea about what a piece of code looks like, can create his/her own customized piece of program to help him/her trade in a much more efficient and automated way. By programming we do not mean C or Java, but what we mean is creating a program using all but simple English!

If we step back and look at the evolution of programming languages and compilers. Machine level programming was the most general one, but it was always so hard to write and understand, also it belonged only to the domain of the computer scientists. Compiler's added a layer of abstraction so we could program using general languages instead of machine level programming; but the general languages shared some of the same problems that machine code had: it would get very complex for big software systems and hence was again restricted to the new domain of programmers.

Now, there is a new concept called "intentional programming", which is a collection of concepts that enables the software's source code to reflect the precise information, called *intention*, which programmers had in mind when conceiving their work. By closely matching the level of abstraction at which the programmer was thinking, browsing and maintaining computer programs becomes easier[1].

The basic idea that we derive from the concept of intentional programming is to build a of "code generator", so programmers and business domain experts can describe their

[1] Source: http://en.wikipdeia.org. The pioneer of the concept of Intentional Programming is Charles Simonyi, Microsoft.

solutions to a particular problem using a language which would use various keywords used in the stock market and will be very close to the English language, then the solution will be processed by the code generator to generate an appropriate code. If need arises, whenever something needs to be changed in the program, all one has to do would be to change the high level design and send it through the code generator again. Thus, avoiding the need to touch the source code anymore.

The code generator is essentially a compiler which offers the user an abstraction of the underlying layers. The more abstract it is, the simpler it is for the user, and hence the narrower the domain is for the language itself.

To achieve our goal, the syntax of the language would be very closer to the English languages than to mathematical expressions and programming keywords. Also, to provide the extra layer of abstraction our compiler will reside on top of the current compilers and generate Perl, Java or C code instead of machine code.

Features

- Simple

  The intent of the code should be readily apparent to a non-programmer, specifically to individual traders who it targets. By taking the minimal set of necessary concepts from the stock market, the language should unnecessary complications.

- Intuitive

  The keywords and syntax should be closely related to the typical language used in the stock market. It should encourage a coding style that retains its sense in natural language when read aloud. The keywords and operators should be well-defined for the included datatypes, and take the most reasonable action from the perspective of a trader.

- Concise

  It should be possible to specify a simple behavior with a single, clean line of code. By specifying several strategies, the trader can implement comprehensive approaches to the market without losing clarity.

- Powerful

  It should enable the user to focus on crafting the logical architecture of a trading strategy by handling the implementation and monitoring that would otherwise be extremely time consuming.

Examples

Say we believe stock 'GM' is a good buy (undervalued), and competes directly with stock 'FORD' (overvalued), both of which depend on stock 'USSTEEL'. There are several reasonable behaviors to take advantage of this.

We could dump FORD and buy GM immediately:

```
sell all 'FORD'
buy 1000 'GM'
```

Or we could spread the buying out cautiously over time:

```
buy 1000 'GM' over 12h
```

But what if there's a sudden reversal in fortune? We could add a condition so we stop buying if the situation changes:

```
buy 'GM' over 12h while( value 'GM' < value 'FORD' )
```

Or switch so we're always buying the low (presumably undervalued) stock, assuming the steel industry isn't collapsing:

```
buy 1000 min('GM','FORD') over 12h while('USSTEEL'>100)
```

The broker can enumerate features of the market by defining sets of stocks, and thereby exploit relationships between industries. If car stocks are trending downward, we might ditch oil interests:

```
@cars = ('FORD','TOYOTA','HONDA')
@oil = ('OPEC','USOIL')
if(trend @cars < 0) sell all @oil
```

Function definitions will allow the reuse of code for recurring relationships or strategies:

```
function pump_and_dump(%target) = {
   buy %target until(trend %target > 1)
   sell all %target while(trend %target < 0)
  }
```