Artificial Markets for Computer Science

Karl Lieberherr

March 11, 2009

1 Introduction

We are interested in solving the real world question of how to produce reliable, evolutionary software. By evolutionary software we mean software that can be easily and safely evolved and that can be easily re-targeted.

Our thesis is: Developing design and programming tools using a generic artificial market (1) creates better design and programming technology for real world evolutionary software (2) creates better algorithms for the computational problem used as argument to the generic artificial market (3) makes creating better design/programming technology and better algorithms more fun and experiential (4) creates better teaching tools for software development courses that makes computer science more attractive.

An artificial market ArtificialMarket(C) for a computational problem C is driven by a feedback loop where computer science knowledge about C is put into trading robots (software robots) that will compete in simulations. The behavior of the robots is analyzed and new generations of robots are produced to compete in the next simulation. This fast turn-around requires tools to produce reliable, evolutionary software.

The robots produce and consume derivatives, both life-energy enhancing and life-energy draining. The challenge for the robots is to survive by accumulating enough life-energy.

ArtificialMarket(C) may evolve in the following three dimensions, independently or combined: (1) The computational problem C evolves. (2) The market definition evolves because the market was not fair. (3) The robots evolve because a better way to play the game was found.

ArtificialMarket(C) is interesting for several reasons:

1. It drives software development technology for evolutionary software.

2. ArtificialMarket(C) is a teaching tool to teach about computational problem C using a hands-on approach.

   An exciting aspect of this teaching approach is that the students become engaged with their robot which is a simple artificial organism.

3. ArtificialMarket(C) not only supports learning about C but also creates practically useful new knowledge about C. ArtificialMarket(C) is an excellent teaching tool to teach about computational problem C.

4. ArtificialMarket(C) has likely implications on real markets.

5. The artificial markets offer an interesting alternative to benchmark based evaluation. Instead of measuring the quality of an algorithm for the computational problem C on a benchmark that is static and needs to be enhanced from year to year, we submit the algorithm to ArtificialMarket(C) and watch its evaluation. This means, however, that the algorithms need to be good at both finding good solutions as well as creating instances for which its "hard" to find a good solution. Dealing with both issues is likely to enhance existing algorithms. Note how the robots will automatically create more and more challenging benchmarks based on their need to survive.

2 Evolutionary Software

ArtificialMarket(C) for a computational problem C provides indeed a context for solving the real-world evolutionary software problem (RES). The artificial market is a very important context because it drives RES in the right direction. Software development is much more than writing the program from a specification. It involves a lot of design, requirements analysis and potentially adjustments to the requirements if they don't make sense.

In the artificial market approach to RES the requirement is quite simple: Win within the rules of the market. But to turn that into evolutionary software is a challenging task because the artificial market needs to be thoroughly understood to win in the game.

Once you have that understanding of the market, you come to the lower level task of translating it into code. That is where our RES-supporting programming tools come in: We use Functional Adaptive Programming, implemented in DemeterF for Java and C#, to separate the important concerns that the programmer has based on her market understanding. DemeterF

is the best of all implementations of Demeter I know of. DemeterF supports safe evolution of class graphs thanks to a fine-grained type checking. DemeterF supports easy retargeting of computations to where they are run most effectively: heap versus stack, using multi-core without or minimal programmer intervention, etc. We need support to perfect DemeterF.

While we already have good support for RES at the programming level, we also would like to develop tools to help at the requirements analysis level. But this is longer range.

Our work on RES is primarily at the programming level and secondary at the design level and it uses artificial markets as the context to have fun in the process. The work is interdisciplinary because you can start with any computational problem C. And as part of developing RES for C, you will learn a lot about C.

3 Constraint Satisfaction

We have developed the artificial market loop for the computational problem called Constraint Satisfaction. ArtificialMarket(ConstraintSatisfaction) makes a strong contribution to engaging students in a robot development process where they evolve a baby robot (which they get at the beginning of a software development course) through several stages of development, using a baseball metaphor: T Ball, slow-pitch Softball, fast-pitch Softball to Baseball. Weekly competitions between the robots give immediate feedback on their "intelligence."

ArtificialMarket(ConstraintSatisfaction) is useful to teach development of reliable software using a solid base of computer science and mathematics.

In order to stay alive, the robots must reliably maximize the profit from their derivative trades and follow the rules of the artificial market. Unreliable behavior of the robots leads to their removal because of negative balance or other violations of the rules of the artificial market.

4 Summary

We propose the study of artificial markets to improve the knowledge in a computational problem C and to make teaching of this knowledge entertaining. We let the artificial markets drive our software development tools for evolutionary software.

5 Appendix: Artificial Market Definition

Given a computational problem C (optimization or decision problems), we define ArtificialMarket(C) to be roughly a tuple (Predicate, Derivatives, RawMaterials, FinishedProducts, Quality, Profit, Rules). The derivatives are predicates on the instances of C and have a price. After a derivative d = (pred,price) is bought, raw material is delivered which is an instance of C satisfying the predicate pred. The raw material consists also of the quality of a secret solution that the creator of the derivative has found and tries to "hide" from the buyer. The idea is that it is "expensive" for the buyer to recover or approximate the secret. The secret S is private to the creator at the moment. After the raw material r = (inst, quality(S)) has been delivered, the buyer finishes the raw material. i.e. produces a solution S' for r.inst. The profit is determined based on quality(S') compared to quality(S) (several profit formulas are possible). As the last step of the protocol, the creator reveals the secret S to the buyer. The Rules define the world: Every robot starts with an initial capital and must offer a new derivative on each round and must buy at least one derivative or reoffer all derivatives currently for sale at a lower price ...

5.1 Signatures

Sets and functions:

1. Predicate: An intentional definition of a subset of raw materials. Defined by a predicate language defined by a grammar. $Predicate : RawMaterial(C) \rightarrow Boolean$.
2. Price: real.
3. Derivative = (Predicate, Price)
4. $Quality : RawMaterial\ FinishedProduct\ Resources \rightarrow Real$.
5.