Basic Genetic Algorithm Pattern for Use In Self-Organizing Agile Security

Copyright Material IEEE
Paper No. ICCST-2012-41

Rich Messenger
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030
USA

Rick Dove
Member, IEEE
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030
USA

Abstract—Security strategies and techniques are falling behind the agile pace of adversarial innovative capabilities. A project is underway that has identified six so-called SAREPH characteristics of adversarial self-organizing agility, and is now cataloging patterns toward a pattern language of self-organizing security techniques that can be employed for equal or superior security agility. Many such patterns have recently been developed. This paper adds the Genetic Algorithm (GA) to the catalog. The essence of a GA is to express the problem to be optimized in terms of a "fitness function" that evaluates how well candidates optimize the solution. In natural evolution fitness is an organism's ability to survive and reproduce. Computing applications abstract fitness to match the problem at hand, such as an Intrusion Detection System attempting to correlate seemingly unrelated events that collectively constitute a threat. Reviewed first are the pattern project and the general nature of the GA. A reusable generic pattern description is developed. How the pattern conforms to the SAREPH characteristics is shown. Then three examples from the literature show how the pattern is employed in SAREPH conformity: predator-prey behavior evolution in robot swarms, future behavior prediction in financially traded stocks, and attack detection in an Intrusion Detection System.

Index Terms—Intrusion detection, SAREPH.

I. INTRODUCTION

Self-organizing agile security appears to be a necessary and new strategy to gain at least parity with the agility and self-organization exhibited by the constant innovation and evolution of adversarial communities.

The SDOE 683 graduate course on self-organizing agile systems at Stevens Institute of Technology is identifying and cataloging patterns for self-organizing agile security [1]. The International Council on Systems Engineering (INCOSE) Systems Security Engineering Working Group expects to publish a catalog of refined and consistent patterns as an INCOSE product when a sufficient number of patterns are developed. The project is currently in a preliminary pattern discovery phase, with minor changes in pattern representation consistency and pattern qualifications both converging toward a stable state.

The purpose of identifying these patterns is to accelerate the adoption of this new strategy, and to do so by developing a relevant common pattern language for systems engineers and security engineers to discuss and understand the essential characteristics.

The pattern project identified six common characteristics of agile security systems, modeled on the advantageous agile-practice characteristics observed in the security adversarial communities. These six characteristics are referred to as SAREPH, as summarized in Table I.

TABLE I
PATTERN QUALIFICATION SAREPH FILTERS

[S]	**Self-organizing** – with humans embedded in the loop, or with systemic mechanisms.
[A]	**Adapting** to unpredictable situations – with reconfigurable, readily employed resources.
[R]	**Reactively resilient** – able to continue, perhaps with reduced functionality, while recovering.
[E]	**Evolving** with a changing environment – driven by situation and fitness evaluation.
[P]	**Proactively innovative** – acting preemptively, perhaps unpredictably, to gain advantage.
[H]	**Harmonious** with system purpose – aiding rather than degrading system/user productivity.

Table II shows the format that the pattern project is using to capture and display essential pattern features. Each candidate pattern is described in this structure. References are provided in the pattern structure to facilitate deeper study of pattern employment examples in a variety of domains, where pattern-employment nuance under different situations broadens the understanding.

TABLE II
PATTERN DESCRIPTION FORMAT

Name: Descriptive name for the pattern.
Context: Situation that the pattern applies to.
Problem: Description of the problem.
Forces: Tradeoffs, value contradictions, key dynamics of tension and balance, constraints.
Solution: Description of the solution.
Graphic: A depiction of response dynamics.
Agility: Qualifying SAREPH characteristics.
Examples: Referenced cases of pattern employment.

Pattern projects generally assemble commonly used patterns that have been employed within a single domain, such as construction architecture [2] or traditional-security patterns [3]. Self-organizing cyber and physical security

IEEE International Carnahan Conference on Security Technology (ICCST), Boston, MA, USA, 15-18 Oct. 2012. Corrections made 2Aug2012

doesn't have an historical record from which patterns can be drawn; so at this stage the pattern project is drawing upon a variety of domains where appropriate patterns can be found, most frequently in natural systems, where sustainable security is a fundamental and primary objective.

Consequently, non-cyber/physical security domain examples are necessarily used to provide pattern base-lines, examples, and application nuance.

The working theory for the pattern project requires that pattern candidates include at least four of the SAREPH characteristics as follows:

- The pattern exhibits both Self-organizing and Harmonious characteristics.
- The pattern exhibits either or both of the Evolving and Adapting characteristics.
- The pattern exhibits either or both of the Proactive and Reactive characteristics.

Evolution in natural systems occurs through successive generations, and the genetic algorithm typically employs this concept. In artificial systems the concept of successive generations may become blurred at times, when a system maintains identity but adapts or improves certain aspects of its functionality through experimentation and selection that may employ a genetic algorithm. In this later case the pattern project wants to clearly distinguish between evolution and adaptation; though at this early stage some confusion may be evident in some pattern descriptions previously published.

This paper presents a *basic* genetic algorithm as a candidate pattern for the self-organizing agile security pattern project.

II. GENETIC ALGORITHM INTRODUCTION

Life on earth has a breathtaking diversity, thriving in innumerable niches. Natural selection (survival of the fittest) and variety arising from reproduction mechanisms has evolved organisms and their life sustaining processes suitable to diverse niches. In computer science, a genetic algorithm (GA) is an abstracted computational model of the underlying mechanism of natural evolution, typically applied to learning, searching, and optimization problems.

The genetic algorithm methods described here are based on techniques initially developed by John Holland and his students and colleagues at the University of Michigan in the 1960's and 1970's, popularized by the publishing of his landmark text on the subject [4]. Since that time the GA has been a widely-used algorithm to address problems in optimization and in modeling artificial life [5].

The essence of a genetic algorithm is to express the solution to an optimization problem in terms of a "fitness function" that evaluates how well candidate solutions address the problem. In natural evolution, fitness is the organism's ability to survive and reproduce. Computing applications, on the other hand, abstract fitness to match the problem at hand. In robotics, for example, fitness may represent a robot's ability to navigate successfully. In keeping with the evolutionary metaphor, the candidate solutions are called "individuals," the encoding of each candidate solution is called its "chromosome", and each distinct element within a chromosome is called a "gene".

A common computational representation of a chromosome is a bit string or a string of integers for simplicity of manipulation; but many variant representations are possible. With this structure in place, the GA applies "operators" based on natural evolution to evolve an initial randomly-generated (un-fit) population of individuals into more fit individuals, until a suitably fit individual emerges. The GA creates new generations of individuals from previous generations, evolving their chromosomes until a time limit is reached or until sufficient fitness is achieved.

The choice of chromosome representation may make the difference between a successful GA and a failure. For example, Forrest and Mitchell [6] discuss chromosome encoding schemes whose properties inhibit the GA from reaching higher-order solutions due to the combinatorial relationship between genes in the chromosomal representation and the corresponding features of the best solution.

To create a new generation, the GA evaluates the fitness of the individuals in the current generation using the fitness function, and then selects the fittest individuals stochastically, i.e., with some randomization. The fittest individuals are selected to breed new individuals for the next generation with the GA *crossover* operator, which mimics sexual reproduction in nature – two individuals are selected to breed, and their children's chromosomes are created by swapping a portion from each parents' chromosome at a randomly-selected location. Other moderately fit individuals in the current generation are allowed to survive to the next generation, and the least fit individuals are replaced by the new products of crossover to maintain the overall population size. The other commonly applied GA operator is *mutation* – occasionally, a random gene (bit or string of bits) of a chromosome is changed for the next generation.

There is a third GA transform operator modeled on natural evolution called *inversion*, which randomly reverses the order of a group of genes in the chromosome – a bulk form of mutation effective and efficient for certain problem types. Mitchell and Forrest noted in a seminal 1994 paper [5] that "Inversion is rarely used in today's GAs, at least partially because of the implementation expense for most representations." That implementation expense may not be the factor today it was then; but as crossover and mutation are able to evolve the same chromosomal outcomes, the Basic Genetic Algorithm pattern described in this paper includes only the selection operator and the crossover and mutation transform operators, recognizing them as minimally necessary.

For the interested reader, there are many variations and additional operators for GAs described by Aga in [7]; but the essential features are chromosomal encoding of multiple distinct individuals, repetitive selection of the fittest individuals over multiple generations, and stochastic application of transformation operators to create new generations.

The inherent randomness of the selection, crossover, and mutation operators allows a GA to explore the "fitness landscape" to find novel solutions. Introducing random changes to the current best solutions is a trade-off between exploration of new territory in the fitness landscape and exploitation of the currently-known best solutions (local optima). The mutation operator is applied much less often than crossover, but importantly helps the GA avoid getting stuck at local optima.

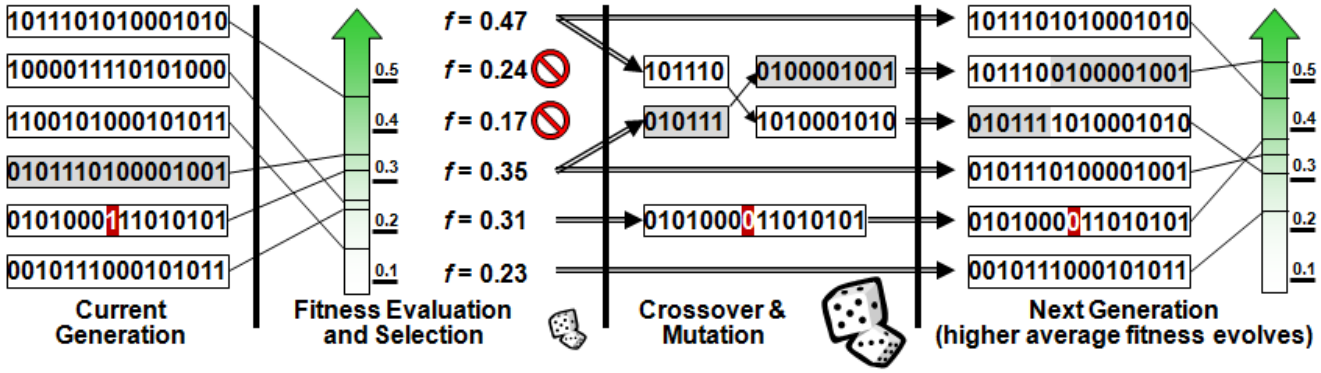
GAs explore many independent hypotheses in parallel at the same time; and are well suited to problems where the fitness landscape is complex and not well structured. Given a suitable problem, GAs converge relatively quickly on good solutions, but may not converge on the absolute optimal solution – resembling biological natural selection.

For more on the subject, a commonly cited book is Melanie Mitchell's *Introduction to Genetic Algorithms* [8], and John Holland has a very readable *Scientific American* [9] introduction to genetic algorithms available on the web.

III. GENETIC ALGORITHM PATTERN

Table III describes the basic genetic algorithm as a candidate pattern. Note that there are many ways to embellish this algorithm in detail implementations not addressed here.

TABLE III
BASIC GENETIC ALGORITHM PATTERN

Name: Basic Genetic Algorithm (with selection, crossover, and mutation operators)
Context: A system or process faced with a multi-dimensional problem that requires systemic learning or searching to find an optimal solution.
Problem: Multiple or complex trade-offs for optimization; lack of *a priori* understanding of how to solve a problem; environment may vary over time; hill-climbing or other optimization algorithms get stuck at local optima.
Forces: Exploration vs. exploitation; computationally-intensive vs. approximate fitness function; rate of crossover vs. mutation; rapid convergence vs. more trials.
Solution: Express the problem domain with a genetic representation amenable to crossover and mutation operators. Develop a function to evaluate the fitness of individuals at each generation. Choose the initial generation of individuals, randomly or by seeding. Create successive generations by selecting the fittest individuals in the current generation by applying the fitness function with some random variation, and then randomly applying genetic operators (crossover of chromosomes from two individuals, random mutation of a chromosome, possibly others) on these fit individuals to create uniquely-new individuals for the next generation, with higher average fitness than the previous generation.
 Genetic Algorithm Evolution: Best fitness evaluations–keep all; midrange fitness evaluations–keep some, mutate others; worst fitness evaluation–replace with crossover mix of best fitness evaluations.
[S]elf organization occurs as iterative chromosome experimentation searches for optimal chromosome configurations that satisfy the fitness landscape. [A]daptation occurs within the GA pattern when it accommodates additional or replaced factors in the fitness evaluation required by a change in the environment. [R]eactive resilience is exhibited when unfit or low-fit chromosomes are identified and replaced by new alternatives, and with continued fitness evaluations after convergence on an optimal set in case a changing environment reduces the fitness of the converged set. [E]volution occurs as a stable fitness function converges on the discovery of a solution with optimal fitness. [P]roactive innovation occurs by breeding uniquely-new speculative individuals at each generation using the genetic operators of selection, crossover and mutation. [H]armony between generations is maintained when the majority of good-fit individuals are preserved from the previous generation, minimizing the effects of new-chromosome experimentation.
Example – Natural evolution – variations in next generation created by crossover (sexual reproduction) and mutation of chromosomes in fit individuals (i.e., those that survive to breed) [4, 9]. **Example –** Robotics – GA that trains neural networks for collision-free navigation, homing, predator-prey co-evolution, joint evolution of brains and body morphology, and evolution of cooperation and altruism [10]. **Example –** Financial forecasting – genetic algorithm to predict the performance of publically traded stocks [11]. **Example –** Cyber security – Intrusion Detection System (IDS) using a genetic algorithm [12]. **Example –** Cyber security – Genetic algorithm used for network intrusion detection [13]. **Example –** Cyber security –Attack detection using a genetic algorithm in a policy based network [14].

Three examples from the literature in diverse domains are discussed next. Each example is characterized with respect to the SAREPH principles described in Table I.

IV. ROBOT BEHAVIOR EVOLUTION

Floreano, in a fascinating and revealing report [10], employed GAs to evolve neural networks that control simple robots. These small wheeled robots learned collision-free navigation and homing, cooperation and altruism behaviors, and continuous cycle predator-prey co-evolution. For a deeper treatment of GAs combined with artificial neural networks see [15].

In all cases a simple neural net controlled a robot's wheel speed and direction as output, with input from eight object distance sensors. In some cases additional sensors were employed, such as one for detecting the robot's recharging "home" for the homing test, and sensors on "prey" that could distinguish a "predator", and vice versa. The fitness function for the different cases was set for the experimental task.

Neural net connections have relevance weights, from zero for no connection to something with positive or negative significance. A robot's GA evolves chromosomes that designate these weights, through successive generations of robot "life" in a fixed population of robots. At the completion of each generation's life span, the GA stochastically selects chromosomes with promise to create the next generation of robotic neural net weights. Fig. 1, reprinted from [10], depicts the GA's neural-net-weighting evolutionary cycle. Only crossover and mutation chromosome-transformation operators were employed.

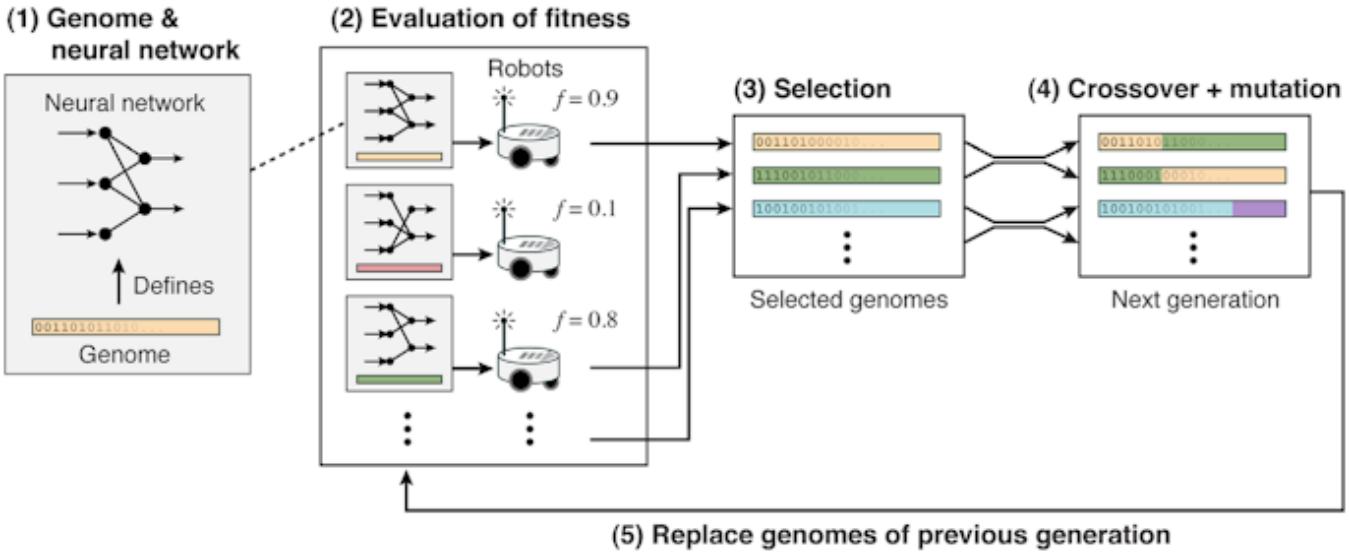


Fig. 1 – (as depicted and captioned in [10]) Major steps of Darwinian selection with robots. 1) The robots have a neural network with the strength of connections between neurons determining their behaviour as a function of the information provided by the environment. 2) The fitness *f* of each robot (i.e., the performance in the task assigned to them) is measured in the experimental setting using real robots or physics-based simulators. 3) The genomes of robots with highest fitness are selected to form a new generation. 4) The selected genomes are paired to perform crossover and mutations. 5) The new genomes are used to perform a new round of selection in the next generation.

Robots with different chromosomes behave differently, affecting their fitness. Fitness may be defined as how fast and straight a robot moves and how often it avoids obstacles; or in the case of predator-pray experiments, how successful prey is at avoiding contact with predator, and how successful predator is at catching prey, within a time limit.

Typically robots from a population of 80-100 were employed with different initial chromosome weight-genes. Trials within a generation were done two robots at a time, within a small walled perimeter containing additional obstacles for some test cases.

Floreano notes: "In all cases, robots initially exhibited completely uncoordinated behaviour because their genomes had random values. However, a few hundreds of generations of random mutations and selective reproduction were sufficient to promote the evolution of efficient behaviours in a wide range of environmental conditions. The ability of robots to orientate, escape predators, and even cooperate is particularly remarkable given that they had deliberately simple genotypes directly mapped into the connection weights of neural networks comprising only a few dozen neurons."

Notably, Floreano discusses the shortcomings of generation trials in simulation vs. those carried out in the "real world" – in this case meaning actual robots navigating within the experimental perimeter. Real-word evolution discovers idiosyncratic environmental factors of value that are typically missing in simulation models. Nevertheless, many of the trials employed simulations initially to speed up early convergence and then finished with a series of real-world trials for final chromosome evolution.

This example of GA employment exhibits all six of the SAREPH characteristics, as described next.

Self-organizing **–** Each new generation of robotic control is systemically re-designed by using the crossover and mutation operators on the chromosomes from the fittest robots of previous generation. The fittest behaviors are selected as those that achieve the highest measured ability to achieve the

task at hand. Behavior trait convergence is self-organizing without external decision-making.

Adapting to unpredictable situations – The predator-prey case is a good example of adaptation to a changing environment. As prey became more adept at avoiding predation, the predators adapted to their own less-fit condition and evolved new successful strategies, which in turn caused the prey to adapt by evolving new avoidance strategies. The basic GA implementation remained the same, converging on new chromosome sets appropriate to the environmental change.

Reactively resilient – The GA generates new chromosomes with crossover and mutation operators rather than wholesale random replacement. This has the effect of maintaining some prior history in new chromosomes, which lends resilience when chromosome changes are less fit in a new generation than in a prior generation. In the predator/prey task, the predators react to the behavior of the prey, and evolve a new approach to predation that increases their fitness, thereby lowering the relative fitness of the prey. This causes the prey to evolve new behavior to counteract the strategy of the predators, raising their own fitness which lowers the relative fitness of the predators. And so it continues, in an ongoing leap-frog co-evolution.

Evolving with a changing environment – Chromosome evolution is inherent in a sufficiently designed GA, and satisfactory chromosomes will evolve if that is possible. The predator-prey experiments affirm that the GA employed can deal with a changing environment. In these robotic experiments, solution possibility is determined by sufficient sensory input, sufficient action capability, a sufficient fitness function, sufficiency of the chosen transform operators (crossover and mutation) and their stochastic parameters, sufficiency of selection-operator parameters, and an environment that permits a solution. These constraints on possibility are generally applicable to any GA application.

Proactively innovative – A properly implemented GA is by nature proactively innovative; it creates speculative chromosomes in a search for a superior chromosome. All of the robotic experiments exhibit this capability. One unique GA driven experiment discussed in [10] used a physics-based simulation to evolved robot body configurations, made up of simple building blocks: bars of various lengths, joined by ball joints, and coupled with linear actuators that changed their length, controlled by a simple neural network. The fitness function was the robots' ability to move a distance on a flat surface. After 300 generations, the fittest individual robot designs were fabricated and tested in the real world to validate the physics-based model. The picture of one of these evolved robots [10 Figure 5] looks like nothing designed by man, and is a clear example the GA acting innovatively and unpredictably.

Harmonious with system purpose – A common characteristic of all the robotic evolution experiments is that the evolution did not back-slide to lesser fitness in the aggregate as a species. Harmony between generations is preserved by tuning the genetic algorithm to make incremental improvements through random crossover and mutation of the fittest individual robots' chromosomes, without wholesale destruction of the evolutionary improvements gained to-date.

V. FINANCIAL FORECASTING

Mahfoud and Mani [11] describe a genetic algorithm to predict future performance of publically traded stocks. A GA is set up to analyze the performance of 1,600 publically traded stocks, and forecast the returns of each individual stock 12 weeks in the future, relative to the average return of all the stocks.

The information that the GA analyzes is time-phased historical data on 15 proprietary attributes (such as price-to-earnings ratio and growth rate) for each stock. Instead of a simple bit array, the chromosome in the financial forecasting GA is composed of if-then rules that manipulate these attributes. Mahfoud and Mani cite a rule evolved in one of the experiments as:

IF [Earnings Surprise Expectation > 10% and
Volatility > 7% and . . .]
THEN Prediction = Up

One interesting feature of this chromosomal representation is the evolved rules can be understood by humans directly. The GA evolves the individuals' chromosomes of if-then rules through selection, crossover, and mutation, which are then used to classify each of the stocks and produce a recommendation to buy or sell (or no recommendation), and a forecasted return for the stock 12 weeks in the future.

Mahfoud and Mani compare the results of the financial forecasting GA with an established neural network (NN). This neural network had been used by the authors for three years to forecast stocks. Over 1,600 stocks are analyzed by the GA and the NN, and a series of predicted values are made over 12 weeks. At the end of the period, the actual current values of the stocks are compared against the predicted values. In a related experiment, the relative realized rate of return for buys/sells is compared between the GA and the NN. The GA performs significantly better than the NN, and a combined GA+NN outperforms both. All three (GA, NN, and GA+NN) perform better than all market indices (Dow Jones Industrial Average, S&P 500, etc.) over the same period, to a statistically significant 95% confidence.

Self-organizing – Self organization is inherent in GA applications, exhibited here as if-then rules are reconfigured until they successfully characterize and predict performance on a set of stocks. This GA application has the capability to output if-then rules suitable for embedding in a formal expert system for future forecasting: "…no expert is required, and hence the tedious process of transforming that expert's knowledge into a set of rules is eliminated" [11: 562].

Adapting to unpredictable situations – The financial forecasting GA adapts to whatever historical stock data is available in its "environment"; achieving a suitable prediction regardless of the stock portfolio.

Reactively resilient – The financial forecasting GA shares the inherent resilience of all GAs that use stochastic (probabilistic) methods to create new generations of candidate solutions, which allows for recovery from sub-optimal solutions. If a clause of an if-then rule (e.g., some attribute less than some value) is evolved but does not contribute to an optimal solution, the random application of crossover and mutation operators are able to explore alternate paths without getting stuck with that clause.

Evolving with a changing environment – The financial forecasting GA starts with randomly generated if-then rules, then evolves them over the course of generations by evaluating the fitness of each if-then rule, selecting the fittest individuals for the next generation, and applying the crossover and mutation operators to create a new generation of on-average better if-then rules. Changing stocks within a portfolio (environment) will evolve a new set of predictions based on the portfolio content.

Proactively innovative – A GA is inherently proactive in its speculative search for solutions fit to the environment. The financial forecasting GA develops innovative if-then rules that are different than a human creates. According to Mahfoud and Mani [11: 562], "a [human] expert is likely to produce a set of rules qualitatively different than then those produced by a GA, due in part to the expert's *a priori* bias against counterintuitive or contrarian rules."

Harmonious with system purpose – Harmony between generations is preserved by tuning the GA to make incremental improvements through random crossover and mutation of the fittest if-then rules, without wholesale destruction of the evolutionary improvements to-date. Evidence of overall system harmony is the success of the GA compared with the neural network approach and with overall market indices.

VI. CYBER SECURITY INTRUSION DETECTION

Diaz-Gomez and Hougen [12] describe an off-line Intrusion Detection System (IDS) to examine audit trail (log) files. This work provides an example of the criticality of the fitness function for GA success. The Genetic Algorithm for Simplified Security Audit Trails Analysis (GASSATA) described by Mé [16] is analyzed and improved by Diaz-Gomez and Hougen, and the results of running a GA using these alternative fitness functions are compared.

Computer-system audit logs record what the system and users are doing. An IDS may then analyze these audit logs to detect and report abnormal behavior that may indicate an attack. The method used by Diaz-Gomez and Hougen searches an audit trail for patterns of activities of known attacks. There may be many types of attacks, each with a large number of component activities. These activities are typically innocuous in isolation, but indicate an attack when combined in certain ways. Identifying the most likely attacks given an audit log of actual activities is a difficult problem to solve directly (the search is an NP-complete problem).

In this example an attack matrix is constructed of all auditable events against all known attacks, where each element of the (sparse) matrix shows the number of occurrences of the event for that attack. The chromosome for each individual (candidate solution) is represented as a bit string, with a set bit indicating a hypothesis that the corresponding attack occurred. The fitness function evaluates how likely the hypothesized set of attacks occurred, given the actual events in the audit log. An initial population of individuals is generated randomly, and then successive generations are created through selection, crossover, and mutation. The GA is run through a number of generations until the change in maximum fitness between generations falls below some threshold.

The GA used in this IDS example exhibits all six of the SAREPH characteristics. However, it is noted that this IDS system, at the macro level, exhibits neither evolution nor proactive innovation, as discussed below.

Self-organizing – The GA in the IDS self-organizes to discover and analyze combinations of events in the audit log: over successive generations, individuals (candidate solutions) are reconfigured automatically toward higher average fitness, with no external decision-making.

Adapting to unpredictable situations – Adaptation is generally exhibited by a GA implementation when it adapts to a different environment. In this example the environment is the collection of known attacks and known attack activities. That collection would be augmented periodically if the example IDS system were deployed, and nothing in the reviewed example appears to restrict or preclude appropriate adaptation.

Reactively resilient – The GA in the example IDS performs crossover on selected parents with a probability of 60%, and mutation with a probability of 2.4% [12: 5]. This randomness in creating candidate solutions allows for recovery from sub-optimal solutions.

Evolving with a changing environment – Evolution is an inherent feature of any properly implemented genetic algorithm, but is dependent on the parameters employed for tuning the algorithm. With the adaptive capability and reactive resilience previously discussed, the reviewed example appears capable of evolving a new set of effective detection patterns if the environment changed. It should be noted, however, that the example IDS system does not evolve detection capability for zero-day never-seen-before attacks and attack activities.

Proactively innovative – There is proactive innovation at the level of the GA in the creation of uniquely new candidate solutions at each generation through the application of genetic operators. From a higher point of view, however, the range of proactive innovation in this IDS example is limited, as it only deals with known attacks and known attack activities.

Harmonious with system purpose – The GA implementation for this example IDS behaves harmoniously, preserving knowledge gained in previous generations through successive generations and converging on higher fitness. Again, this is generally inherent in any properly implemented GA, but is dependant on the nature of the fitness function and parameters tuned appropriately for the fitness landscape. This IDS's GA fitness function exhibited improvement over the fitness function used in the GASSATA project [16] by retaining history to limit the occurrence of false positives and help the GA converge.

VII. CONCLUSION

The six SAREPH principles were employed to test the qualifications of a basic genetic algorithm. Without surprise, the basic genetic algorithm is a candidate pattern for the self-organizing agile security pattern language under development. Genetic algorithms are presented here in the stylized format of a pattern to enhance communication and mutual understanding between systems and security engineers, and contribute to the understanding of self-organizing agile systems of any kind.

As illustrated by the three examples of a genetic algorithm employed in diverse domains taken from the literature – behavior-controller training of simple robots, financial forecasting, and an IDS for computer security – properly deployed genetic algorithms exhibit all six "SAREPH" characteristics.

Examining the pattern in multiple domains develops an appreciation of pattern employment under different circumstances, and helps focus pattern abstraction on domain independent fundamentals.

The computer security example, however, shows that although the GA pattern is inherently capable of exhibiting all six SAREPH characteristics with appropriate fitness function and parameter tuning, that does not necessarily mean that a system built using a GA will itself exhibit all of the SAREPH characteristics. A higher-level lack of agility in a system can limit the potential of lower-level agile processes employed by the system.

Patterns at this stage of the pattern project are considered preliminary. Each new pattern added helps refine the use of the generic pattern form as a description mechanism; and also helps refine the distinctions among the six SAREPH characteristics used as a pattern-qualifying filter. The work reported here wrestled with the need to include inversion as an operator – and concluded that there is value in a identifying the minimal GA pattern, with the expectation that an additional GA pattern will be added to the pattern family that includes inversion and the context and problem space in which inversion is necessary.

VIII. REFERENCES

[1] Rick Dove, "Pattern Qualifications and Examples of Next-Generation Agile System-Security Strategies," IEEE International Carnahan Conference on Security Technology (ICCST), San Jose, CA (US), 5-8 Oct. 2010,

[2] Christopher Alexander, A Pattern Language: Towns, Buildings, Construction, Oxford University Press, 1977.

[3] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, Peter Summerland, Security Patterns: Integrating Security and Systems Engineering, Wiley, 2006.

[4] John Holland, Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975.

[5] Melanie Mitchell and Stephanie Forrest, "Genetic Algorithms and Artificial Life," Artificial Life, 1 (3): 267-289, 1994.

[6] Stephanie Forrest and Melanie Mitchell, "What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation," Machine Learning, 13: 285-319, 1993.

[7] P. Larran Aga, C.M.H. Kuijpers, R.H. Murga, I. Inza and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," Artificial Intelligence Review 13: 129–170, Kluwer Academic Publishers, 1999.

[8] Melanie Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems). A Bradford Book, Third Edition, 1998.

[9] John H. Holland, "Genetic Algorithms." Scientific American, 267 (1): 66-72, 1992. http://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm

[10] Dario Floreano and Laurent Keller, "Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection," PLoS Biol 8(1): e1000292, Jan. 2010.

[11] Sam Mahfoud and Ganesh Mani, "Financial forecasting using genetic algorithms." Applied Artificial Intelligence, 10 (6): 543-565, 1996.

[12] Pedro A. Diaz-Gomez and Dean F. Hougen, "Improved Off-Line Intrusion Detection Using a Genetic Algorithm," *Proceedings of the Seventh International Conference on Enterprise Information Systems*, Miami (US), 24-28 May 2005.

[13] Wei Li, "Using Genetic Algorithm for Network Intrusion Detection." *Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference*, Kansas City, Kansas (US), 24-27 May 2004.

[14] Alim Al Islam, Ariful Azad, Khurshid Alam, Shamsul Alam, "Security Attack Detection using Genetic Algorithm (GA) in Policy Based Network," *Proceedings of the IEEE International Conference on Information and Communication Technology*, pp 341-347, 2007.

[15] David Streisand and Rick Dove, "Basic Genetic-Algorithm-Neural-Network (GANN) Pattern with a Self-Organizing Security Example," IEEE International Carnahan Conference on Security Technology (ICCST), Boston, MA (US),15-18 Oct. 2012.

[16] Ludovic Mé, "GasSAtA, a Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis," In First International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium, 14-16 Sep1998.

IX. VITA

Rich Messenger is a systems engineer in the U.S. defense industry, and during the past 27 years has been a software and systems engineer for a diverse collection of defense and intelligence systems and research projects, satellite ground stations, and bioinformatics systems. He holds a BS in Computer Science from Purdue University, and is working on an ME in Systems Engineering from Stevens Institute of Technology.

Rick Dove develops agile self-organizing systems as a principle investigator and application program manager at Paradigm Shift International, and chairs the INCOSE working group for Systems Security Engineering. He is an adjunct professor at Stevens Institute of Technology in the School of Systems and Enterprises where he teaches basic and advanced courses in agile systems, and is author of *Response Ability – the Language, Structure, and Culture of the Agile Enterprise* (Wiley 2001). He co-led the OSD/Navy-funded project that identified systems agility as the new competitive frontier, and then led the research at the DARPA/NSF-funded Agility Forum. He holds a BSEE from Carnegie Mellon University, with graduate work in Computer Science at U.C. Berkeley.