# A Scenario-Based Approach to Protocol Design Using Evolutionary Techniques

Sérgio G. Araújo, A. Mesquita, Aloysio C. P. Pedroza

Electrical Engineering Dept.
Federal University of Rio de Janeiro
C.P. 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brazil
Tel: +55 21 2260-5010 - Fax: +55 21 2290-6626
{granato,aloysio}@gta.ufrj.br mesquita@coe.ufrj.br

**Abstract**. An evolutionary approach to design communication protocols from scenario-based specifications is presented. It enables to automatically generate finite-state models of protocol entities from Message Sequence Charts. By converting the Message Sequence Charts into input/output sequences, the problem reduces to evolving finite-state machines with the specified input/output behaviors. The proposed approach does not overgeneralize the entity behavior producing, by construction, minimal, deterministic and completely specified finite-state machines.

## 1 Introduction and Related Works

Distributed systems play a significant role in the implementation of computer-based applications. The masking of the data/process location for the user, through reliable communication protocols, is the main concept behind these systems. Nevertheless, their modeling and implementation remain, today, a complex task. During the last two decades, several methods to formally describe telecommunication systems and services have been proposed with limited dissemination in industry. As an alternative, much attention has been given to the more intuitive semi-formal approaches, such as interaction scenarios. Scenario-based modeling is particularly well suited for operational descriptions, which are critical in reactive systems, and can be introduced in iterative and incremental design processes [1].

State-oriented specifications and interaction-oriented descriptions techniques provide orthogonal views of systems. Automata represent projections of the complete system behavior onto individual *components*, while Message Sequence Charts (MSCs) represent projections of the complete system behavior onto particular *services*. Because automata usually reveal, to a certain degree, how a particular behavior is achieved, their typical position in the system development process is closer to design and implementation than to requirements capture [2].

Clearly, the application of system scenarios, typically used in reverse engineering, within a forward engineering approach calls for precise conversion between models.

So, a number of strategies for translating scenarios to automata have been proposed in the recent literature [2, 3]. The SCED algorithm [3] employs the domain-specific assumption that the capability of outputting a specific message uniquely identifies the state of a component. SCED may produce automata with more general behavior than the scenarios themselves as a consequence of the merging of scenarios into state-transition paths. While examples in a typically inductive system can be either positive or negative, scenarios consistently represent positive data and, thus, do not give sufficient information about the target system for inference [4]. Krüger [2] derived an automaton from a given MSC specification by successively applying four transformation steps: 1. *Projection* of MSCs onto the component, 2. *Normalization* to determine the transition-path segments defined by the projected MSCs, 3. *Transformation* into an automaton by turning every message into a transition and by adding intermediate states, and 4. *Optimization* of the resulting automaton. In contrast to SCED the automaton produced by Krüger's approach is neither deterministic nor minimal by construction requiring a distinct optimization step.

On the other hand, Evolutionary Algorithms (EAs) and variations have been used in the synthesis of sequential machines enabling to efficiently explore the solutions space of this category of design problems. Early attempts [5] used some of the EAs concepts to evolve an automaton that predicts outputs based on known input sequences. Lacking the crossover genetic operator [10] the approach has shown, however, poor performances. Recent works have been successful in synthesizing sequential systems with the aid of Genetic Algorithms (GAs) [6, 7, 8]. An approach to synthesize synchronous sequential logic circuits from partial input/output (I/O) sequences is presented in [6]. By using a *technology-based* representation, essentially a netlist of gates and flip-flops, the method was able to synthesize a variety of small FSMs, such as serial adders and 4-bit sequence detectors. Others methods [7, 8] used a technology-independent *state-based* representation in which the next-state and the output corresponding to each current-state/input pair of the state-transition table are coded in a binary string defined as the *chromosome*.

In this work an approach based on evolutionary techniques for the construction of protocol specification models from scenarios is proposed. The method is efficient in generating minimum automata by construction and differently from other approaches [3] does not overgeneralize the entity behavior.

## 2 Definitions

### 2.1 MSC and FSM

Message Sequence Charts (MSCs) are scenario notations widely used for requirements capture in the telecommunications domain [1]. A MSC is a graphical scenario consisting of *vertical axes* and *arrows*. The vertical axes represent the system reactive instances, or communicating entities, and the arrows, directed from a sending to a receiving instance, denote a communication event.

Since MSCs usually describe a partial behavior of the system, a question arises about how complete the information contained in the MSCs is and how to transit between possible and mandatory behaviors. It is convenient by this time to correctly understand the *interpretations* of the MSCs with respect to the system under development. Krüger [2] defines four such interpretations: 1. *Existential*: the behavior can, but does not need to occur during the system execution, 2. *Universal*: the behavior must occur in all executions, 3. *Exact*: explicitly prohibits other behaviors than the ones specified through the MSCs and 4. *Negation*: used to describe forbidden or undesirable protocol behaviors. The exact interpretation not only requires what *can* or *must* happen (the behavior explicitly represented by the MSCs); it also makes precise what *must not* happen (everything else).

Finite-state machines (FSMs) are commonly used for specifying communication protocols [9]. At the same time, the FSM is the mostly used target model in construction schemes starting from scenarios [1]. The Mealy FSM model $M$ is formally defined as a 7-tuple $\{Q,V,t,q_0,V',o,D\}$ where $Q \neq \emptyset$ is a finite set of states of $M$, $V$ is a finite input alphabet, $t$ is the state transition function, $q_0 \in Q$ is the initial state, $V'$ is a finite output alphabet, $o$ is the output function and $D$ is the specification domain, which is a subset of $Q \times V$. $t$ and $o$ together characterize the behavior of the FSM, i.e., $t(q,v)$: $Q \times V \rightarrow Q$ and $o(q,v)$: $Q \times V \rightarrow V'$. If $D = Q \times V$, then $t$ and $o$ are defined for all possible state/input combinations and therefore the FSM is said to be *completely specified*.

### 2.2 Evolutionary Computation

Evolutionary Computation (EC) consists of the design and analysis of probabilistic algorithms inspired by the principles of Darwinian natural selection. The parallel search performed by EC differs from conventional problem solving techniques in that it does not depend on decomposition. Therefore, EC solutions are best applied to highly non-linear problems for which deterministic solutions are not available.

Genetic Algorithm (GA) and Genetic Programming (GP) are among the most widely used instances of EC. The *chromosome* is the basic component of the GA. It represents a point of the search space of the problem. The fitness value measures how close the individual is to the solution. By iteratively applying the genetic operators (fitness evaluation, fitness-based selection, reproduction, crossover and mutation) to a randomly started population of individuals, it gradually evolves in order to breed at least one offspring with the desired behavior. GA, which usually works with strings (chromosomes) of *fixed length*, is widely used in optimization problems. GP [10] is a branch of GA, the main difference being the solution representation. Unlike GA, GP can easily code chromosomes of *variable length*, which increases the capacity in structure creation.

Programs, or structures, may be produced by combining context-free grammar (CFG) with GP, known as $G^3P$ (Grammar-Guided GP). GP Kernel (GPK) [11] is a complex $G^3P$ system and was used in this work to evolve trees, each encoding an automaton, from a Backus-Naur form (BNF) provided as input.

## 3 Methodology

The proposed methodology searches a deterministic FSM, possibly minimal with respect to the number of states, consistent with a given sample of I/O behaviors extracted from MSCs. The I/O behaviors, in the following called *training sequences* (*TSs*), are derived from the projection of the MSC specification onto the entity to be modeled. The execution flow of the methodology is drawn in Fig. 1a: the GP box outputs a population of FSMs for fitness evaluation based on *TSs*; if at least one individual reproduces the I/O behavior given by the *TSs*, the algorithm stops. The resulting FSM describes the protocol entity.

The model adopted for fitness evaluation (shaded box of Fig. 1a) uses the *black-box* approach (Fig. 1b), in which the entity to be evolved interacts with the cooperating entities through an event-driven interface. It is assumed, as in most protocol synthesis approaches [12], that the communication system has reliable FIFO channels. In each generation the system probes a population of FSMs with input sequences and records the corresponding output sequences. These output sequences are compared with the *correct* output sequences, i.e., the outputs of the *TSs*, and a fitness value is assigned to each candidate solution.

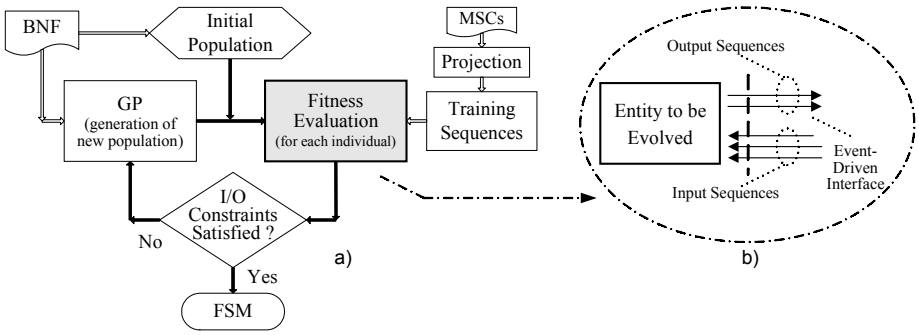


**Fig. 1.** a) Execution flow of the methodology; b) Model for fitness evaluation

### 3.1. Derivation of Training Sequences from MSCs

A trace or *training sequence* (*TS*) is defined as a finite sequence of *correct* I/O pairs < $v_1/v_1'$, $v_2/v_2'$… $v_L/v_L'$>, where $v \in V$, $v' \in V'$ and $L$ is the length of the sequence. The derivation method is based on the exact interpretation of the MSC specification. The derivation of *TSs* from a set of MSCs is a two-step procedure:

**Step 1.** Generation of *TSs* with positive mandatory information by projecting all the MSCs onto the object for which the state machine will be synthesized. A projection is carried out by traversing the vertical line of that object, from top to bottom; sequential ordered incoming and outgoing events are grouped into *TS* I/O pairs. Sequential composition of traces from two MSCs containing the object of

interest is achieved by means of condition symbols; the random combination of traces in such a way allows the generation of diverse, lengthy *TSs*.

Fig. 2 depicts the MSC specification of a refined connection-oriented protocol. These MSCs have two condition symbols, ACTIVE and IDLE, used to link the MSCs.

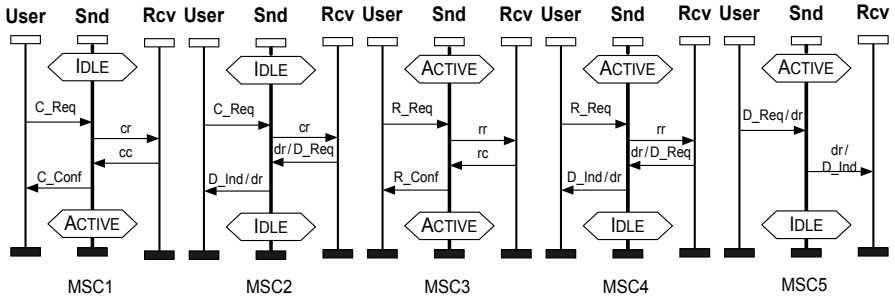


**Fig. 2.** Connection-oriented protocol MSC specification

The following *TSs* were generated from the procedures described in the first step, by selecting the sender entity as the target object:

$ST_1$: <C_Req/cr, cc/C_Conf, D_Req/dr> (MSC1 and MSC5);
$ST_2$: <C_Req/cr, cc/C_Conf, R_Req/rr, rc/R_Conf> (MSC1 and MSC3);
$ST_3$: <C_Req/cr, dr/D_Ind, C_Req/cr, cc/C_Conf, dr/D_Ind> (MSC1, MSC2 and MSC5).

The *TSs* produced in step 1 does not cover unspecified (non-expected) input events, resulting in incomplete information for state-machine inference. The concept of *completeness* is essential to achieve equivalence between the synthesized FSM and the unknown one from which the sample traces are taken. Completeness implies that there exists a transition for every input event in every state; this assumption assures that the traces will eventually give sufficient information about the state machine for inference [13]. Moving from partial machines to completely specified ones is accomplished in basically two ways: requiring that the state machine remains in the present state without producing any output (i.e., reacting with the *null* output) for any unspecified input or forcing a transition to an added "error state". The implementation of the former on traces is straightforward, as seen in the next step.

**Step 2.** Insertion of negative information into the *TSs*, obtained in the first step, by randomly adding unspecified input events between I/O pairs of such *TSs*; in this case, the pair is completed with the *null* output forcing the automaton to perform a self-loop. The unspecified input event set ($V_U$) is inferred from the "exact" MSCs, as follows: $V_U = V - V_E$, where $V_E$ is the expected input event set obtained from the MSC specification. It is assumed that unspecified input events occur with uniform distribution, i.e., $P(E_i) = 1/N$, for $E_i \in V_U$, $i = 1,...,N$.

This step ensures that each state of the entity is able to respond to every input of its input alphabet, i.e., the construction process yields a completely specified FSM. The following *TSs* were obtained at the end of the second step:

$ST_{1:}$ <C_Req/cr, **rc/Null**, cc/C_Conf, **cc/Null**, **rc/Null**, D_Req/dr>;
$ST_{2:}$ <C_Req/cr, **C_Req/Null**, cc/C_Conf, **cc/Null**, R_Req/rr, **D_Req/Null**, rc/R_Conf>;
$ST_{3:}$ <**R_Req/Null**, C_Req/cr, **rc/Null**, **R_Req/Null**, dr/D_Ind, **rc/Null**, **cc/Null**, C_Req/cr, **R_Req/Null**, cc/C_Conf, **cc/Null**, dr/D_Ind>.

### 3.2. Length of the Training Sequences (*TSs*)

The *TSs* must be long enough to exercise all paths of the FSM that describes the protocol entity. Ref. [6] gives an approximation formula to estimate the length of the *TSs* that yields a correct FSM, based on the *waiting times in sampling* problem solution. This formula defines the length of the input sequence as $L = E(S) \times E(I)$, where $E(S)$ and $E(I)$ are the *expected value of the number of state transitions* and the *expected value of the number of inputs*, respectively. The expected value of the number of state transitions to traverse all states of a given FSM can be computed using $E(S) = S\,(S + {}^{S}/_{2} + \ldots + {}^{1}/_{S})$. Likewise, the expected value of the number of inputs to traverse all paths from a given state is evaluated by $E(I) = I\,(I + {}^{1}/_{2} + \ldots + {}^{1}/_{I})$. However, since the number of states $S$ required to describe the protocol entity is unknown, it must be overestimated a priori.

### 3.3. Chromosome Coding and BNF Definition

The chromosome, which encodes a FSM, uses state-based representation (SBR). The resulting string (chromosome) with $S$ states and $I$ inputs is shown in Fig. 3.

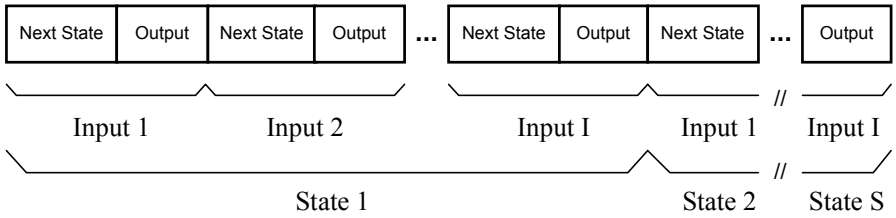


**Fig. 3.** Chromosome coding using SBR

Unlike other approaches [7, 8], in the present one the length of the chromosome is allowed to vary. GP algorithm was used, since this is the most efficient way to implement an evolutionary process with variable length chromosomes known to date. The BNF that allows the variable length chromosome coding is defined as:

```
S          :=  <expr>;
<expr>     :=  <state>|<expr> <state>;
<state>    :=  <next_st><out> <next_st><out> <next_st><out>
               <next_st><out> <next_st><out> <next_st><out>;
<next_st>  :=  "0"|"1"|"2"|"3"|"4"|"5";
<out>      :=  "0"|"1"|"2"|"3"|"4"|"5"|"6";
```

where <*expr*> allows the chromosome to vary in length, while <*state*> fixes the number of input events (six, in the above BNF definition, each yielding a next-state/output pair, i.e., <*next_st*> <*out*>).

### 3.4. Fitness Evaluation

The number of states used to implement the FSM weights the fitness value assigned to a given FSM behavior. The fitness function *F* has the form:

$$F = \left( \sum_{i=1}^{N} w_i H_i \right) * \left(1 + K(S) * GR\right) \quad (1)$$

where *N* is the number of fitness cases (*TSs*), $w_i$ is a weighting factor for fitness case *i* ($TS_i$), $H_i$ is the number of *output hits* due to the fitness case *i*, *K(S)* is a weighting factor that considers the effect of the number of states *S* used to implement the FSM and *GR* is the gradient of the overall output hits of the best individual. $H_i$ is computed as follows. Initially, the FSM must be in the reset state. In the sequence, for each input of the $TS_i$, its output is compared with the *correct* output and an output hit is signed in case of a match. *K(S)*GR* raises *F* for FSM implementations with lower number of states. However, as the fitness value tends to get trapped into local optima for an inadequate FSM, *GR* goes to zero allowing fair competition among FSMs regardless of their sizes.

## 4. EXPERIMENTAL RESULTS

As an example, the synthesis of a protocol entity specification for the sender side, $PS_S$, of the connection-oriented protocol from given interaction scenarios, is discussed. The training sequences, *TS,* were produced from the MSC specification of Fig. 2 using the derivation method presented in Subsection 3.1. The *TS* length was evaluated using six inputs (*I*) and an estimated value of five for the parameter *S,* as described in Subsection 3.2, leading to a 168-input/output *TS*. In fact, eighteen (*N* = 18) 32 bits-*TSs* ($w_i$ = 1) were used corresponding to more than three 168-length *TSs*.

The problem consists in finding a minimum-state deterministic FSM consistent with the derived *TSs*. The population size, *M,* and the maximum number of generations, $G_{MAX}$, were chosen as 500 and 3,000, respectively. The two-point crossover and mutation probabilities were set to $p_c$ = 0.65 and $p_m$ = 0.05, respectively. Linear rank selection was used considering the elitist strategy. The population of FSMs was created using the BNF described in Subsection 3.3 allowing FSM implementations with up to six states. The factor *K(S)* was defined by the set of values *K(S)* = {0.01, 0.008, 0.006, 0.004, 0.002, 0.000} for *S* = {1, 2, 3, 4, 5, 6} respectively. *GR* was continuously evaluated over 50 generations, except for the first 50 generations and at the end of the evolution where it was set to 1; in this last case

because *GR* tends naturally to zero. To improve the population diversity the mutation probability, $p_m$, was increased to 0.15 whenever $GR = 0$.

Fig. 4 depicts the fitness curve of the best FSM, its number of states and *GR* for a typical run. In this figure, it can be observed that at the first 50 generations the number of states of the best FSM fluctuate significantly due to the high population diversity. The same behavior is observed between generations 153 and 205, this time as *GR* is set to zero. Finally it can be observed that, in general, each increase in the fitness value of the best FSM is correlated with an increase followed by a decrease in the number of its states. This is particularly perceptible at the first 50 generations and between generations 260 and 350. The state-transition graph (STG) of the 4-state resulting FSM, which successfully describes the $PS_S$ after 576 output hits at generation 435, is given in Fig. 5.

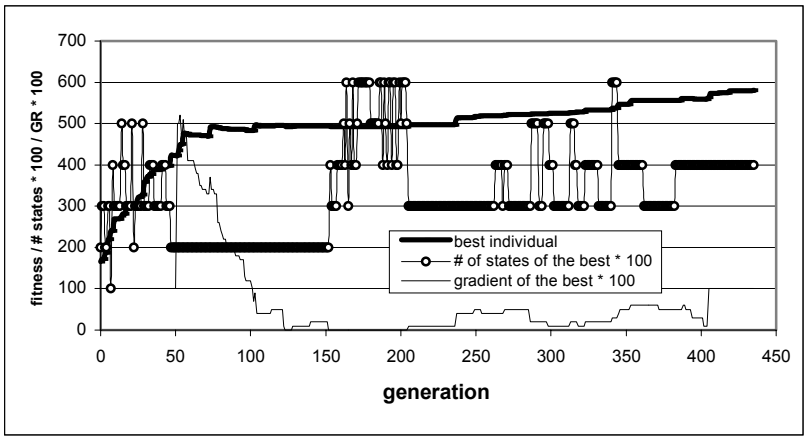


**Fig. 4.** Fitness curve of the best individual

It should be mentioned that the proposed system does not generate the minimal FSM in all successful runs, i.e., runs that achieve all output hits within the specified $G_{MAX}$. In some trials the fitness value of the best individual got stuck into local optima for a great number of generations. In these cases the system provokes the premature selection of FSMs with the highest possible number of states yielding to non-optimal solutions. To minimize the early selection of large FSMs, a "selection restriction" procedure was implemented. According with this procedure, individuals with two or more extra states by report to a previous best-fitted individual were disqualified to the selection step.

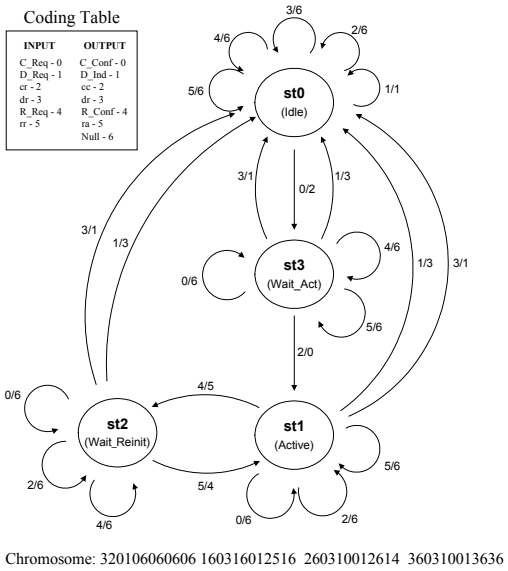


**Fig. 5.** $PS_S$, using STG, of the fittest FSM

Table 1 compares the results of 50 independent runs performed with three different setups. In Setup 1 the fitness function does not take into account the number of states. In Setup 2 the fitness function is weighed by the number of states, according to eq. 1, but does not include the selection restriction procedure. In Setup 3 both a weighed fitness function and the selection restriction procedure are applied. From Table 1 it can be conclude: (a) Without an explicit control in the number of states the system tends to breed a solution using all available states, (b) The control of the number of states coupled with the selection restriction procedure implemented in Setup 3 allow to converge to the minimum FSM in 59% of all successful runs. This represents an improvement of, approximately, 400% by report to Setup 2. (c) The number of runs not yielding to the global optimum in 3,000 generations remained basically the same, regardless of controlling or not the number of states of the FSM.

**Table 1.** Comparison among three different setups for 50 independent runs

| SETUP | Runs not yielding to global optimum in 3,000 generations | Runs yielding to global optimum in 3,000 generations | | |
|---|---|---|---|---|
| | | *4 states* | *5 states* | *6 states* |
| 1. F = $\sum_{i=1}^{N} w_i H_i$ | 20 | 0 | 3 | 27 |
| 2. Eq. (1) | 18 | 4 | 15 | 13 |
| 3. Eq. (1) with selection restriction | 21 | 17 | 9 | 3 |

## 5. CONCLUSION

An evolutionary approach to the design of communication protocols specified by Message Sequence Charts was presented. As in other construction approaches [2, 3] the assumption that a set of MSCs specifies the system behavior *completely* was adopted. However, the methodology relaxes the exact MSC interpretation by adding behaviors not explicitly specified by the MSCs and, contrary to [3], avoids the over-generalization of the synthesized automata.

State-based representation is not efficient for systems with large number of states. However, the benefit of using it in the protocol synthesis is that it guarantees the completeness of the synthesized specifications.

The resulting automata are minimal in many cases, deterministic and completely specified by construction. The application of an analytic method, such as that used in [2], on MSC specification of Fig. 2 yields, in the first steps, a non-deterministic FSM. Moreover, contrary to SCED approach, the proposed approach is not sensitive to the order of traces in its input.

## REFERENCES


[1] Amyot, D., Eberlein, A.: An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development, Telecommunications Systems Journal, 24:1, 61-94 (2003)

[2] Krüger, I. H.: Distributed System Design with Message Sequence Charts, PhD Thesis, Technische Universität München (2000)

[3] Koskimies, K., Männistö, T., Systä, T., Toumi, J.: Automated Support for Modeling OO Software, IEEE Software, 15, 1, pp. 87-94 (1998)

[4] Systä, T.: Static and Dynamic Reverse Engineering Techniques for Java Software Systems, PhD Thesis, University of Tempere (2000)

[5] Fogel, L.: Autonomous Automata, Industrial Research, 4:14-19 (1962)

[6] Manovit, C., Aporntewan, C., Chongstitvatana, P.: Synthesis of Synchronous Sequential Logic Circuits from Partial Input/Output Sequences, ICES'98, pp. 98-105 (1998)

[7] Collins, R., Jefferson, D.: Representation for Artificial Organisms, in Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior, MIT Press (1991)

[8] Chongstitvatana, P., Aporntewan, C.: Improving Correctness of Finite-State Machine Synthesis from Multiple Partial Input/Output Sequences, in Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware, pp. 262-266 (1999)

[9] Bockmann, G., Petrenko, A.: Protocol Testing: A Review of Methods and Relevance for Software Testing, ISSTA'94, ACM, Seattle, U.S.A., pp. 109-124 (1994)

[10] Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press (1992)

[11] Hörner, H.: A C++ Class Library for Genetic Programming, Release 1.0 Operating Instructions, Viena University of Economy (1996)

[12] Probert, R., Saleh, K.: Synthesis of Protocols: Survey and Assessment, IEEE Transactions on Computers, Vol. 40, No 4, pp. 468-476 (1991)

[13] Koskimies, K., Mäkinen, E.: Automatic Synthesis of State Machines from Trace Diagrams, Software Practice and Experience, 24(7), 643-658 (1994)