System Level Hardware–Software Design Exploration with XCS

Fabrizio Ferrandi, Pier Luca Lanzi, and Donatella Sciuto

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32
I-20133 Milano, Italy
{ferrandi,lanzi,sciuto}@elet.polimi.it

Abstract. The current trend in Embedded Systems (ES) design is moving towards the integration of increasingly complex applications on a single chip. An Embedded System has to satisfy both performance constraints and cost limits; it is composed of both dedicated elements, i.e. hardware (HW) components, and programmable units, i.e. software (SW) components, Hardware (HW) and software (SW) components have to interact with each other for accomplishing a specific task. One of the aims of codesign is to support the exploration of the most significant architectural alternatives in terms of decomposition between hardware (HW) and software (SW) components. In this paper, we propose a novel approach to support the exploration of feasible hardware-software (HW-SW) configurations. The approach exploits the learning classifier system XCS both to identify existing relationships among the system components and to support HW-SW partitioning decisions. We validate the approach by applying it to the design of a Digital Sound Spatializer.

1 Introduction

Embedded Systems (ES) design has undergone several deep transformations during the past few years. The most relevant concern is the raising of the abstraction level of the system modeling, made possible by recent evolutions in the description languages. Another trend in the evolution of the design formalisms is the use, as basis for the system design specification, of typical software languages such as C and C++. The introduction of new description languages for the embedded systems applications such as `SystemC`, poses new problems, mainly due to their ability to express system models at levels of abstractions not possible before. The choice of an architecture, is one of the important steps in design. An architecture is defined by a collection of components which can be either programmable (SW components), re-configurable or customized (HW components), Hardware (HW) and software (SW) components have to interact with each other for accomplishing a specific task. The aims of hardware-software (HW-SW) codesign is to support the exploration of the most significant architectural alternatives in terms of decomposition between hardware (HW) and software (SW) components.

© Springer-Verlag Berlin Heidelberg 2004

We present a new methodology based on the learning classifier system XCS [11] to support the HW-SW codesign of embedded systems. The method we propose extract interesting design patterns from raw HW-SW codesign data. These patterns are then exploited by designers to explore the space of the possible partitionings. The method is based on a formal approach to the modeling of the HW-SW codesign problem in which the embedded system is functionally decomposed at the highest possible (functional) level of abstraction. The formal model is used to build a simulator which can be used to obtain an accurate estimate of partitioning costs. The simulator is used to obtain data about the cost function associated to the design of the embedded system. Finally, the learning classifier system XCS is trained with the data obtained by simulator to extract interesting design patterns expressed as condition-action rules which associate cost-patterns to design decisions. While the focus of this paper is mainly the application of XCS to the system level HW-SW design exploration, more details concerning the formal model used to represent the HW-SW design problem can be found in [9]; in addition [3] provides an analysis regarding the use of neural networks to approximate the cost function so to speed up the mining of HW-SW codesign data. The paper is organized as follows. In Section 2 we provide a brief overview of the overall approach we propose. In Section 3 we present a case study involving the design of a Digital Sound Spatializer. In Section 4 we overview the results produced by XCS from the design data. Section 6 ends the paper providing some consideration regarding the results and the future research directions.

2 Proposed Approach

We now shortly overview the codesign approach, we introduced in [9], that we use to generate the data used for the analysis in this paper. We refer the interested reader to the original paper [9] for a detailed description of the overall codesign approach. The Hardware-Software Codesign process is organized into

three main steps that lead to a final partitioning optimized with respect to the given metrics [9]. First, the software performance of the different functional components is estimated from a first description of the application which, nowadays, is usually provided in terms of *C*-like programming languages. Then, the global and local system performance are estimated in terms of hardware description, communication interfaces, and mixed hardware-software architecture. In the final step, the above information is used to guide the algorithm which will provide the optimal partition.

2.1 Software Performance Estimation

This first step aims at giving both a characterization of the timing behavior of the system when specified in some programming language (nowadays `C` and `C++` are the most widely used) and a mean to dynamically determine, via simulation, the best granularity at which the partitioning process is to be faced. The choice of the best granularity, is of paramount importance, as it defines how to decompose the system into sub-parts, which will then be the units over which the final partitioning phase will reason for defining the final system architecture. According to the approach we introduced in [9] this problem is tackled by starting at the *process* level of granularity, which is a sort of meeting in the middle, between the whole system and the operation level. To accomplish this, the target micro-processor is selected and the executable code of the application is produced. Then a parsing tool identifies in the code, by creating a Data Flow Graph (DFG), all the data dependencies to be used for classifying the code fragments, based on which to dynamically identify the mathematical relations that describe execution time as a function of the input data. After having found these mathematical relations, for all the processes involved in the system, the designer, according to the timing system constraints, is able to understand if some of these processes are time critical for the application. For such processes, a finer grain granularity is obtained, by decomposing them in smaller sub-parts, which will be computationally manageable. The outputs of this phase are (i) the identification of the granularity for analyzing the system, based on which all the subsequent phases will be carried on, and, (ii) a characterization in terms of execution time of all the system subparts seen as software modules. This characterization is employed in the final simulation of the system seen as a mixed hardware-software architecture.

2.2 Hardware and Communication Performance Estimation

Given the software performance characterization of the system, and the selected granularity, in this second step we estimate both local and global system performance taking into account the possibility of realizing the different components both in hardware and in software. For gaining a fast estimation on hardware timing, the approach we introduced in [9] relies on the information collected by the parsing tool at the beginning of the software estimation phase. By considering the Data Flow Graph (DFG) of each system component identified by the dynamically determined granularity, we apply to each of them an unoptimized version of an unconstrained scheduling algorithm [6], which gives a characterization of hardware performance. The same approach is applied to model the communication among different classes of components (see [9] for details).

2.3 The Simulator

After having characterized all the components and all the possible communication channels a simulator is built. This is used to simulate several instances of the embedded system architecture, which differ one from the other on the hardware or software characterization of the single sub-parts. Each instance also carries the correct model of communication according to its specific mix of hardware and software parts. The simulator is written in `SystemC` and it is the exact translation of the formal model used to describe the target embedded system. It exploits both the local characterizations given by the previous software and hardware estimation phases and the communication model to compute *global* system performance when instantiating different possible combinations of hardware and software modules. At a very high level of abstraction, the simulator takes as input a binary string representing a partitioning and returns a cost. This mapping is used in [9] to support the search for the final partitioning and in [3] we show how we can successfully approximate the whole map by means of neural networks. In contrast, in the work presented here, we use XCS to analyze the input-output mapping obtained from the simulator so as to identify interesting patterns which characterize the embedded system.

2.4 Exploration of the Partitioning Space with XCS

The simulator can then be used (i) to extract some data regarding the cost of HW-SW configurations and (ii) to mine such data in search of interesting information regarding the system's criticalies for a given class of ES's. For this purpose, we view the ES model, implemented by the simulator, as a black box with inputs $\overline{x}$, representing a certain partitioning and an output y, representing the cost of the (input) partitioning. The model *can be applied* to an example to obtain a cost; but we basically assume that the model is unknown in that we do not know what is the target function f_t which the model implement. For a certain problem a set of examples E is collected from the simulator; each example is a pair $\langle \overline{x}_j, y_j \rangle$, where $\overline{x}_j$ is a possible input configuration (i.e., partitioning), y_j is the cost (performance) that the model predicts for the partitioning $\overline{x}_j$. Starting from the set of examples E, we can apply statistical and data mining techniques to extract interesting relations among input-output configurations. For instance, in [9] we have applied Principal Component Analysis (PCA) and Linear Regression (LR) to extract basic *linear* relations among the different system components; in [3] we focused on the use of neural networks to approximate the cost function from a limited number of examples. Here, we employ a different approach in which the learning classifier system XCS [11] is applied for mining interesting, highly non-linear patterns, from the data obtained from the simulator. For this purpose, XCS is applied as usually done in any other single-step problem. An experiment consists of a number of problems that XCS must solve. For each problem, an input partitioning, represented by a binary string (like those in Table 2) is presented to XCS. Classifier conditions are represented as strings (one for each component) over the ternary alphabet {0,1,#}; the don't care symbol # means that the corresponding position in the classifier condition can either either 0 and 1, i.e., the corresponding component can be implemented either in hardware or software. There is one action for each component identified by the system-level description; for instance, in the example discussed in the previous section, there are nine possible actions, since there are nine possible components. Based on the current input partitioning, XCS suggests an action which identifies a possible modification to the current partitioning through the flip of a bit in the current hardware-software configuration. The action is *performed*, and the current partitioning is modified. As a result, XCS receives a reward computed as the difference between (i) the cost of the current partitioning and (ii) the cost of the new partitioning obtained through the suggested modification. Thus XCS will receive a positive reward if the suggested action corresponds to a decrease in the cost of the partitioning, a negative reward if the suggested action corresponds to an increase in the cost of the partitioning. The performance is measured as the error between the actual reward received as effect of the proposed action, and the reward that XCS estimated. Therefore XCS learns to predict how the modifications of the input partitioning will influence the target cost function. More precisely, since our cost function is simply the estimated execution time, XCS learns to prediction how the modifications of the input partitioning will influence the overall execution time. Note that, according to this settings, XCS is used as a step-wise function approximator as firstly done by Wilson [12] instead of being used to learn an optimal behavior like in [11].

3 Case Study

The application chosen as a case study is a Digital Sound Spatializer, which is significant as it encompasses all the characteristics of a typical embedded system and it is simple enough to allow a straightforward illustration of the methodology. A sound spatializer is a machine, be it analog, digital or hybrid, which takes as its input a sound which is acquired by one or more sound sources: e.g., from a musical instrument or a singer, which is usually recorded with high–quality noise–reduction microphones. Its output is the input sound as it would be perceived by someone in a room, in which the sound is supposed to be played. The code of the application has been written in C++ for giving its software characterization needed for performance estimation and because C++ made it easy to convert it in `SystemC`, which is the language used for the global simulation. The choice of `SystemC` is due to its suitable characteristics both of supporting channels description at a high level of abstraction and because it is undoubtedly the leading specification language for Embedded System design. The target microprocessor we choose for testing the software behavior is the Xilinx MicroBlaze, which is a 32 bit RISC soft-processor. The choice of this specific processor is due to its great suitability for being integrated with the Xilinx Virtex II Pro FPGA, which is the final target architecture for the deployed version of the application. For simulating the acquisition phase of the audio stream and the output of the reverberated sound we used an audio file and sampled it at 44.1 KHz extracting a 16 bits sample; once the sample has been processed by the application we wrote the resulting sample on an output file. The code describing the application is composed of 9 processes, which describe all the different components of the Digital Audio Spatializer: the Multitap Delay line, which simulates both the direct sound and the discrete echoes, the Multitap Delay mixer which scales the output of the Multitap Delay line, the Sound Source, which is one of the processes modeling the room where the sound is diffused, the Walls (which for the final partitioning are considered as 4 different processes), the Listener, who receives the audio sample modified by the four walls, reassembles and scales it and the Output Mixer process, which receives the processed samples from the delay line and from the listener and reassembles it applying to it the final distortion according to the given parameters. The analysis performed on this code shows that, as far as the data dependencies are concerned, the only one involved is the simplest, that is the execution time is *linear* in the number of samples, which is easily understandable if one thinks that the emulation of the room effects on the sound practically are applying a delay and scaling it according to a dispersion factor. According to the results of this phase and the performance needs of the application, which has to process one audio sample per sampling interval, we can state that the granularity of processes is suitable for the overall system, so that we can carry on the subsequent analysis keeping it unchanged. After having found the local information on the software behavior of each process, we turned to the more complex issue of adapting the code to fit in a model allowing the channel insertion with the proper delays and the possibility of instantiating all the possible mixed HW-SW configurations taking into account the previous information on the software performance. To compute the communication delays we referred to a microprocessor frequency f_{sw} of 50 MHz and a hardware frequency of the target $FPGA$ of 100 MHz. We considered a HW-HW communication via dedicated parallel lines. Considering the hardware frequency of 100 MHz and the fact that a complete data exchange operation takes 1 clock cycle to execute, we get: $t_{hh1} = 10\,ns$. For the HW-SW communication model, we cannot derive a single value, as from instance to instance the number of processes communicating via DMA can change; we then inserted the mathematical equation relative to the computation of t_{bus} and t_{dma} in the simulator, so that, once the single instance was given and the corresponding transition labels identified accordingly, we could compute the true delay value. As for the queue parameters other than the N number of customers, we used for their derivation a bus bandwidth of 1 $\frac{Mb}{s}$, obtaining,

$$\mu = \frac{\#bits}{B} = \frac{16b}{10^6 b/s} = 160\,ms.$$

Considering the sample rate of 44.1 KHz, we computed λ (the average interarrival time of the customers), as:

$$\lambda = \frac{1}{44.1KHz} = 227\,ms.$$

Finally, the numerical value of ρ, the traffic intensity, is given by:

$$\rho = \frac{\lambda}{\mu} = \frac{227}{160} = 1.42.$$

The same reasoning applies to the SW-SW communication, as also here the number of processes accessing the shared memory area changes instance by instance, so we inserted in the simulator the mathematical relation for computing t_{lock}, where all the parameters are computed as shown before, with the only difference that, according to the slower frequency of the microprocessor, we used as bandwidth $B = 500\ \frac{Kb}{s}$, so $\mu = 320\,ms$ and $\rho = 0.71$. We show in Table 1 the

Table 1. Local Characterization of the Processes Behavior

Process:	Delay	Del. Mixer	Source	Wall	Listener	Out Mixer
HW ex-time	$2.15 * 10^8$	$5.864 * 10^8$	$2.552 * 10^8$	$1.82 * 10^8$	$7.197 * 10^7$	$6.642 * 10^6$
SW ex-time	$1.0736 * 10^9$	$1.076 * 10^9$	$1.042 * 10^9$	$3.79 * 10^8$	$8.605 * 10^7$	$5.845 * 10^7$

results of the local behavior of the modules both as HW and as SW, where the execution time is expressed in *ns* per *KB* of samples.

After having completed the local characterization, we run a complete simulation of all the possible system instances, which are, according to the 9 different processes we consider, $2^9 = 512$. This simulation took 3 hours and half to complete on an Intel, PIV, 1.7 GHz, 1 GB RAM. This simulation takes into account the proper communication delays by inserting, instance by instance, the correct values in the `WAIT` instructions inserted in the system channels by the simulator. We show a sample output of the simulator in Table 2, where the global execution time is expressed in *ns* per *KB* of samples.

Table 2. Global Characterization of the Application Behavior

System Instance	Global Execution Time
1 1 1 1 0 0 0 1 1	4449011566
1 1 1 1 0 0 1 0 0	4585806280
1 1 1 1 0 0 1 0 1	4634142896
1 1 1 1 0 0 1 1 0	4597492768
1 1 1 1 0 0 1 1 1	4645829384
1 1 1 1 0 1 0 0 0	4585806280
1 1 1 1 0 1 0 0 1	4634142896
1 1 1 1 0 1 0 1 0	4633614640

In Table 2, each entry represents one HW-SW configuration, a one indicates that the process is implemented in software, a zero indicates that the process is implemented in hardware; the order of the processes is: sound source, multitap delay line, delay mixer, wall 1, wall 2, wall 3, wall 4, listener and output mixer.

4 The Results Produced by XCS

We apply the XCS classifier systems to mine interesting patterns from the cost function obtained by applying the codesign approach we described in Section 2 to the Digital Sound Spatializer we illustrated in Section 3. Classifier conditions are strings of 9 symbols (one for each component) over the ternary alphabet {0,1,#}; There are 9 possible classifier actions, numbered from 0 to 8, that represent the change of the corresponding bit in the current input partitioning. The performance of XCS is measured as the error between the actual reward received as effect of the proposed action, and the reward that XCS estimated.

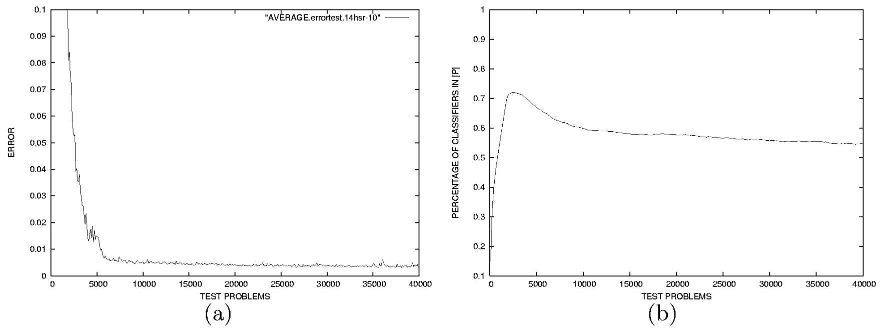

Fig. 1. XCS applied to the cost function for the Digital Sound Spatializer: (a) error on the prediction of the cost; (b) percentage of classifiers in the population. Curves are averages over 10 experiments.

Figure 1a reports the error over XCS prediction for the data derived for the Digital Sound Spatializer; population size N is 5000 classifiers and $\epsilon_0 = 10^{-4}$, while all the other parameters are set as usual (e.g., [11]). As the figure shows, XCS learns to predict quite accurately how the modifications in the current partitioning will affect the cost, i.e., the execution time. Figure 1b reports the percentage of classifiers in the population. Initially, the population rapidly grows while XCS is starting to learn, as the learning proceeds the population size shrinks and the number of classifiers in the population decreases showing that XCS is converging toward a minimal set of classifiers that represent some accurate piece of knowledge extracted from the cost function.
Table 3 reports some classifiers evolved during one of the runs depicted in Figure 1. The first classifier indicates that if component c_2 and component c_3 are implemented in software, then changing component c_2 to hardware (action is 2) will cause a reduction p in the cost (i.e., in the execution time) equal to 0.61, and that this prediction is affected by an absolute error (ϵ) of 1.384×10^{-4}. Note that the second and the third classifiers provide complementary information. The second classifier suggests that if component c_0 and c_1 are implemented in hardware, then changing component c_1 to software will cause *an increase of 0.76* in the cost (we remind the reader that a negative prediction means that the resulting partitioning has an higher cost). Conversely, the third classifier suggests that if component c_0 is implemented in hardware and component c_1 is implemented in software, the changing component c_1 to hardware (so to obtain a partitioning matched by the first classifier), will cause *a decrease of 0.76* in the cost. Note that the classifiers reported in Table 3 represent high level and accurate information about the cost function; all the classifiers in table 3 are

Table 3. Examples of classifiers evolved by XCS from the data obtained for the Digital Sound Spatializer: `Condition` identifies the classifier condition; `Action` identifies the classifier action; p is the classifier prediction which estimates how the partitioning cost will be modified by the corresponding action; ϵ is the prediction error which estimate how much accurate is the prediction p.

`Condition`	`action`	p	ϵ
`##11##########`	2	0.61	1.384e-04
`00############`	1	-0.76	5.520e-03
`01############`	1	0.76	5.292e-03
`11############`	1	0.72	6.658e-05
`0###010#######`	4	-0.15	4.481e-03
`1###100#######`	4	0.16	4.335e-03
`0###110#######`	4	0.16	4.410e-03
`######1#100###`	8	0.15	4.563e-03
`########0#101#`	10	0.16	5.407e-03
`######0#101###`	8	1.52	3.923e-03
...	...	...	...

very general in that they apply to many partitioning (i.e., they have many don't care symbols); in addition they are very accurate since their prediction error ϵ is very small if compared to the prediction value. The evolved classifiers can be used to improve the designer understanding of the existing interactions among different system components, either as an effective support to the search of the best partitioning.

5 Related Work

The first significant work dates back to 1993 [4] and it represents one of the first approaches tackling the complete design of embedded applications. The system is described at the behavioral level by means of an appropriate specification language, whose underlying formal model is a Control Data Flow Graph (CDFG) and the chosen granularity is the operation. The operation delays (needed for performance estimation) are provided separately for hardware and software implementations, based on the type of hardware to be used and on the processor used to run the software. The partitioning algorithm focuses the attention on minimizing communication delays and measuring the partition effects on system performance, trying to devise a partition cost function that captures these properties by means of iterative heuristics. In [10] the problem of HW-SW partitioning is tackled by defining closeness metrics, that is a measure of the likelihood that two pieces of the specification should be implemented on the same system component. The objects are addressed at the procedural level of granularity, and the metrics also have been developed with this granularity in mind. The final algorithm used for finding the final partitioned architecture is based on N-way clustering. There are works (e.g., [2]) focused on solving the partitioning problem according to iterative improvement heuristics, basically exploiting the Simulated Annealing and Tabu Search algorithms. In these works partitioning is performed at the loop level of granularity, with the aim of minimizing communication costs and enhancing parallelism. The partitioning algorithm takes into account simulation statistics, information from static analysis of the source specification and cost estimations, focusing on two main statistics: computational load and communication intensity. In [8], a hierarchical evolutionary approach to HW-SW partitioning is presented. The main characteristic of this work is to apply a hierarchical structure and dynamically determine the granularity of tasks and hardware modules to adaptively optimize the solution while keeping the search space as small as possible. Another work managing a flexible granularity in the partitioning phase is [5]. Recently, several approaches perform design exploration based on Genetic Algorithms (GAs) [1], [7], which seem particularly promising when dealing with parametrized systems to be tuned for final deployment. The approach is a combination of two phases: globally the authors use a parameter dependency model of the target parametrized specification to pre-prune non-optimal subspaces, while locally GAs are applied to discover Pareto-optimal configurations representing the range of performance trade-offs obtainable with parameters tuning.

6 Summary and Future Research Directions

We have presented a novel approach to face the design exploration of the architectural solutions of an embedded system specification based on Wilson's XCS. Our approach is based on the system-level decomposition of the target embedded system. The high-level description is exploited to develop a simulator to estimate the performance of possible hardware-software partitionings. The simulator is used to produce data, associating HW-SW partitionings to cost, which can be analyzed either using basic statistical techniques, as done in [9], either using learning classifier systems, as proposed here. In particular, in this paper we have applied XCS to the mining of HW-SW data from a Digital Sound Spatializer. Although the application is limited in complexity it is significant as (i) it encompasses all the characteristics of a typical embedded system and, most important, (ii) it is a real-world problem provided by a major company interested in the design of embedded system. The results reported here show that XCS can extract accurate rules which identify interesting design information. The results reported here have also been presented to researchers involved in the design of embedded systems for a main company. The comments we received so far are promising. Generally speaking, the rule-based representation of design patterns provided by learning classifier systems appear to be more intuitive than the information provided by the statistical techniques we discussed in [9]. For instance, from our interactions with field experts, we noted that the concept of *generalization* is more easily conveyed by means of rules than it is by using linear models. At the moment, we are applying the same approach on another real-world application involving the design of an embedded mpeg codec component.

References

1. G. Ascia, V. Catania, and M. Palesi. Parameterized system design based on genetic algorithms. In *Proceedings of the Ninth International Workshop on Hardware/Software Codesign*, April 2001.
2. P. Eles, K. Kuchcinski, Z. Peng, and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Journal on Design Automation for Embedded Systems*, 2:5–32, 1997.
3. Fabrizio Ferrandi, Pier Luca Lanzi, and Donatella Sciuto. Mining Interesting Patterns from Hardware-Software Codesign Data with the Learning Classifier System XCS. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, pages 1486–1492, Canberra, Australia, 9-12 December 2003. IEEE.
4. R.K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *Design & Test of Computers, IEEE*, 10:29–41, 1993.
5. J. Henkel and R. Ernst. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9:273–289, 2001.
6. G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.
7. M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the Tenth International Workshop on Hardware/Software Codesign*, pages 67–72, May 2002.
8. G. Quan, X. Hu, and G. Greenwood. Preference-driven hierarchical hardware/software partitioning. In *International Conference on Computer Design (ICCD '99)*, pages 652–657, 1999.
9. Donatella Sciuto, Fabrizio Ferrandi, Pier Luca Lanzi, and Mara Tanelli. System-level metrics for hardware/software architectural mapping. In *Proceedings of the 2nd IEEE International Workshop on Electronics Design, Test and Applications (DELTA 2004)*, Burswood Resort, Perth, Australia, January 2004.
10. F. Vahid and D.D. Gajski. Closeness metrics for system-level functional partitioning. In *Proceedings EURO-DAC '95 Design Automation Conference with EURO-VHDL*, pages 328–333, September 1995.
11. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
12. Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.