AUTOMATED DESIGN APPROACHES FOR MULTI-DOMAIN DYNAMIC SYSTEMS USING BOND GRAPHS AND GENETIC PROGRAMMING

K Seo[a], J Hu[a], Z Fan[a], ED Goodman[a], RC Rosenberg[b]

[a]Genetic Algorithms Research and Applications Group (GARAGe), Michigan State University, USA
ksseo@egr.msu.edu, hujianju@egr.msu.edu, fanzhun@egr.msu.edu, goodman@egr.msu.edu
[b]Department of Mechanical Engineering, Michigan State University, USA
rosenber@egr.msu.edu

Received: 19/10/2001
Accepted in the final form: 15/01/2002

Abstract. This paper suggests a method for automatically synthesizing designs for multi-domain dynamic systems. The domain of those systems includes mixtures of, for example, electrical, mechanical, hydraulic, pneumatic, and thermal components, making it difficult to design a system to meet specified performance goals with a single design tool. Bond graphs are domain independent, allow free composition, and are efficient for classification and analysis of models, allowing rapid determination of various types of acceptability or feasibility of candidate designs. This can sharply reduce the time needed for analysis of designs that are infeasible or otherwise unattractive. Genetic programming is well recognized as a powerful tool for open-ended search. The combination of these two powerful methods is therefore an appropriate target for a better system for synthesis of complex multi-domain systems. The approach described here evolves new designs (represented as bond graphs) with ever-improving performance, in an iterative loop of synthesis, analysis, and feedback to the synthesis process. As an illustrative example, an eigenvalue design problem is tested for some sample target sets and target clusters of eigenvalues.

Keywords: Multi-Domain Dynamic Systems, Bond Graphs, Genetic Programming, Eigenvalues, Open-Ended Design

1 Introduction

Multi-domain dynamic system design differs from conventional design of electronic circuits, mechanical systems, and fluid power systems in part because of the need to integrate several types of energy behavior as part of the basic design [1]. Multi-domain design is difficult because such systems tend to be complex and most current simulation tools operate over only a single domain. In order to automate design of multi-domain systems, such as mechatronic systems, a new approach is required. The goal of the work reported in this paper is to develop an automated procedure capable of designing mechatronic systems to meet given performance specifications, subject to various constraints. The most difficult aspect of the research is to develop a method that can explore the design space in a topologically open-ended manner, yet find appropriate configurations efficiently enough to be useful. Our approach combines bond graphs for representing the mechatronic system models with genetic programming as a means of exploring the design space. Bond graphs allow us to capture the energy behavior underlying

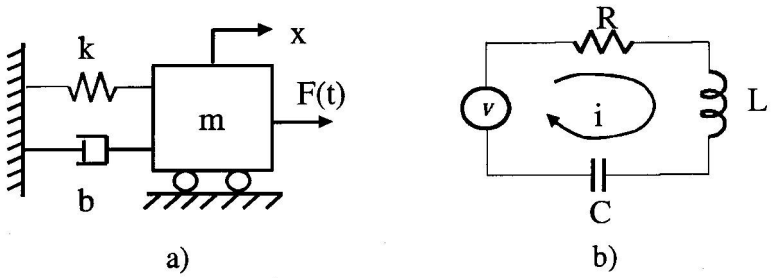

Figure 1: Example single-domain systems: a) mechanical, and b) electrical

the physical aspects (as opposed to the information aspects) of mechatronic systems in a uniformly effective way across domains. Being topological structures, they are also ideal for representing a structured design space for open-ended generation and exploration.

Genetic programming is an effective way to generate design candidates in an open-ended, but statistically structured, manner. A critical aspect of the procedure is a fitness measure, which must guide the evolution of candidate designs toward a suitable result in a reasonable time. There have been a number of research efforts aimed at exploring the combination of genetic programming with physical modeling to find good engineering designs. Perhaps most notable is the work of Koza et al. [2, 3, 4]. He presents a single uniform approach using genetic programming for the automatic synthesis of both the topology and sizing of a suite of various prototypical analog circuits, including low-pass filters, operational amplifiers and controllers. This system has already shown itself to be extremely promising, having produced a number of patentable designs for useful artifacts, and is the most closely related approach to that proposed here; however, it works in a single energy domain.

Our approach, while not nearly as far along in development as Koza's, has two potential advantages. First, it enables the analysis of multi-energy-domain systems with a unified inter-domain tool. Second, it allows efficient and rapid evaluation of individual designs, using a two-stage procedure — causal analysis of the graph followed, only if needed, by appropriate detailed calculation using a derived state model.

As our first class of design problems we chose one in which the objective is to realize a design having a specified set of eigenvalues. Since the problem can be studied effectively using linear components with constant parameters, we only needed to introduce one-port (generalized) resistance, capacitance, and inductance elements in our designs. Section 2 discusses the inter-domain nature, efficient evaluation and graphical generation of bond graphs. Section 3 describes evolution of bond graphs by genetic programming. Section 4 presents some results for an eigenvalue design problem, and Section 5 concludes the paper.

2 Design Domains

2.1 Multi-Domain Dynamic Systems

Multi-domain system design differs from conventional design of electronic circuits, mechanical systems, and fluid power systems in part because of the need to integrate several types of energy behavior as part of the basic design. For example, in addition to appropriate "drivers" (sources), lumped-parameter dynamical mechanical systems models typically include at least masses, springs and dampers (Figure 1a) while "RLC" electric circuits include resistors, inductors and capacitors (Figure 1b)). Figure 2 shows a drive system for a typewriter that involves the drive shaft and the load, with important physical properties modeled. The input is the driving torque, Td., generated through the belt coupling back to the motor.

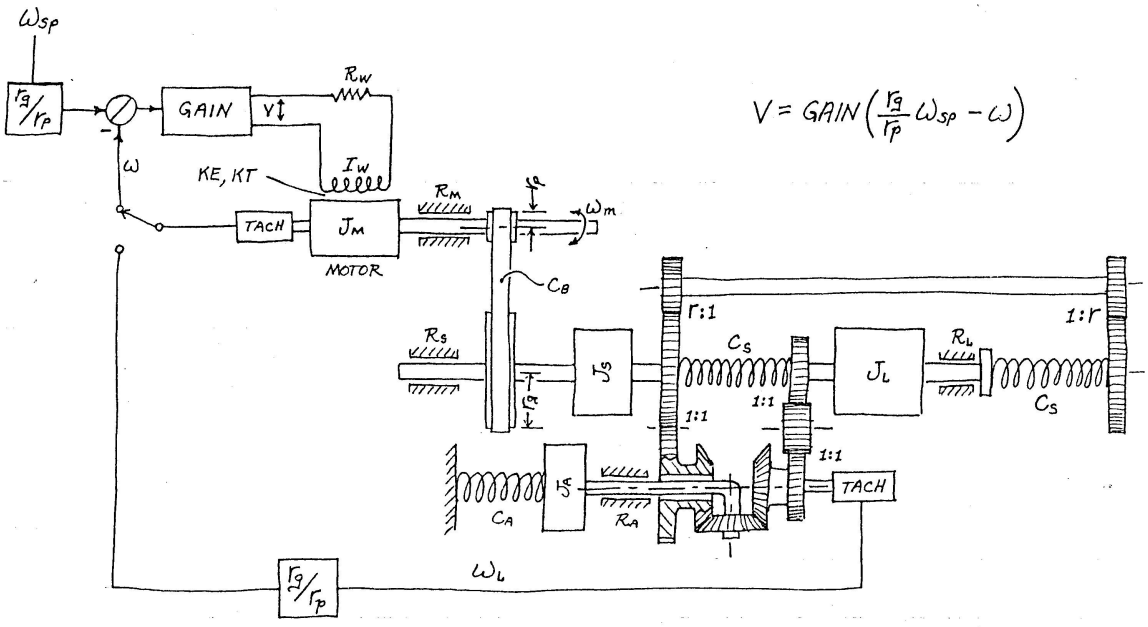

Figure 2: Schematic diagram of an example mechatronic system — the ball drive of an electric typewriter

2.2 Bond Graphs

The bond graph is a modeling tool that provides a unified approach to the modeling and analysis of dynamic systems. Bond graph models can describe the dynamic behavior of physical systems by the connection of idealized lumped elements based on the principle of conservation of power. These models provide very useful insights into the structures of dynamic systems [5, 6, 7, 8, 9]. Much recent research has explored bond graphs as tools for design [10, 11, 12, 13].

Bond graphs consist of elements and bonds. There are several types of elements, each of which performs analogous roles across energy domains. The first type — C, I, and R elements — are passive one-port elements that contain no sources of power, and represent capacitors, inductors, and resistors (in the electrical domain). The second type, S_e and S_f, are active one-port elements that are sources of power, and that represent effort sources and flow sources, respectively (for example, sources of voltage or current, respectively, in the electrical domain). The third type, TF and GY, are two-port elements, and represent transformers and gyrators, respectively. Power is conserved in these elements. A fourth type, denoted as 0 and 1 on bond graphs, represents junctions, which are three-port (or more) elements. They served to interconnect other elements into subsystems or system models.

Bond graphs have three embedded strengths for design applications — the wide scope of systems that can be created because of the multi- and inter-domain nature of bond graphs, the efficiency of evaluation of design alternatives, and the natural combinatorial features of bond and node components for generation of design alternatives. First, multi-domain systems (electrical, mechanical, hydraulic, pneumatic, thermal) can be modeled using a common notation, which is especially important for design of mechatronic systems. For example, the mechanical system and the electrical circuit in Figure 1 have the *same* bond graph model (Figure 3). Second, this representation of dynamic systems is also efficient for computational implementation. The evaluation stage is composed of two steps: 1) causality analysis, and, when merited, 2) dynamic simulation. In causality analysis, the causal relationships and power flow among elements and subsystems can reveal various system properties and inherent characteristics that can make the model unacceptable, and therefore make dynamic simulation unnecessary. While the strong typing used in the GP system (see below) will not allow the GP system to formulate "ill-formed" bond graphs, even "well-formed" bond graphs can have causal properties that make it undesirable or unnecessary to derive their state models or to simulate

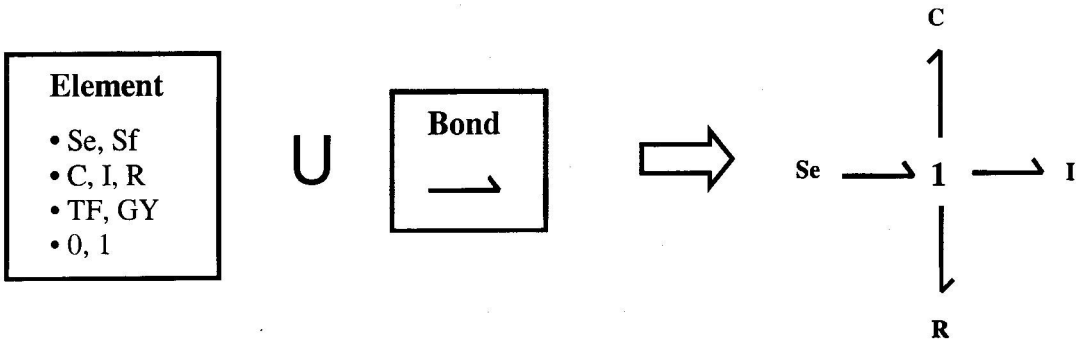

Figure 3: The combinatorial nature of bond graph generation

the dynamics of the systems they represent. Causality analysis is fast, and can rapidly eliminate further cost for many models that are generated by the genetic programming system, by performing assignment of effort and flow variables and making checks for violations of the appropriate constraints. This simple filtering cuts the evaluation workload dramatically. For systems passing causal analysis, state equations are easily and systematically derived from bond graph models. Then various analyses (of eigenvalues, for example) or simulation based on the state model allow computation of the desired performance measures. Third, the graphical (topological) structure characteristic of bond graphs allows their generation by combination of bond and node components, rather than by specification of equations. This means that any system model can be generated by a combination of bond and node components, because of their free composition and unbounded growth capabilities (Figure 3). Therefore it is possible to span a large search space, refining simple designs discovered initially, by adding size and complexity as needed to meet complex requirements.

The particular procedures used for synthesis of bond graph models are a developing and crucial part of this work, since they determine the search space within which design solutions will be contained.

2.3 Combining Bond Graphs with Genetic Programming

Genetic programming is an extension of the genetic algorithm, using evolution to optimize actual computer programs or algorithms to solve some task [14, 15], typically involving a graph-type (or other variable-length) representation. The most common form of genetic programming is due to John Koza [16, 17, 18], and uses trees to represent the entities to be evolved. Genetic programming can manipulate variable-sized strings and can be used to "grow" trees that specify increasingly complex bond graph models, as described below. If the scope and analysis efficiency of the bond graph model can be successfully integrated with the impressive search capability of genetic programming when utilized to its full potential, an extremely capable automated synthesis procedure, without need for user intervention, should result.

As with any fairly general system for design automation, the user must, as part of the specification of the problem to be solved, indicate the target performance that is desired and how it is to be evaluated. That generally includes identifying some input variable(s) or driver(s) and some output(s) at which the desired behavior is to be observed, and the desired relationships among them. For a system to be represented as a bond graph, this amounts to specifying an "embryonic" physical model for the target system, which will remain *invariant* during the design process. That embryo should include any inputs, usually specified as time-varying sources of effort or flow (e.g., voltages, currents, forces, velocities, pressures, etc.). It must include any outputs required to evaluate fitness (for example, voltages across a given load resistance, flow rates through pipes, etc.). That these components should NOT be allowed to be changed/eliminated during design evolution is obvious — the problem is not defined without their presence. When the user has formulated the problem (i.e., the external boundaries of the physical model with its environment and the performance measures

Name	#Args	Description
add_C	4	Add a C element to a junction
add_I	4	Add an I element to a junction
add_R	4	Add an R element to a junction
insert_J0	3	Insert a 0-junction in a bond
insert_J1	3	Insert a 1-junction in a bond
replace_C	2	Replace the current element with a C element
replace_ I	2	Replace the current element with an I element
replace_ R	2	Replace the current element with an R element
+	2	Add two ERCs
-	2	Subtract two ERCs
enda	0	End terminal for add element
endi	0	End terminal for insert junction
endr	0	End terminal for replace element
erc	0	Ephemeral random constant (ERC)

Table 1: GP terminals and functions

to be used), the user must specify it as an embryonic bond graph model and a "fitness" function (objective function to be extremized). The user also specifies one or more "sites" in the embryo model where modifications/insertions are allowed. Then an initial population of trees is created at random, using that embryo as a common starting point. For each tree ("individual"), the bond graph analysis is performed. This analysis, including both causal analysis and (under certain conditions) state equation analysis, results in assignment of a fitness to the individual. Then genetic operations — selection, crossover and mutation — are performed on the evaluated population, generating new individuals (designs) to be evaluated. The loop, including bond graph analysis and GP operation, is iterated until the termination condition is satisfied. The result is a "best" bond graph ready for physical realization.

3 Evolutionary Design with Bond Graphs

3.1 Bond Graph Construction

A typical GP system (like the one used here) evolves GP trees, rather than more general graphs. However, bond graphs can contain loops, so we do not represent the bond graphs directly as our GP "chromosomes". Instead, a GP tree specifies a *construction procedure* for a bond graph. Bond graphs are "grown" by executing the sequence of GP functions specified by the tree, using the bond graph embryo as the starting point.

The initial studies reported here use the following set of bond graph elements: C, I, R; 0, 1, plus any sources in the embryo. This set is sufficient to allow us to achieve designs that have practical meaning in engineering terms, while still permitting other methods to be used for comparison, as an aid in assessment of our work.

We define the GP functions and terminals for bond graph construction as follows. There are four types of functions: first, *add* functions that can be applied only to a junction and which add a C, I, or R element; second, *insert* functions that can be applied to a bond and which insert a 0-junction or 1-junction into the bond; third, *replace* functions that can be applied to a node and which can change the type of element and corresponding parameter values for C, I, or R elements; and fourth, *arithmetic* functions that perform arithmetic operations and can be used to determine the numerical values associated with components (Table 1).

Some typical operations — add_R (a 1-port resistor) and insert_J0 (a 0-junction) — are explained

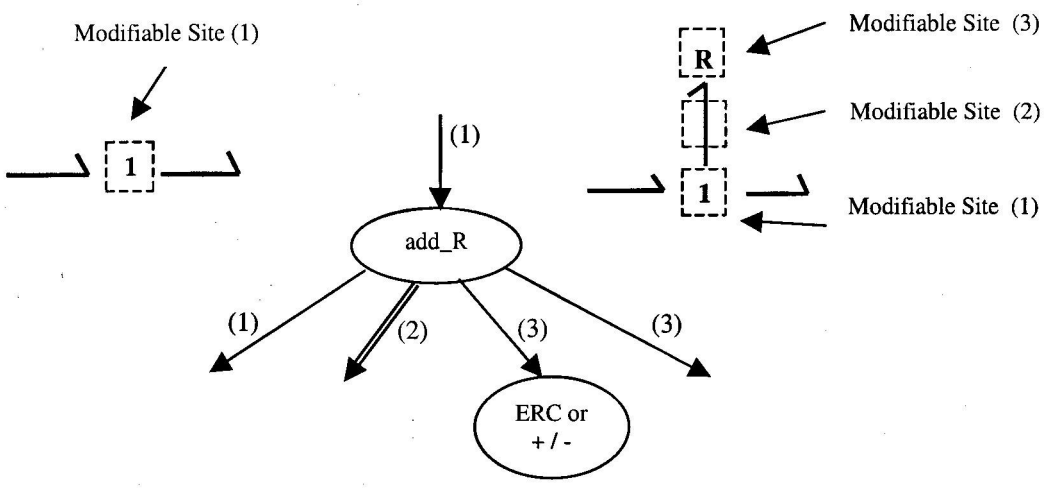

Figure 4: The add_R function

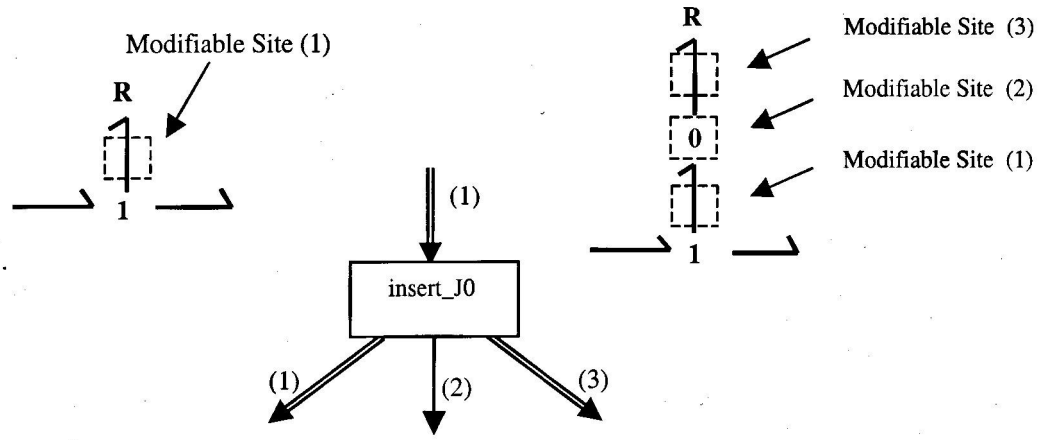

Figure 5: The insert_J0 function

in detail as follows. In Figure 4, the R element is added to an existing junction by the add_R function. This function adds a node with a connecting bond. An R element also requires an additional parameter value (ERC — ephemeral random constant).

The insert_J0 function can be applied only at a bond, and performs insertion of a 0-junction at the given modifiable site (Figure 5). Inserting a 0-junction between node R and a 1-junction yields a new bond graph (the right side of Figure 5). As a result, three new modifiable sites are created in the new bond graph. At each modifiable site, various bond growth functions can be applied, in accordance with its type. In GP terminology, this is a *strongly typed* GP.

3.2 Overall Design Procedure and Algorithm

The flow of the entire algorithm is shown in Figure 6. The user must specify the embryonic physical model for the target system (i.e., its interface to the external world, in terms of which the desired performance is specified). That determines an embryonic bond graph model and corresponding embryo (starting) element for a GP tree. From that, an initial population of GP trees is randomly generated. Bond graph analysis is then performed on the bond graph specified by each tree. This analysis consists of two steps — causal analysis and state equation analysis. Based on those two steps, the fitness function is evaluated. For each evaluated and sorted population, genetic operations — selection,

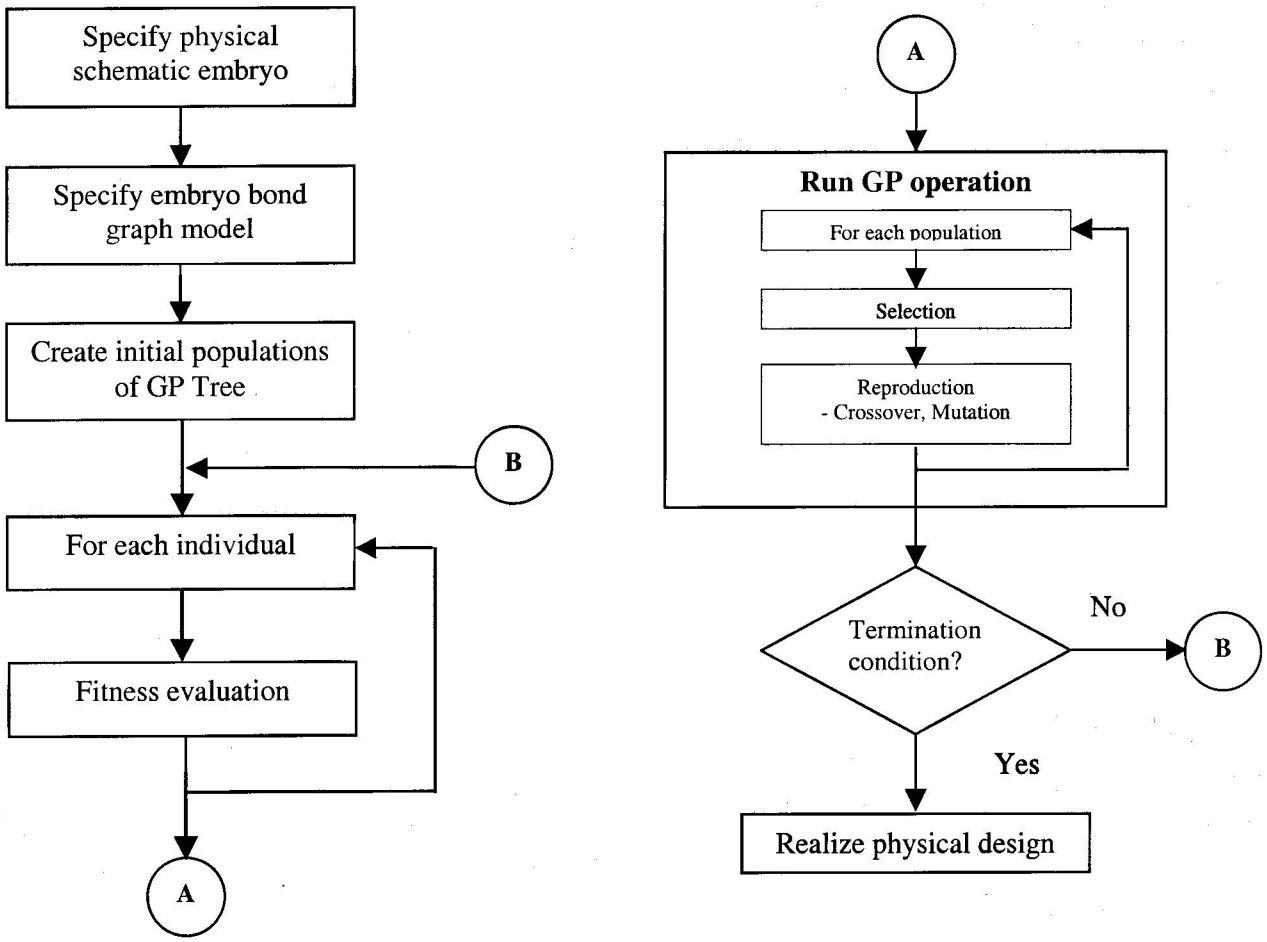

Figure 6: The entire algorithm

crossover and mutation — are performed. This loop of bond graph analysis and GP operation is iterated until a termination condition is satisfied. The final step in instantiating a physical design would be to realize the highest-fitness bond graph in physical components, which is not done in the current work.

3.3 Bond Graph Evaluation

As mentioned earlier, a two-stage evaluation procedure is executed to evaluate bond graph models. The first, causal analysis (see [5]), allows rapid determination of feasibility of candidate designs, thereby sharply reducing the time needed for analysis of designs that are infeasible. For those designs "passing" the causal analysis, the state model is automatically formulated. The evaluation procedure is shown Figure 7.

4 Case Study for Eigenvalue Assignment

Although the final design of practical multi-domain systems still requires physical realization of the best generated bond graph model, it is sufficient to design a bond graph model with the desired performance in order to demonstrate the utility of our automated design methodology for multi-domain systems. In this work, the main design objective is to find bond graph models with minimal distance errors from the target sets of eigenvalues. The problem of eigenvalue assignment has received a great deal

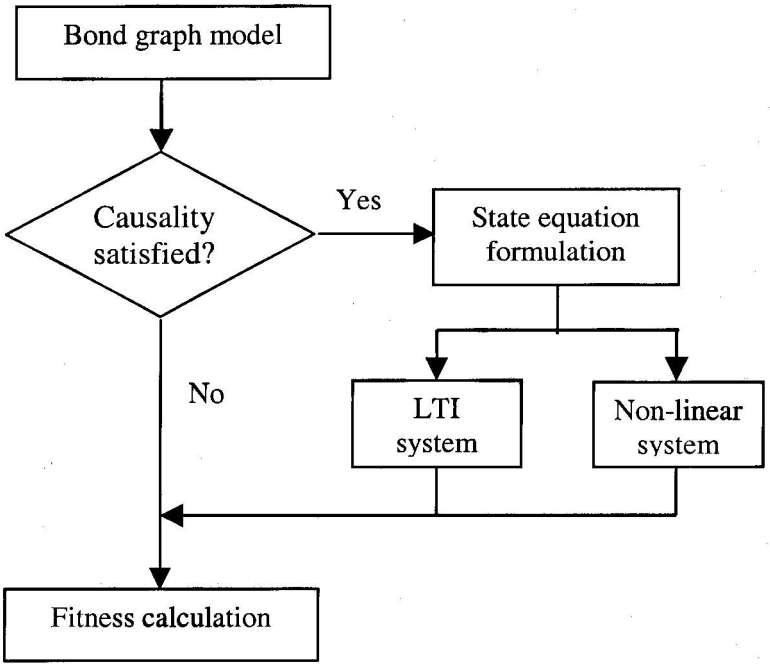

Figure 7: Evaluation of bond graph models

of attention in control system design. Design of systems to avoid instability and to provide response characteristics as determined by their eigenvalues is often an important and practical problem. The following experiments were done to illustrate the performance of genetic programming on this problem and to explore the topological and parametric behaviors of the bond graph models evolved.

4.1 Generating Bond Graphs with Target Sets of Eigenvalues

4.1.1 Problem Definition

In the example that follows, first, a set of target eigenvalues is given and a bond graph model with those eigenvalues is generated. An example of a target set of eigenvalues is shown in Figure 8. The following three sets (consisting of 2, 4, and 6 target eigenvalues, respectively) were used as targets for example genetic programming runs:

$-1 \pm 2j$

$-1 \pm 2j, -2 \pm j$

$-1 \pm 2j, -2 \pm j, -3 \pm 0.5j$

The following sets of experiments (six total) were conducted, with each run repeated 5 times for 6-eigenvalue problems and 10 times each for 2- and 4-eigenvalue problems, all with different random seeds.

1 Each target set using an embryo with one modifiable site

2 Each target set using an embryo with three modifiable sites

The two types of embryo model used are shown in Figure 9. Figure 9a represents an embryo bond graph with one initial modifiable site, while the embryo bond graph in Figure 9b has three initial modifiable sites. Each dotted box represents an initial modifiable site ("writehead"). In each case,

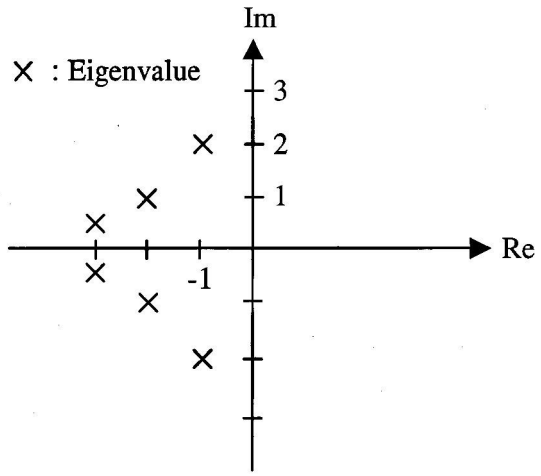

Figure 8: An example of a target set of eigenvalues

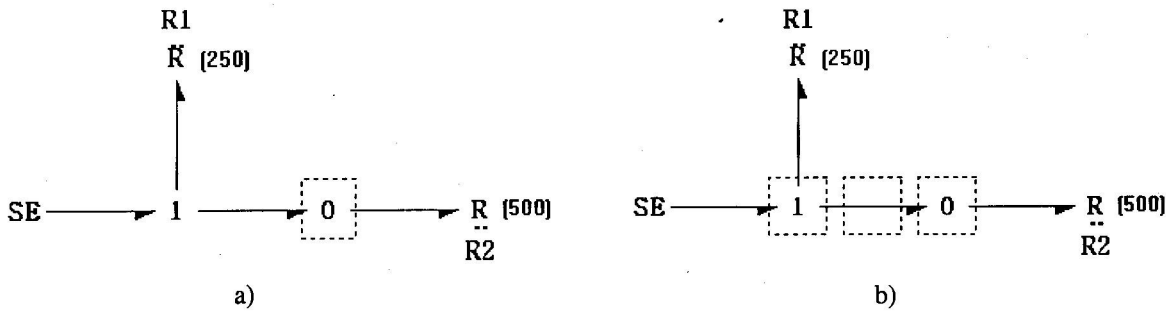

Figure 9: Two types of embryo bond graph model with a) One modifiable site, b) Three modifiable sites

the fixed components of the embryo are sufficient to allow definition of the system input and output, yielding a system for which the eigenvalues can be evaluated, including appropriate impedances. The construction steps specified in the GP tree are executed at that point. The numbers in parentheses represent the parameter values of the elements.

We used a strongly-typed version [19] of lilgp [20] to generate bond graph models. These examples were run on a single Pentium III 800Mhz PC with 256MB RAM. The GP parameters were as shown below. The GP parameters were as shown below, and were selected after a small series of preliminary experiments was conducted over a typical range of settings.

Number of generations:	100 - 500
Population sizes:	100 - 2500 for single population runs for 2 and 4 eigenvalues
	200 in each of 15 subpopulations for multiple population runs for 6 eigenvalues
Initial population:	half_and_half
Max depth:	17
Initial depth:	3–6
Selection:	Tournament (size = 7)
Crossover:	0.9
Mutation:	0.1

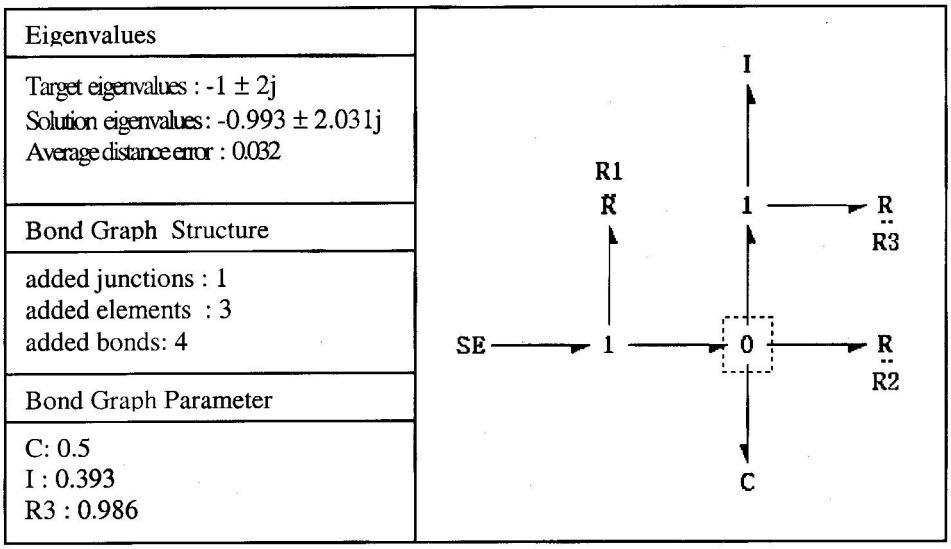

Figure 10: Two eigenvalues result with one modifiable site

4.1.2 Results

Figure 10 gives the solution eigenvalues obtained for a typical run with targets $-1 \pm 2j$, and summarizes the solutions, average distance errors from the targets, structure and parameter information. The corresponding bond graph model obtained is shown on the right side of Figure 10. One 1-junction and three elements (one each of C, I, and R) were added to the embryo bond graph model of Figure 9a in evolution of the solution. This final resulting bond graph is obtained after post-processing to remove unnecessary connections and reduce the non-state-variable R to its simplest equivalent form, using well-established rules. In the other runs, topologically similar structures, with C, I, and R elements and a 1-junction attached to the 0-junction, as in Figure 11, dominated the results.

Figure 11 illustrates solution eigenvalues and a corresponding bond graph model obtained for the same target set $-1 \pm 2j$, but starting from an embryo with three initial modifiable sites (Figure 9b). Nonetheless, in this case, the C, I and R elements all evolved from the third modifiable site in the embryo, because only two state variables were needed. Nine of 10 runs had the same structure as Figure 11. Only in one case were the C, I, and R attached to the 1-junction (first modifiable site).

Figure 12 illustrates the solution eigenvalues obtained for the target set $-1 \pm 2j$, $-2 \pm j$, along with the corresponding bond graph model. One 1-junction and six elements (two each of C, I, and R) evolved from the embryo. Four state variables thus evolved — corresponding to each C or I element. In the 4-eigenvalue problem, the topological search space is larger than in the 2-eigenvalue problem, and a greater variety of structures is discovered. Half of them for the one modifiable site case have very similar structures to that of Figure 12. It is interesting that in 3 of 10 cases, the number of C's and I's in the state vector are not the same (for example, some had one C and three I's). However, in most cases, especially with three modifiable sites, the C's and I's evolved in matched pairs.

Figure 13 illustrates the result for the target set $-1 \pm 2j$, $-2 \pm j$ when started with three modifiable sites. Two C's, two I's and one R element evolved from the 0- and 1-junctions. Unlike the case of one modifiable site, components evolved at three different modifiable sites. In 8 cases of 10, the resulting bond graph appeared more balanced than when evolved from a single modifiable site.

The computation time for the 6-eigenvalues problem was much larger than for the 4-eigenvalue problem, It took 6-8 hours on a Pentium III 800Mhz PC with 256MB RAM for each solution. It was more difficult to achieve acceptable error distances.

Figures 14 and 15 illustrate two typical 6-eigenvalue solutions for the target set $-1 \pm 2j$, $-2 \pm j$,

Eigenvalues
Target eigenvalues : -1 ± 2j Solution eigenvalues: -1.022 ± 2.051j Average distance error : 0.056
Bond Graph Structure
added junctions : 0 added elements : 3 added bonds: 3
Bond Graph Parameters
C: 0.215 I : 0.885 R3 : 24.191

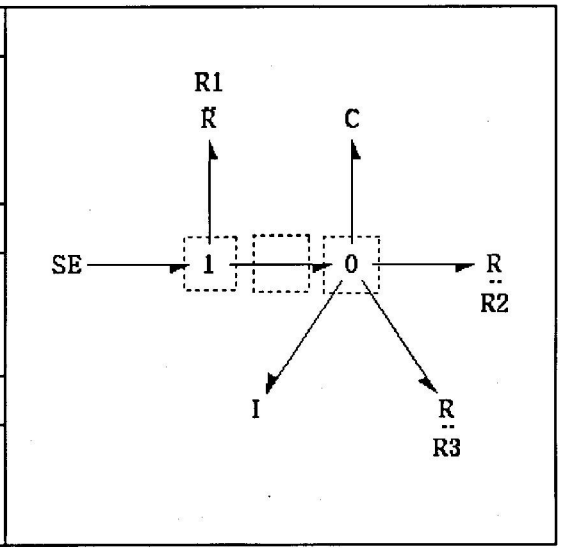

Figure 11: Typical two-eigenvalue result from three modifiable sites

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j Solution eigenvalues: -0.996 ± 1.997j, -1.989 ± 1.001j Average distance error: 0.008
Bond Graph Structure
added junctions : 1 added elements : 6 added bonds: 6
Bond Graph Parameter
C1 : 0.029 C2 : 1.189 I 1 : 0.17 I 2 : 6.822 R3 : 13.179 R4 : 0.209

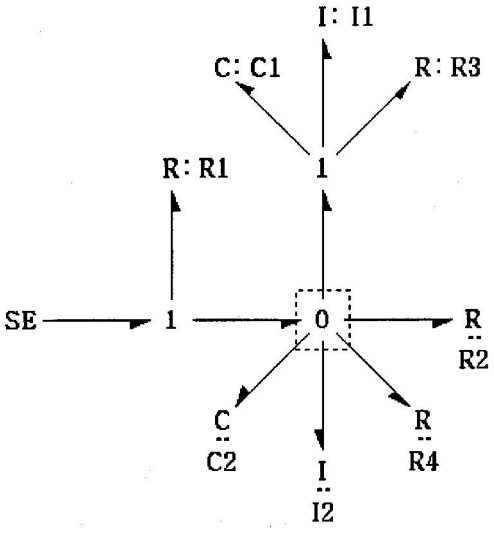

Figure 12: Four-eigenvalue result from one initial modifiable site

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j Solution eigenvalues : -1.028 ± 1.929j, -2.008 ± 0.995j Average distance error : 0.043
Bond Graph Structure
added junctions : 0 added elements : 5 added bonds : 5
Bond Graph Parameters
C1 : 0.0029 C2 : 7.499 I 1 : 121.581 I 2 : 0.027 R3 : 0.033

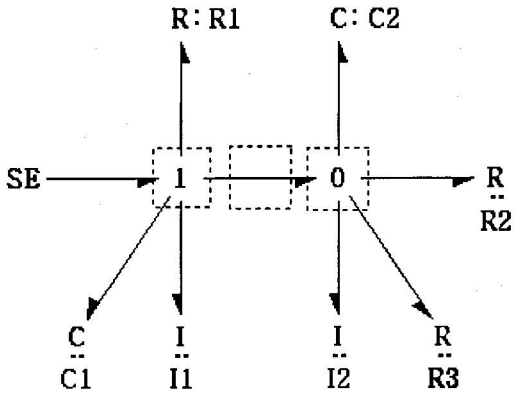

Figure 13: Four-eigenvalue result from three initial modifiable sites

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j , -3±0.5j Solution eigenvalues : -0.998 ± 2.005j, -1.983 ± 0.961j -2.970 ± 0.492j Average distance error : 0.02
Bond Graph Structure
added junctions : 4 added elements : 10 added bonds : 14
Bond Graph Parameter
C1 : 1.072, C2 : 2.702 C3 : 1.23 I 1 : 0.235, I2 : 0.982 I3 : 0.222 R3 : 0.289, R4 : 4.197 R5 : 0.106, R6 : 3.009

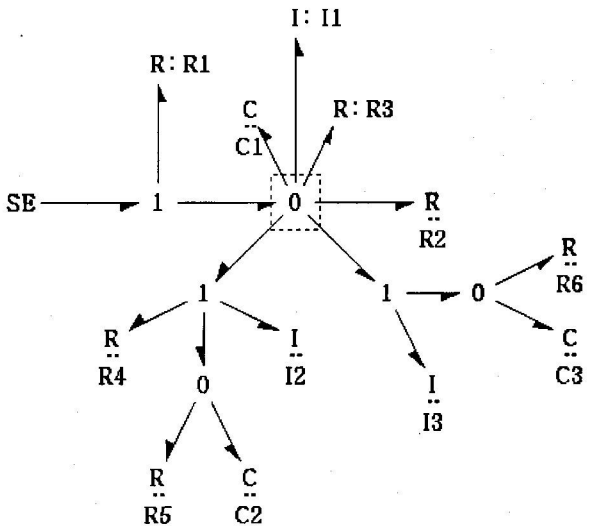

Figure 14: Six-eigenvalue result with one initial modifiable site

Eigenvalues
Target eigenvalues : -1 ± 2j, -2 ± j , -3±0.5j Solution eigenvalues:-1.014 ± 2.012j, -2.053 ± 1.016j -2.998 ± 0.503j Average distance error : 0.18
Bond Graph Structure
added junctions : 3 added elements : 10 added bonds : 13
Bond Graph Parameter
C1 : 0.106, C2 : 0.309 C3 : 0.000 I 1 : 100.574, I2 : 10.0 I3 : 570.908 R3 : 21.39, R4 : 40.393 R5 : 1.116, R6 : 0.002

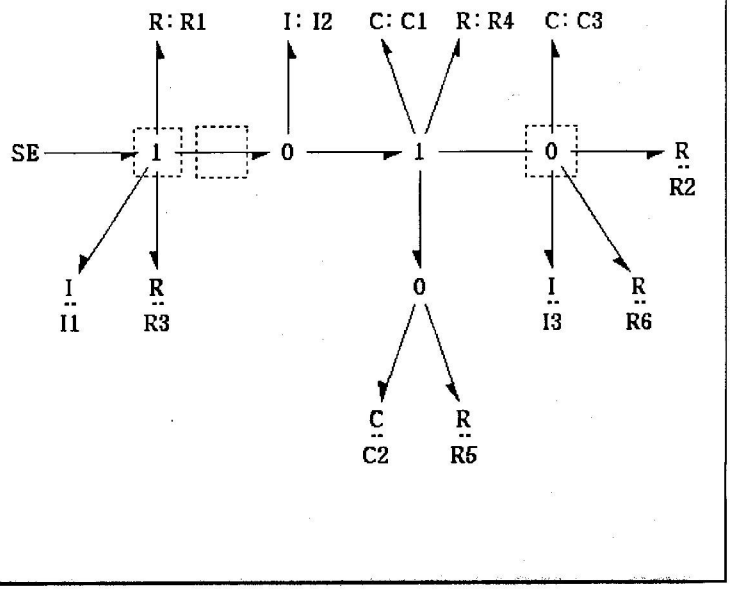

Figure 15: Six-eigenvalue result with three initial modifiable sites

$-3 \pm 0.5j$. Both figures show that more junctions and elements were added, yielding a more complex structure than in the case of the four-eigenvalue problem. The solution starting from three modifiable sites has a more balanced structure, just as was found in the 4-eigenvalue problems, although stronger conclusions must await additional testing.

These runs were produced before the ADF (Automatically Defined Function) facility had been implemented for this strongly typed bond graph representation, but we expect that ADF will allow production of better results for much larger problems, based on the prior work of Koza [2] and the emergence of some useful sub patterns in the bond graphs in our experiments.

The results reported here illustrate two things: 1) Many topological forms of bond graph are capable of satisfying the specified design objectives, and 2) the form of embryo and GP operations used can strongly influence the form of the design evolved. Therefore, care and understanding of the evolutionary process are important in generating designs that maintain desirable topological properties while satisfying the specified design objectives.

Although the experiments run to date are not sufficient to allow making strong statistical assertions, it appears that the search capability of genetic programming is good enough to make feasible the automated design methodology proposed here for multi-domain systems.

4.2 Assigning Eigenvalues in Target Clusters

4.2.1 Problem Definition

A second eigenvalue placement problem, with somewhat different characteristics, has been used for further exploration of the GP/bond graph design capabilities. The objective is to generate a bond graph for which all of the (specified number of) eigenvalues are located in a cluster (with a defined radius) about a prespecified point in the complex number space. For an example, the target cluster was centered around $-2 \pm 2j$, with an allowable radius of 0.5 in Figure 16. The fitness function is maximized if all eigenvalues are located within the two (complex conjugate) clusters.

The result of a typical run, for six eigenvalues, is given in Figure 17. It yielded solution eigenvalues

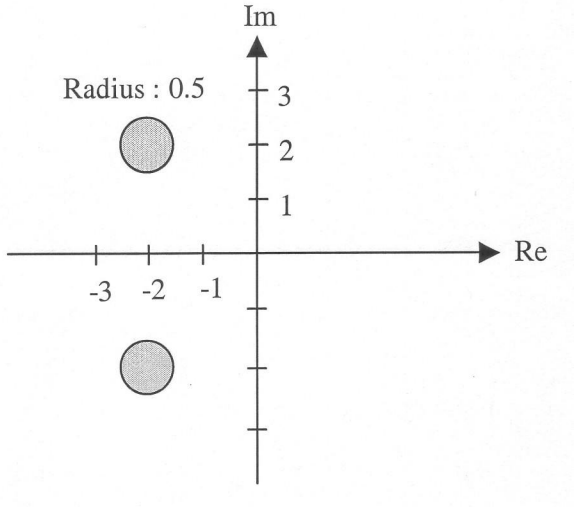

Figure 16: The example of target clusters

Eigenvalues
Target region : -2 ± 2j with radius 1 Number of eigenvalues : 6 Solution eigenvalues : -1.969 ± 2.122j, -1.709 ± 1.678j -1.599 ± 2.238j
Bond Graph Structure
added junctions : 4 added elements : 10 added bonds : 14
Bond Graph Parameter
C1 : 14770.008, C2 : 10.000 C3 : 0.000 I 1 : 0.017, I2 : 0.000 I3 : 7396.610 R3 : 28288.369, R4 : 28.563 R5 : 0.000, R6 : 0.000

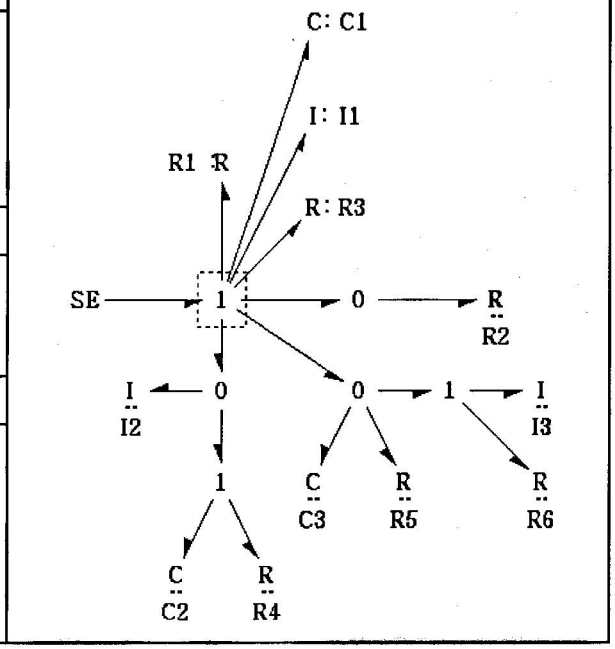

Figure 17: Six-eigenvalue result for target clusters

within the target clusters. In this case, one modifiable site (a 1-junction) was used in the embryo model. The structure of the bond graph model evolved is similar to that evolved in the 6-eigenvalue target set, but it is interesting that some of the parameters of the C, I, and R elements have very large values. Further exploration will be needed to better understand this phenomenon.

5 Conclusion

This paper has suggested a new design methodology for automatically synthesizing designs for multi-domain, lumped parameter dynamic systems — assembled from mixtures of electrical, mechanical, hydraulic, pneumatic, and thermal components. A careful combination of bond graphs and genetic programming, including a multi-step evaluation procedure that greatly increases the efficiency of fitness assessment, appears to be an appropriate approach to development of a method for synthesis of complex multi-domain systems, such as mechatronic systems. As a proof of concept for this approach, evolution of a limited set of bond graphs with specified target eigenvalues was tested, with good results produced rapidly using very limited computing resources. This provides some support for the conjecture that much more complex multi-domain systems with more detailed performance specifications can be automatically designed, given longer execution times and/or using inexpensive cluster computing facilities. Further study will be made as the implementation of the entire set of bond graph constructor functions is completed and appropriate practical design targets are chosen.

Acknowledgments: The authors gratefully acknowledge the support of the National Science Foundation through grant DMI 0084934.

References

[1] E Coelingh, T de Vries, J Amerongen, "Automated performance assessment of mechatronic motion systems during the conceptual design stage," *Proc. 3rd Int. Conf. on Adv. Mechatronics*, Okayama, Japan, 1998.

[2] JR Koza, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Trans. Evolutionary Computation*, vol. 1, no. 2, pp. 109–128, 1997.

[3] JR Koza, D Andre, FH Bennett, MA Keane, "Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier," *Proceedings of the 1997 ACM Symposium on Applied Computing*, San Jose, California, pp. 207–216, 1997.

[4] JR Koza, *et al.*, "Automatic creation of both the topology and parameters for a robust controller by means of genetic programming," *Proceedings of the 1999 IEEE International Symposium on Intelligent Control, Intelligent Systems, and Semiotics*, Piscataway, NJ, pp. 344–352, 1999.

[5] DC Karnopp, RC Rosenberg, DL Margolis *System Dynamics, A Unified Approach*, 3rd Ed., John Wiley & Sons, 2000.

[6] RC Rosenberg, J Whitesell, J Reid, "Extendable simulation software for dynamic systems," *Simulation*, vol. 58, no. 3, pp. 175–183, 1992.

[7] RC Rosenberg, "Reflections on engineering systems and bond graphs," *Trans. ASME J. Dynamic Systems, Measurements and Control*, vol. 115, pp. 242–251, 1993.

[8] RC Rosenberg, MK Hales, M Minor, "Engineering icons for multidisciplinary systems," *Proc. ASME IMECE 1996*, DSC-V.58, pp. 665–672, 1996.

[9] RC Rosenberg, *The ENPORT User's Manual*, Rosencode Associate Inc., E Lansing, MI, 1996.

[10] JE Sharpe, RH Bracewell, "The use of bond graph reasoning for the design of interdisciplinary schemes," *1995 International Conference on Bond Graph Modeling and Simulation*, pp. 116–121, 1995.

[11] E Tay, W Flowers, J Barrus, "Automated generation and analysis of dynamic system designs," *Research in Engineering Design*, vol. 10, pp. 15–29, 1998.

[12] K Youcef-Toumi, YA Glaviano, P Anderson, "Automated zero dynamics derivation from bond graph models," *1999 Int. Conference on Bond Graph Modeling and Simulation*, pp. 39–44, 1999.

[13] RC Redfield, "Bond graphs in dynamic systems designs: Concepts for a continuously variable transmission," *1999 Int. Conference on Bond Graph Modeling and Simulation*, pp. 225–230, 1999.

[14] JH Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[15] D Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[16] JR Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[17] JR Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, The MIT Press, 1994.

[18] JR Koza, FH Bennett, D Andre, MA Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann Publishers, 1999.

[19] S Luke, *Strongly-Typed, Multithreaded C Genetic Programming Kernel*, http://www.cs.umd.edu/users/-seanl/gp/patched-gp/, 1997.

[20] D Zongker, W Punch, *lil-gp 1.1 User's Manual*, Michigan State University, 1996.