Combining Information Technology Components and Symbolic Equation Manipulation in Modeling and Simulation of Mechatronic Systems

Antonio Diaz-Calderon, Christiaan J. J. Paredis, Pradeep K. Khosla
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a hybrid representation for modeling of mechatronic systems. This representation consists of a linear graph and block diagrams and supports our concept of composable simulation. *By composable simulation we mean the ability to automatically generate simulations from individual component models through manipulation of the corresponding physical components in a CAD system. The approach is based on an* augmented system graph *that represents the topology of the system. This graph captures all the interactions between different energy domains (including rigid-body mechanics, electrical, hydraulic, and information technology domains.) This form of virtual prototyping will reduce the design cycle significantly by providing immediate feedback to the designer with minimal intervention of simulation and modeling specialists.*

Keywords

Composable simulation, design verification, linear graphs, software architectures

I. INTRODUCTION

Due to the fierce competition in the current global economy, it is critical for successful companies to react quickly to changing trends in the market place: new technologies, changes in customer demands, fluctuations in the cost of basic materials and commodities, etc. Because design is such an important component in the development of new products, reduced design cycle time will provide a distinct competitive advantage.

The cost and duration of design is determined to a large extent by the cost and duration of design verification. It is possible, however, to verify a design in a virtual environment, based on functional simulations of the design artifact. Such virtual prototyping has the potential to provide significant reductions in design cycle time and cost, under the assumption, of course, that it is less expensive and time-consuming to create a simulation model than a physical mock-up. However, creating high fidelity simulations for complex mechatronic systems can be quite a challenging task itself. To maximize the benefits of virtual prototyping, it is important that simulations can be created effortlessly and at any stage of the design cycle. We believe that our framework for composable simulation addresses these requirements by integrating simulation tightly with the design environment (i.e. CAD software) and allowing the designer to create the simulations directly with minimal intervention of simulation experts.

In this context, a mechatronic system is a system that spans multiple energy domains (mechanical, electrical, and hydraulic) and includes information technology components (such as control algorithms or sensor pre-processors.) Mechatronic systems are in general represented by a set of symbolic differential-algebraic equations, information technology components are usually represented as computer code.

In admitting the occurrence of information technology components in the system of dynamic equations, we must consider their causality. Descriptions of physical processes are non-causal by nature which means that there are no predefined set of inputs and outputs. Causality is an artifact introduced by the modeler to be able to solve the equations. In contrast, information technology components are causal; an implementation of an algorithm has certain inputs from which the outputs are computed. Causality must be accounted for when dealing with hybrid systems.

To address the composable simulation problems outlined above, we are developing a methodology based on a system graph representation. The system graph is a linear graph which captures the topology of the energy flow in the system. We have extended the system graph to include information technology components by combining the linear graph with block diagrams, resulting in a unified system graph representation. The system graph is used to generate the set of differential-algebraic equations that describe the system behavior. The block-diagrams capture the relationships between information technology components.

The relationship between physical systems and linear graphs was first recognized by Trent [13] and by Brannin [2]. Roe [12] and Koenig [9] apply the theory of linear graphs to the systems theory and provide important results that can be directly related to the two basic laws in circuit theory: Kirchhoff's voltage and current laws. Linear graph theory has been used in the analysis of rigid body dynamics [10] and in the analysis of other engineering systems that include interaction between different energy domains [11]. The system graph approach that we are developing builds on linear graph theory.

Another graph representation is bond graphs [8]. Bond graphs are energy-based system descriptions in which energy elements are connected by energy conserving junction structures. Similar to our approach, bond graphs define a minimal set of generalized elements that can be used to model system behavior across energy domains. Connections between elements are made through power bonds which represent the power flow in the system. Although bond graphs (with appro-

priate extensions) can be used to represent mechatronic systems, we have chosen linear graphs because they can be more easily adapted to model 3D rigid body mechanics, and reflect the topology of the physical system directly, making it easier for non-specialists to create system descriptions.

Composition of simulation models can also be accomplished by combining fundamental building blocks described in a high level object-oriented modeling language [3], [7]. The object-oriented approach facilitates model reuse and simplifies maintenance. Using these modeling languages, software executables can be generated automatically from individual sub-models and the interactions between them.

A different approach to composability of simulation models is presented in [4]. In this approach a software architecture that supports the integration of simulation modules is defined. I this view, composition is achieved by connecting software components through a well defined interface. The arrangement of components defines the system to be executed by the simulation engine.

To compose simulation models directly from 3D mechanical systems, in [5] we present a methodology that derives the system graph of a 3D mechanical system directly from the geometry. This system graph is then used to derive the dynamic equations for the 3D mechanism.

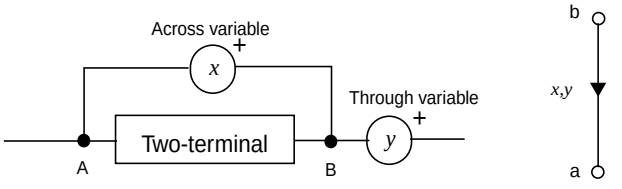
In the following sections we first introduce some general linear graph theoretic concepts. Then we describe the algorithm used to obtain correct simulation code from the hybrid model representation. Next, we briefly describe the integration stage. We then present a short example to demonstrate the applicability of this technique, and we conclude with some observations about the merits and difficulties associated with this method.

II. Graph-theoretic modeling

Lumped physical systems can be described by a collection of interconnected generalized elements. The topological properties of such arrangement can be captured in a system graph to which we associate two terminal variables.

Between any two terminals in the physical component, a pair of oriented measurements can be taken, namely *across* and *through* measurements, as shown in Figure 1. The variables associated with this pair of measurements are called *terminal variables*. The mathematical relations between the terminal variables define the component's physical characteristics and are called *terminal equations*.

Figure 1: Through and across measurements on a general two-terminal element and its terminal graph.



The system graph of the component is a directed edge that joins two the terminal points. This graph representation is called *terminal graph* of the component, and the *system graph* is the collection of terminal graphs connected at the appropriate nodes. In a mechatronic system, the system graph may be non-connected, due to the presence of processes in different energy domains.

Based on the type of relationship between the terminal variables, one can distinguish three classes of elements: passive elements (that can be further divided into dissipative and non-dissipative elements), generators, and transducers. A dissipative element is one which cannot supply energy to the system while a non-dissipative element, does not dissipate energy but can store it for later recovery. These elements can be divided in two categories: *delay* elements which store energy by means of their through variables, and *accumulator* elements which store energy by means of their across variables. The second class of components contains the generators or drivers. A driver forces an across or through quantity to follow a prescribed function of time. The third class of elements, the transducers (also referred to as couplers), transmits energy from one part of the system to another. An ideal transducer is a transducer that can neither store nor dissipate energy, i.e., there is no energy loss in the component.

Interactions between different energy domains, cannot be described with a two-terminal element. It is necessary to introduce elements that have more than two terminals — *n-terminal* elements. Within this category we find the transducer elements defined previously. The system graph associated with an n-terminal element will be derived from measurements taken between pairs of terminals. However as is shown by Roe [12], we only need $n-1$ across measurements to completely determine the across variables between any pair of terminals. This number corresponds to the number of branches in a tree selected in the graph: the terminal graph of an *n*-terminal element is the tree **T** of $n-1$ edges connecting the *n* vertices corresponding to the *n* terminals of the system component. To illustrate this case consider the electric transformer (a 3-terminal system component) shown in Figure 2. Two across measurements will completely determine the device giving a terminal graph with two edges.

As is the case with two-terminal elements, the edges in a terminal graph of an *n*-terminal element will be associated with measurements taken between terminal pairs in the physical system.

In summary, there exists an isomorphism between a linear graph and a physical system provided that one can define pairs of across and through variables. For a system composed of *m* subsystems, the *system graph* is the union of all terminal graphs for all the components of the system.

Figure 2: n-terminal component

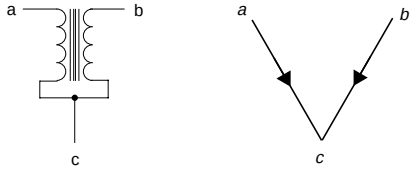


Table 1 shows the terminal variables associated with different energy domains. The derivatives of across or through

variables are across or through variables as well. For example, velocity and acceleration are across variables while the derivatives of force and torque are also through variables.

Table 1: Through and across variables for various energy domains

Type of system	Through Variable		Across variable	
	Name	Symbol	Name	Symbol
General		$x(t)$		$y(t)$
Electrical	Current	$i(t)$	Voltage	$v(t)$
Hydraulic	Fluid flow	$g(t)$	Pressure	$p(t)$
Mechanical	Force, Torque	$f(t)$, $\tau(t)$	Displace-ment	$\mathbf{r}(t)$, $\theta(t)$

A. Constraint equations

We can associate two matrices with the system graph, namely the incidence and circuit matrix. The incidence matrix **A** of a directed graph is **G** with *v* vertices and *e* edges is a $(v-1)\times e$ matrix where the entries can be -1, 1, 0 if the edge is positively, negatively or not incident onto node *v.* Furthermore, if **T** is a tree of the connected graph **G**, the $v-1$ columns of **A** that correspond to the branches of the tree **T** constitute a nonsingular matrix. Thus if a tree is chosen and the columns of **A** are properly arranged, the matrix **A** can be partitioned into the $(v-1)\times(v-1)$ submatrix $\mathbf{A}_T$, referring to the branches of the tree only, and the $(v-1)\times(e-v+1)$ submatrix $\mathbf{A}_C$, referring to the chords of the cotree

$$\mathbf{A} = \left[\mathbf{A}_T\ \mathbf{A}_C\right] \tag{1}$$

The circuit matrix **B** captures the connectivity relations between circuits and edges. If a tree **T** has been chosen, we can arrange the columns of the **B** such that it can be partitioned into the $(e-v+1)\times(v-1)$ submatrix $\mathbf{B}_T$ referring to the branches of the tree and the $(e-v+1)\times(e-v+1)$ submatrix $\mathbf{B}_C$ referring to the chords of the cotree. The chords of the tree **T**, will define a set of *fundamental-circuits* since adding a chord to the tree defines a closed loop. Furthermore, since each chord appears exactly once in any given fundamental circuit in the positive sense, the matrix $\mathbf{B}_C = \mathbf{U}_C$; i.e., a unit matrix. Then we can write

$$\mathbf{B} = \left[\mathbf{B}_T\ \mathbf{U}_C\right] \tag{2}$$

From the *principle of orthogonality* [12] which states that the vector space spanned by the rows of matrix **A** and the vector space spanned by the rows of matrix **B** are orthogonal complements (i.e., $\mathbf{AB}^{\mathsf{T}} = \mathbf{0}$ and $\mathbf{BA}^{\mathsf{T}} = \mathbf{0}$) we can obtain an expression for $\mathbf{B}_T$ as follows:

$$\mathbf{B}_T = -\mathbf{A}_C^{\mathsf{T}}(\mathbf{A}_T^{-1})^{\mathsf{T}} \tag{3}$$

The algebraic properties of the system graph can be stated in the following two theorems [12]:

Theorem I. The oriented sum of through variables associated with the edges incident to a given vertex is zero at any instant of time:

$$\mathbf{Ay} = \mathbf{0} \tag{4}$$

Theorem II. The oriented sum of the across variables associated with the edges in a given circuit is zero at any instant of time.

$$\mathbf{Bx} = \mathbf{0} \tag{5}$$

The $e-v+p$ fundamental circuit equations and the $v-p$ cut-set equations of a system graph **G** with *e* edges, *v* vertices and *p* connected components are referred to as the *constraint equations* of the system (6).

$$\mathbf{x}_C(t) = -\mathbf{B}_T\mathbf{x}_T(t) \quad \text{and} \quad \mathbf{y}_T(t) = -\mathbf{A}_T^{-1}\mathbf{A}_C\mathbf{y}_C(t) \tag{6}$$

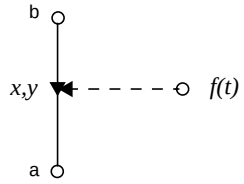
Together these equations form a system of *e* linearly independent equations in $2e$ unknowns. To find a unique solution to this system, we add the *e* independent equations that are derived from the relationships between the across and through variables for the components in the system graph. The equations in this set are called *terminal equations*. In general, for an *n*-terminal component, there will be *n* terminal equations.

Components in a mechatronic system are represented by one or more edges in the system graph connecting well defined interface points. The subset of edges of the system graph that represent a system component is called *terminal graph*.The model of a system component includes both the terminal equations and the associated terminal graph. The terminal graph provides the topological structure of the system component while the terminal equations provide the mathematical model of the basic operation of the system component.

B. Low-power component modeling

In order to include software components as well as other types of low-power devices in the system graph modeling approach, it is necessary to extend our view of the modeling elements presented so far to include the use of signals. A *signal* represents the flow of some system variable value at a very low power level. To introduce signals in the system graph we define the concept of *variable elements*. A variable element accepts an input signal that defines its parameter. The simplest variable element is the signal-controlled across or through driver. In this case, either the across or through variable associated with the terminal graph will be completely defined: $x(t) = f(t)$ or $y(t) = h(t)$. Its terminal graph is shown in Figure 3.

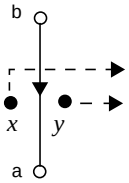
Figure 3: Terminal graph of variable elements.



Similar to a signal-controlled driver, variable passive elements are also signal-controlled, however, the input signal is modulating the parameters defining the element.

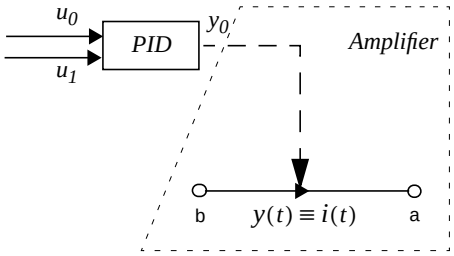
Reading variables from the system graph requires the definition of two abstract nodes. These nodes are used to graphically indicate the interface points between the signal-level components and the high-power components in the system graph. The value for the corresponding signal is derived from the actual values of these variables (Figure 4.)

Figure 4: Reading values from a terminal graph



To illustrate this, consider a current amplifier which is attached to the output of a PID controller (Figure 5). To introduce this element into the system graph, a signal-controlled current source element is used. The output of this element is the through variable $i(t) = y_o(t)$ which is a function of the input signal.

Figure 5: A signal-controlled current amplifier and its terminal graph.



The generation of equations proceeds by considering only the system-graph portion of the entire model. In this way, the algorithms presented in the following section can determine a suitable evaluation order. Once the system equations are derived, the block-diagram nodes are incorporated in the set of equations and a sorting algorithm is used to find a correct computational order of evaluation of the system equations derived from the system graph and the computational blocks specified in the block diagram.

III. Structural analysis

Structural analysis of the system equations is performed in two stages. First, the causality of each element in the system graph is determined. This procedure is called *selection of the normal tree*. Once a normal tree has been selected, the system equations can be derived directly from the tree. The second stage deals with the ordering of the equations and software components into a correct sequence of evaluation.

A. Selection of the normal tree.

The terminal equations plus any independent set of e constraint equations unambiguously define the dynamics of the system. However, before these equations can be numerically solved they must be expressed in state space form in which the derivatives of a state x are expressed as explicit functions of the states and time:

$$\dot{x} = f(x, t) \tag{7}$$

Expressing the equations of the system in this form implies using the smallest possible number of equations (equal to the order of the system) and expressing the high order derivatives as a function of low order derivatives of state variables, in each equation

This can be accomplished in the following way. Let us divide the system variables into two groups: primary variables and secondary variables — one of each for every edge. Assume now that in the terminal equation of an edge, the highest order derivative of the primary variable p is expressed as a function of the secondary variable, s:

$$p^{(n)} = f(s) \tag{8}$$

On the other hand, assume that in the constraint equations the secondary variables are expressed as a function of the primary variables:

$$s = g(p) \tag{9}$$

Then, by substituting the constraint equations (9) into the terminal equations (8), we get a minimal set of dynamic equations of the form (10) which is the desired state-space representation.

$$p^{(n)} = f(g(p)) \tag{10}$$

The final step in the derivation is the selection of the primary and secondary variables. Let **G** be a system graph with e edges, v vertices and p connected components. Let the set of $v - p$ across variables associated with the branches of a forest **F** of **G** and the set of $e - v + p$ through variables associated with the chords of a coforest **F'** of **G** be identified as *primary variables*. Similarly, let the set of complementary variables be identified as s*econdary variables* of the system graph.

Any system graph can have many trees, but of particular interest is the *normal tree*. A normal tree is a tree having the following two properties: 1) the terminal equations for the components with algebraic relations show primary variables as explicit function of secondary variables and or time, and 2)

the terminal equations contain as few derivatives of secondary variables as possible.

The algorithm to find a normal tree in a system graph **G** is based on the fact that it is always possible to find a *spanning tree*[1] in a weighted graph having minimum total cost.

The cost of a graph **G** is the sum of all the edge weights or costs. Each edge may be weighted to model some specific characteristics of the elements in **G**. A *minimum cost spanning tree* is a spanning tree with minimal cost.

The normal tree of a system graph **G** is found by defining a real function w: $e \rightarrow \Re^+$ on the edges of **G** that computes the weight of the edges as follows:

Let κ_α and κ_τ represent the highest derivative order of all accumulator elements and all delay elements respectively, and $O{:}e \rightarrow \Re^+$ be a real function defined on the edges that computes the highest derivative order of the element associated with edge e. Next, classify the edges of **G** as follows: let all across drivers and generalized across drivers belong to the class c^Δ, accumulator elements to the classes

$$c_i^\alpha = \{e_\alpha \,|\, O(e_\alpha) = \kappa_\alpha - i\} \qquad i = 0, \ldots, \kappa_\alpha - 1 \ , \quad (11)$$

dissipator elements to class $c^\delta = \{e_\delta \,|\, O(e_\delta) = 0\}$, and delay elements to the classes

$$c_i^\tau = \{e_\tau \,|\, O(e_\tau) = \kappa_\tau - i\} \qquad i = 0, \ldots, \kappa_\tau - 1 \ . \quad (12)$$

Finally, all through drivers and generalized through drivers will belong to class c^Φ. The weight functions w defined on the edges of **G** are chosen for each class such that

$$\begin{aligned} w^\Delta < w_0^\alpha < w_1^\alpha < \ldots < w_{\kappa_\alpha - 1}^\alpha < w^\delta < w_0^\tau < w_1^\tau \ldots \\ < w_{\kappa_\tau - 1}^\tau < w^\Phi \end{aligned} \quad (13)$$

where w^Δ is the weight function associated to class c^Δ, w_i^α is the weight function associated to class c^α, w_i^δ is the weight function associated to class c^δ, w_i^τ is the weight function associated with class c^τ, and w_i^Φ is the weight function associated to class c^Φ.

In other words, the weight function w ranks the edges of **G** based on their respective classes. Any weight function that satisfies the ranking in equation (13) is admissible.

The next step in our approach is to find a *minimum cost spanning tree* that minimizes the total cost (weight) of the weights assigned to the branches of the tree. Aho et. al.[1] shows that it is always possible to find such a tree based on the following property: let V be the set of vertices of **G** and U be a proper subset of V. If $e_{min} = (u, v)$ is an edge of lowest cost such that $u \in U$ and $v \in V - U$, then there is a minimum cost spanning tree that includes e_{min}. The proof of this property is outside the scope of this article but can be found in Aho et. al.[1]

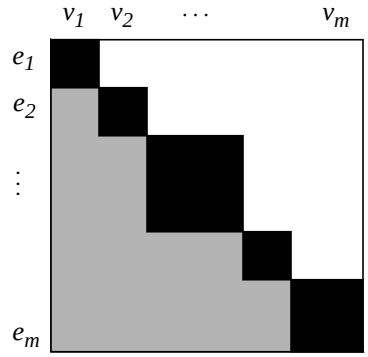
1. A spanning tree for **G** is a free tree that connects all vertices in **G**.

The weight assignment is done only one time and it has order $O(e)$ where e is the number of edges. The minimum-cost spanning tree algorithm is of the order $O(n^2)$ where n is the number of nodes in the graph, if $n \le e$, which is generally the case.

B. BLT form

We seek a form in which the system equations are given as a sequence of blocks of one or more equations[2]. The equations are all first-order differential and algebraic equations. Each block can be solved as a separate problem assuming all previous blocks are solved. Thus, what we seek is a *Block Lower Triangular* (BLT) order of the system equations. The BLT form is a permutation of equations and unknown variables so that the *structural incidence matrix* of the system equations is triangular or block triangular. The incidence matrix is a square matrix where rows represent equations and columns represent unknown variables. It indicates what variables appear in each equation (Figure 6).

Figure 6: Illustration of a BLT form. Black regions indicate that the variable appears in the equation. White areas indicate that the variable does not appear in the equations while gray areas indicate that variables may or may not appear in the equation.

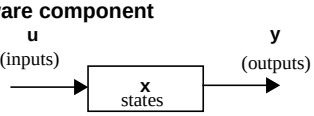


In this form, the state derivatives and the algebraic variables are regarded as unknowns.

The BLT form partitions the set of equations into k blocks of order n_k where n_k is the number of unknown variables to solve for in the subsystem of n_k equations. Equations and software components are sorted such that variables appearing in the equations within each block are either unknowns of the same block or variables solved from previous blocks.

To introduce software components into the incidence matrix we first define two functions associated with it (Figure 7):

Figure 7: Software component



$$\begin{aligned} \dot{\mathbf{x}} &= f_d(t, \mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= f_o(t, \mathbf{x}, \mathbf{u}) \end{aligned} \quad (14)$$

2. We should keep in mind that the system equations may include software components.

That is, a software component will have an operator that computes its outputs, and one that computes its derivatives. The unknown variables of the software component are the inputs **u**. If the output function $\mathbf{y} = f_0(t, \mathbf{x}, \mathbf{u})$ depends on the inputs **u**, the software component is said to be *algebraic*, otherwise, it is said to be *non-algebraic*. This property determines when the software component should be scheduled for evaluation and will be used when we introduce the software component to the incidence matrix.

Before we generate the BLT form, the system equations are augmented with equations of the form:

$$y_i(t) = f_o(t, \mathbf{x}, \mathbf{u}) \tag{15}$$

and all occurrences of references to software components in the system equations are replaced by the y_i variables. In this way every software component is executed at most once in the evaluation of the dynamic equations. The result of this substitution is a system of equations (16) where **v** represents the algebraic variables of the system.

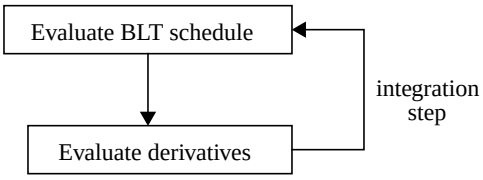
$$\begin{aligned} \dot{x}_i &= f_{1,i}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{y}, \mathbf{v}) \\ v_i &= f_{2,i}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{y}, \mathbf{v}) \\ y_i &= f_{o,i}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}) \end{aligned} \tag{16}$$

Where $\mathbf{u} = \begin{bmatrix} \mathbf{v} & \hat{\mathbf{x}} \end{bmatrix}$ and $\hat{\mathbf{x}}$ is the state vector of the dynamic equations. In this new system of equations, some of the equations are explicit assignments to state derivatives; others are assignments to algebraic variables. The third type of equations are those introduced by the software components. For any equation of this kind, some of the unknown variables will appear as inputs to the function $f_{o,i}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u})$. We call that subset of variables the *dependency set* for the software component. Entries in the incidence matrix indicate whether a variable appears or not in a given equation, however, variables in the dependency set need to be treated in a special way. If the software component is algebraic, the variables in the dependency set are treated as ordinary variables, that is, the equation y_i related to the algebraic software component depends on the inputs. If on the other hand the software component is non-algebraic, the output function $f_{o,i}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u})$ does not depend on the input variables and they are not considered in the incidence matrix. The BLT form orders the output functions of the software components only. The derivative function is evaluated at the integration stage as shown in the following section.

IV. Simulation

Given the correct order of evaluation for the system equations that include software components, we can evaluate them numerically using the following iteration at each major and minor integration step:

Figure 8: Single iteration to evaluate the system equations



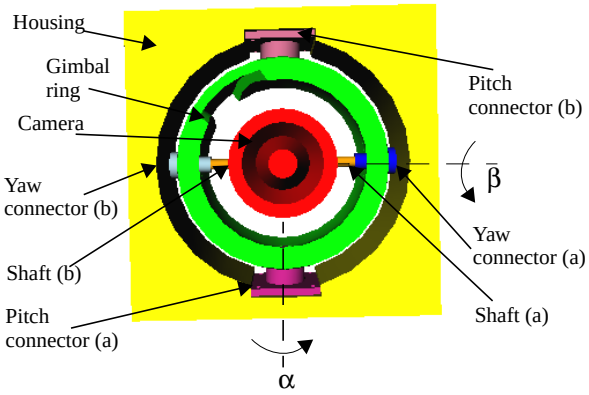
That is, the BLT form provides a correct order of evaluation of equations and software components. We first evaluate the BLT that will compute the unknown derivatives and the values of unknown variables. In doing so, we evaluate the output operators of the software components. Finally, we proceed to evaluate the derivative operators of the software components. Since the variables that are computed by equations together with the output variables of the software components are known, all inputs to the derivative operators for the software components are ready and can be evaluated to complete the evaluation of the vector of derivatives. This is executed every minor/major integration step.

If the simulation kernel supports distributed processing, we may be able to evaluate the output operators of software components given in the BLT form or the derivative operators in parallel on different computers. This provides the ability to run legacy software even in different architectures, in remote computers.

V. Example

To show the concepts presented in this paper we have selected the design of a missile seeker shown in Figure 9.

Figure 9: Missile seeker



The input to the system is a graph defining all components of the device (Figure 10). The connections between components in the design are color coded. Connections in the mechanical domain are represented by edges in orange. Connections in the electrical domain are the blue edges while connections in the signal domain are the edges colored in magenta. To generate the system graph shown in Figure 11 for this design, we use the algorithms presented in [6]. For this system, we obtain a non connected graph where each connected component represents an energy domain. This system graph is then augmented with the components in the signal domain that implement the control system of the device.

These include the PID controllers, the sensors, amplifiers, and reference signals as shown in Figure 10.

Figure 10: Missile seeker input graph.

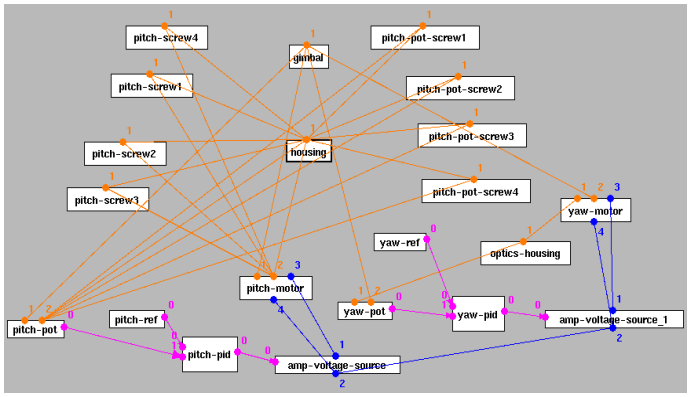
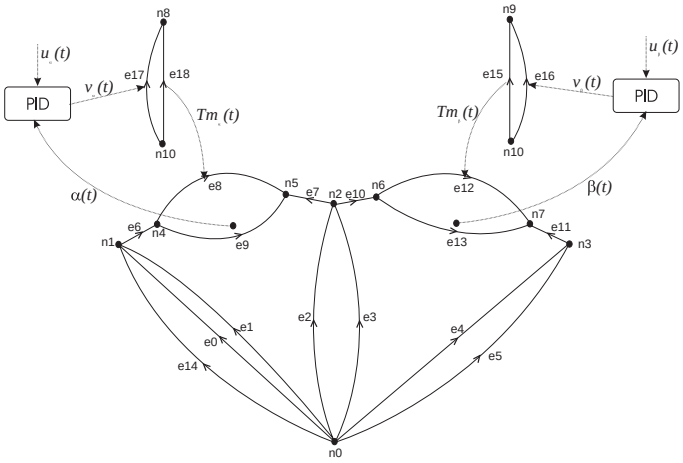


Figure 11: System graph of the seeker design



In Figure 11, the input reference signal to the system is represented by *u(t)* while the sensor is represented by the edge labeled $\beta(t)$. The mechanical system receives input torques that are generated by the transducer represented by *Tm(t)*. The generation of the system equations proceeds in two steps. First the system equations for the non-causal components are generated. Second, the causal relationships derived from the signal domain components are added to the system and the BLT form can be computed to find a computational order of evaluation of the complete set of equations. This ordering also considers the right ordering of the software components since the occurrence of their output functions in the system equations has been accounted for in the BLT.

VI. Acknowledgments

This research was funded in part by DARPA under contract ONR # N00014-96-1-0854, by the National Institute of Standards and Technology, by the Pennsylvania Infrastructure Technology Alliance, by the National Council of Science and Technology of Mexico (CONACyT), and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

VII. References

[1] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, *Data structures and algorithms*. Reading, Massachusetts: Addison-Wesley, 1987.

[2] Branin, F. H. "The algebraic-topological basis for network analogies and the vector calculus," *Symposium on Generalized Networks*, Polytechnic Institute of Brooklyn, 1966.

[3] Cellier, F. E. Automated formula manipulation supports object-oriented continuous-system modeling. *IEEE Control Systems*, Vol. 2, No. 13, April 1993.

[4] Diaz-Calderon A., C. J. J. Paredis, and P. K. Khosla, "A modular composable software architecture for the simulation of mechatronic systems," *ASME Design Engineering Technical Conference*, 18th Computers in Engineering Conference, Atlanta, GA, 1998.

[5] Diaz-Calderon A., C. J. J. Paredis, and P. K. Khosla, "On the synthesis of the system graph for 3D mechanics," American Control Conference, San Diego, CA, June 1999.

[6] A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "Automatic generation of system-level dynamic equations for mechatronic systems," Tech. Report, Institute for Complex Engineered Systems, Carnegie Mellon University, Pittsburgh PA, 1999.

[7] Elmqvist, H., and D. Bruck. "Constructs for object-oriented modeling of hybrid systems." *Eurosim Simulation Congress*, Vienna, Austria, September 1995.

[8] Karnopp, D. C., Margolis, D. L., and R. C. Rosenberg. *System dynamics: A unified approach*. John Wiley & Sons, Inc. New York, NY, 1990.

[9] Koenig, H. E., Tokad, Y., Kesavan, H. K., and H. G. Hedges, Analysis of discrete physical systems. New York: MacGraw-Hill, 1967.

[10] McPhee, J. J., Ishac, M. G., and G. C. Andrews, "Wittenburg's formulation of multibody dynamics equations from a graph-theoretic perspective," *Mechanism and Machine Theory*, vol. 31, pp. 202-213, 1996.

[11] Muegge, B. J. Graph-theoretic modeling and simulation of planar mechatronic systems. MA. Sc. Thesis, Systems Design Engineering Department, University of Waterloo, Waterloo, 1996

[12] Roe, P. H. O'n., *Networks and systems*. Reading, Massachusetts: Addison-Wesley, 1966.

[13] Trent, H. M. "Isomorphisms between oriented linear graphs and lumped physical systems," *The Journal of the Acoustical Society of America*, vol. 27, pp. 500-527, 1955.