Automatic Program Synthesis for Mechanical Systems Dynamic Simulation

Dae Sung Bae†, Tag Gon Kim,†† and Hyoung Jhang††

†Dept of Mechanical Engineering ††Dept of Electrical and Computer Engineering
The University of Kansas
Lawrence, KS 66045

ABSTRACT

Dynamic simulation of a mechanical system is to analyze dynamic motions of the system. In such simulation, a general purpose dynamic program is widely used to formulate the equations of motion with the Cartesian coordinate. The general purpose dynamic program has overhead in computation time and difficulties in maintenance and extension. This paper describes a methodology for automatic program synthesis for dynamic simulation of mechanical systems. The methodology employs system-theoretic representation scheme, which separates structural specification of a simulation model from behavioral one, to synthesize a simulation model in a hierarchical, modular form. A prototype software system implementing the methodology called APSYS is presented along with an example of a tractor system simulation.

1. INTRODUCTION

Dynamic simulation of a mechanical system is to analyze dynamic motions of the system. In such simulation, a set of differential algebraic equations representing the dynamics of a system must be formulated and solved. Since various mechanisms are to be analyzed, the solution algorithms must be flexible enough to add and/or modify modules to adopt new formalisms.

To formulate a set of differential equations of motion for the system, a coordinate system needs to be defined. There are basically two coordinate systems: the Cartesian and relative coordinate systems. The Cartesian coordinate defines a maximum number of generalized coordinates [1]; the relative coordinate defines a minimum number of generalized coordinates, resulting in a smaller dimension of equations [2].

A major advantage of employing the Cartesian coordinate approach in modelling a dynamic system is that it is relatively easy to formulate and implement the equations. Employing the relative coordinate approach has an advantage in computation time, but is more complicated and difficult in formulating the equations. Moreover, modelling within the relative coordinate system is often susceptible to programming errors.

Because of the larger dimensional equations in Cartesian coordinate and a long sequence of conditional statements, the general purpose dynamic program has overhead in computation time. In addition to such overhead, it is difficult to maintain and extend the general purpose program due to its limitations inherited from its implementation languages such as Fortran.

This paper describes a methodology for automatic program synthesis which frees the developer from writing control code into the solution algorithm. The methodology synthesizes a simulation model in a hierarchical, modular form by means of combining structural specification with functional modules for a dynamic system. To do so, we employ system-theoretic representation scheme that separates structural specification of a simulation model from behavioral one [3]. The methodology is similar to that used in [4] which deals with discrete-event systems while this paper deals with continuous systems.

Section 2 describes model formulation. Section 3 presents graph theoretic representation of mechanical systems. Sections 4 and 5 discuss the methodology for automatic synthesis of simulation programs and its prototype implementation called APSYS written in C++, respectively. Section 6 demonstrates the efficiency of our approach through an example of a tractor system simulation within APSYS followed by conclusions in section 7.

2. MODEL FORMULATION

2.1 Relative Coordinate Kinematics

A pair of coupled bodies are shown in Fig. 1, with general relative motion. The x'-y'-z' and x"-y"-z" reference frames are the body and joint reference frames, respectively. Since relative motion between two bodies is represented by relative motion of two joint reference frames, the relative generalized coordinates are defined between two joint reference frames.

Since the velocity equations in the relative coordinate formulation are expressed in terms of the Cartesian generalized coordinates, the Cartesian position must be identified. The centroid of body j in Fig. 1 is isolated by the vector

$$\mathbf{r}_j = \mathbf{r}_i + \mathbf{s}_{ij} + \mathbf{d}_{ij} - \mathbf{s}_{ji} \qquad (1)$$

The orientation matrix of body j is obtained by three sequential transformations starting from the orientation matrix of body i as

$$\mathbf{A}_j = \mathbf{A}_i \, \mathbf{C}_{ij} \, \mathbf{A''}_{ij} \, \mathbf{C}_{ji}^T \qquad (2)$$

where $\mathbf{C}_{ij}$, $\mathbf{A''}_{ij}$, and $\mathbf{C}_{ji}$ are transformation matrices between $x'_i - y'_i - z'_i$ and $x''_{ij} - y''_{ij} - z''_{ij}$, $x''_{ij} - y''_{ij} - z''_{ij}$ and $x''_{ji} - y''_{ji} - z''_{ji}$, and $x'_j - y'_j - z'_j$ and $x''_{ji} - y''_{ji} - z''_{ji}$, respectively.

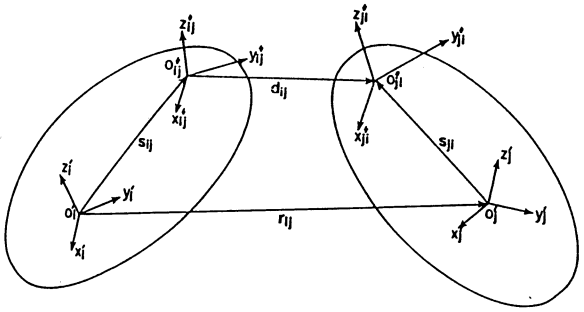


Fig. 1. A Pair of Contiguous Bodies.

The Cartesian velocities are required to perform the acceleration analysis. The Cartesian velocity of a rigid body consists of the translational velocity $\dot{\mathbf{r}}_i$ and the angular velocity ω_i. These velocities may be written in a vector form as $\mathbf{Y}_i = [\dot{\mathbf{r}}_i^T, \omega_i^T]^T$. The Cartesian velocity of body j is obtained by taking differentiations of Eqs. 1 and 2 as

$$\mathbf{Y}_j = \mathbf{B}_{ij1}\mathbf{Y}_i + \mathbf{B}_{ij2}\,\dot{\mathbf{q}}_{ij} \qquad (3)$$

where detailed expressions of $\mathbf{B}_{ij1}$ and $\mathbf{B}_{ij2}$ are defined in Ref. 5.

It must be noticed from Eqs. 1, 2, and 3 that the Cartesian position, orientation, and velocity of body j are expressed in terms of those of body i and the corresponding relative quantities

2.2 Cut Constraint Equations

Geometric characteristics of a cut joint such as parallelism or orthogonality of vectors are used to formulate the cut constraint equations. Since the vectors are expressed in terms of the relative generalized coordinates, the cut constraints may be written as

$$\Phi(q,t) = 0 \qquad (4)$$

Taking the time derivative of Eq. 4 once and twice yields the velocity and acceleration constraints as:

$$\Phi(q,\dot{q},t) = 0 \qquad (5)$$

and

$$\Phi(q,\dot{q},\ddot{q},t) = 0 \qquad (6)$$

2.3 Equations of Motion

The augmented equations of motion were derived in Ref. 2 by using the constraint stabilization method as

$$\begin{bmatrix} \mathbf{M} & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \gamma - 2\alpha\dot{\Phi} - \beta^2\Phi \end{bmatrix}$$

where

$\mathbf{M}$ = Mass matrix
Φ_q = Jacobian matrix for cut joints
λ = Lagrange multiplier vector
$\mathbf{f}$ = Generalized force vector
α and β = Constraint stabilization coefficients

3. GRAPH THEORETIC REPRESENTATION

Graph theoretic representation of a mechanical system is required to systematically derive and implement the dynamic equations [6]. A node and an edge represent a body and a joint, respectively. Topological preprocessing generates a graph that defines the connectivity of mechanical systems. As an example, the graph theoretic representation of a tractor in Fig. 2 is presented in Fig. 3.

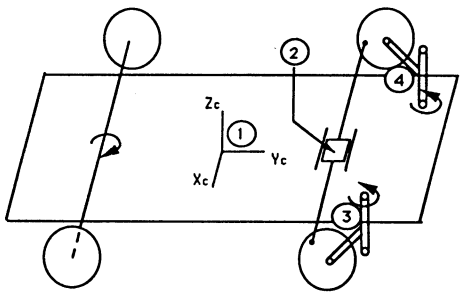


Fig. 2. Schematics of Tractor.

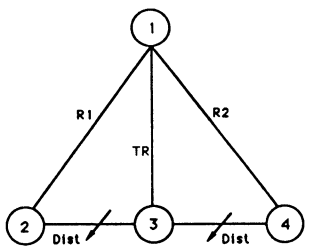


Fig. 3. Graph Representation of Fig. 2.

The cut joint concept was used to handle mechanical systems with closed loops in Refs. 2 and 5. A cut joint is defined within a loop. The cut joint is selected so as to minimize the number of cut constraints and the generalized coordinates [6]. The edges for the cut joints are then removed from a graph. The resulting graph is called a spanning tree structure. The tree structure for the closed

loop system in Fig. 3 is shown in Fig. 4. In order to compensate the effects of cut joints removed, the cut constraint equations are imposed on the equations of motion.

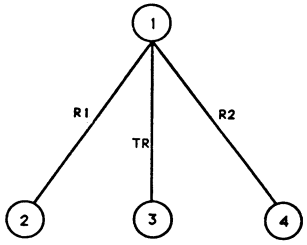


Fig. 4. Spanning Tree of Fig. 3.

4. METHODOLOGY FOR MODEL SYNTHESIS

4.1 Hierarchical, Modular Models

The purpose of our methodology is to automatically synthesize simulation models in a hierarchical, modular form [7]. The hierarchical structure provides a convenient means to modify components and/or subsystems of a system. At the same time, modularity promotes modification and substitution of a subsystem(s) at the various levels of the hierarchy. In such a modular system, no information can be shared between modules for computation. However, the modules exchange information by means of message passings through input/output ports. For example, if a module A has a output port "out" connected to an input port "in" of another module B, then A can send its output to B though A's output port "out" to B's input port "in". Thus, we need to explicitly specify interface between two modules in terms of module names and their connected port pairs.

The methodology for synthesizing simulation models is based on system-theoretic representation of a system. System theoretic representation scheme distinguishes structural specification of a system from behavioral one. Structural specification of a dynamic system represents components of the system and relationships among the components. Behavioral specification, represented by functional subroutines, tells how a component behaves functionally. The main advantage of such separation in specification is that different simulation models can be constructed simply by changing structure of models with behaviors of components unchanged or vice versa.

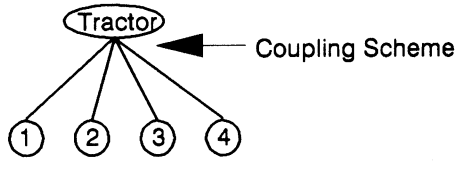
4.2 Augmented Spanning Tree

Structure of a mechanical system can be represented by a spanning tree as shown in Fig. 4. Even though the tree have information on connectivity between two components, it does not have information on which output port(s) of one component is connected to which input port(s) of another component. The interface between output ports of one component and input ports of another is called *coupling scheme* that is necessary to specify message paths for the modular system as described above. For such interface, we shall augment the spanning tree so that coupling scheme may be conveniently associated with edges of the spanning tree. To augment representation of edges that includes coupling scheme, a list of port pairs is added to each edges. For example, an edge (A, B) in a spanning tree is represented by ((A, B) ((out, in))) in an augmented spanning tree, meaning that an output port "out" of a component A is connected to an input port "in" of a component B. Note that we represents coupling scheme as a pair of an edge and a list of pairs of ports. Such coupling specification is required for a modular system in which components of the system communicate each other by the message passings through their input/output ports.

4.3 Hierarchical Augmented Spanning Tree

The augmented spanning tree represents structure of a mechanical system that includes all information components and coupling scheme among each component. Since each node of the tree corresponds to a component of the mechanical system, the tree cannot represent subsystems of the mechanical system explicitly. In other word, no node in the tree represents a subsystem of a mechanical system. Thus, the spanning tree is not suited for manipulation of structure of a mechanical system in a structured manner. To manipulate structure of a mechanical system in a highly structured manner, the augmented spanning tree needs to be represented hierarchically. Such hierarchical representation allows the modeler to add and/or delete components or subsystems of the mechanical system at any level of hierarchy. More specifically, we represent a hierarchical spanning tree in a way that all leaves in the tree correspond to functional modules while internal nodes correspond to subsystems and root node represents the name of a mechanical system. Since an internal node in the tree can have internal children nodes, a subsystem can have children subsystems, each of which, in turn, can have its own subsystems, and so on. Fig. 5 shows an augmented spanning tree representing a tractor system shown in Fig. 4.

Coupling scheme in a hierarchical spanning tree consists of three parts: external input coupling, internal coupling, external output coupling. External input coupling tells how input ports of a component and those of its subcomponents coupled together. Internal coupling says how subcomponents of the component coupled together. External out-



Coupling Scheme = {(Tractor.in, 1.in) (1.out, 2.in)
(1.out, 3.in) (1.out, 4.in) (2.out, Tractor.out1)
(3.out, Tractor.out2) (4.out, Tractor.out3) }

Fig. 5. Hierarchical Spanning Tree of Fig. 4.

put coupling specifies how output ports of subcomponents and a component coupled together.

4.4 Saving Structures and Behaviors

Structural specification representing a hierarchical structure of a mechanical system employs an augmented spanning tree formalism. Behavioral specification employs various functional modules to solve differential equations numerically. Such separated specifications are saved in separated, well-organized libraries for later reuse. We call the library of structural specifications the spanning tree base (STB), and that of behavioral specifications the model base (MB). The STB consists of spanning trees representing different mechanical systems. The MB consists of models representing functional behavior of components of mechanical systems. Each spanning tree in the STB has information on the components of a mechanical system and coupling scheme of the components. Each model in the MB has input port(s), output port(s), a set of state variables, and a functional subroutine.

4.5 Synthesis Procedure

Once a system is specified in terms of its structure and components behaviors, it is ready to synthesize a simulation model. Synthesis procedure in our methodology traverses a spanning tree in preorder and synthesize a simulation model from bottom up. More specifically, when traversing the spanning tree, the procedure retrieves atomic (functional) models in the MB and associates them to corresponding leaves in the spanning tree while creating coupled models and associating them with corresponding internal nodes in the tree. A coupled model so created has information on its children, its parent, its coupling scheme, and others.

To simulate a synthesized model, virtual processors are created and attached to models. The processors has two classes: coordinators attached to coupled models and simulators attached to atomic models. The job of the simulator is to execute a mathematical subroutine associated with the atomic model when it receives input stimuli. The coordinator, when it receives inputs, carries out message communications through input/output ports specified in coupling scheme within the associated coupled model.

5. PROTOTYPING THE METHODOLOGY

APSYS (Automatic Program SYnthesis System) is a software system written in C++ that implements the methodology for automatic program synthesis described in previous section. C++, an object-oriented superset of C language [8], is used to take advantages of object-oriented paradigm such as data encapsulation, information hiding, inheritance, and polymorphism as well as modularity, expendability and reusability.

The APSYS environment facilitates creating model objects and saving them in the model base. The environment also supports specification of augmented hierarchical spanning trees and saving them in the spanning tree base as described in section 4.3. The facility synthesis traverses a hierarchical spanning tree to synthesize a simulation model by retrieving models in the model base as described in section 4.5.

```
class atom : public model {
    int niports;              // # of input ports
    port *iports;             // pointer to port array
    port oport;               // output port
    PF func;                  // work function for this atom
    int count;                // state variable
public:
    atom(char *, model *, int, PF);     // constructorc
    ~atom();                            // destructor
    void setipnmae(char*);              // set input port
    void setopname(char*);              // set output port
    void setoutput(void*);              // set output address
```

Fig. 6. Definition of Class Atom.

Classes in APSYS are organized hierarchically. All classes are derived from the general class models. The class models has a basic definition, which includes its name, parent name, and others, necessary to derive its subclasses. The class models in APSYS has two derived classes: atomic models and coupled models. The coupled models class has a derived class called root models. Thus, the class models inherits all of its properties to the classes of atomic models and coupled models. The coupled models, in turn, inherits both its properties and its inherited properties to the root models class. Fig. 6 shows definition of the atomic models class.

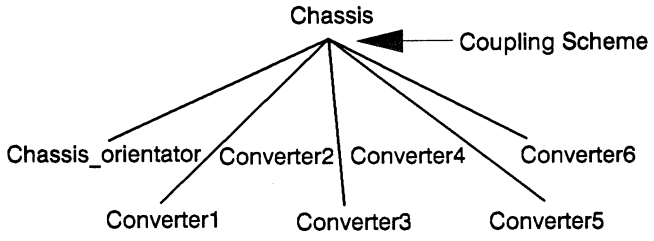
6. TRACTOR SYSTEM: AN EXAMPLE

We demonstrate our APSYS prototype through an example of tracktor system simulation. The example shown here deals with a subsystem of the tractor system called chassis. The example is to show how the simulation model of the chassis subsystem will be synthesized. To do so, we first will specify structure of the chassis including coupling scheme. We will then specify behavior parts of each component of the chassis. Finally, the simulation model of the chassis that APSYS generates will be shown.

The chassis subsystem first computes the orientation matrix of the chassis with known Euler parameters. Six locally defined vectors within the local chassis frame are then converted to the global frame.

The chassis subsystem consists of eight components: one chassis_orientator and six converters. The chassis has eight input ports and eight output ports. Fig. 7 shows the hierarchical spanning tree for the chassis subsystem.

The components of the chassis are defined as objects of the class atomic models. The model chassis_orientator is an object of the class atomic models with the work function of orientation_matrix. Six converters of the class atomic models are defined. They are converter1, coverter2,...., con-




```
Coupling Scheme = {
        (chassis.in1, chassis_orientator.in)
        (chassis.in2, converter1.in2)
        (chassis.in3, converter2.in2)
        (chassis.in4, converter3.in2)
        (chassis.in5, converter4.in2)
        (chassis.in6, converter5.in2)
        (chassis.in7, converter6.in2)
        (chassis_orientator.out, converter1.in1)
        (chassis_orientator.out, converter2.in1)
        (chassis_orientator.out, converter3.in1)
        (chassis_orientator.out, converter4.in1)
        (chassis_orientator.out, converter5.in1)
        (chassis_orientator.out, converter6.in1)
        (converter1.out, chassis.out1)
        (converter2.out, chassis.out2)
        (converter3.out, chassis.out3)
        (converter4.out, chassis.out4)
        (converter5.out, chassis.out5)
        (converter6.out, chassis.out6) }
```

Fig. 7. Spanning Tree for Chassis Subsystem.

verter6, each of which has the work function of convert. Each work function in the objects above is a functional subroutine which carries out its own computation. In APSYS, the atomic model of converter in the chassis is created as:

```
atom converter ("converter", 0, 2, convert);
converter.setipname ("in1");
converter.setipname ("in2");
converter.setopnmae ("out");
```

The above definition tells us the following information: a converter is an object of class atomic models; the work function of the converter is the subroutine convert; the converter has two input ports in1 and in2, and one output port out. Other atomic models can be created in a similar manner.

Given structural and behavioral specifications of the chassis subsystem, the synthesis facility generates a simulation model which basically is a main program for simulation of the chassis. While traversing the spanning tree shown in Fig. 7, the synthesis facility first retrieves all atomic models from the model base, and then creates a couple model called chassis followed by connecting them together. The synthesis facility has successfully generated a main program ready for simulation of the chassis subsystem for the tractor system. Fig. 8 shows part of the main program generated by the synthesis facility. The rest of subsystems for the tractor can be generated in a similar manner.

```
root manager("manager");
coupled chassis("chassis", &manager, 20);
atom chassis_orientator("chassis_orientator", &chassis, 1,
orientation_matrix);
atom converter1("converter1", &chassis, 2, convert);
atom converter2("converter2", &chassis, 2, convert);
atom converter3("converter3", &chassis, 2, convert);
atom converter4("converter4", &chassis, 2, convert);
atom converter5("converter5", &chassis, 2, convert);
atom converter6("converter6", &chassis, 2, convert);

chassis.add(&chassis, "in1", &chassis_orientator, "in")
chassis.add(&chassis, "in2", &converter1, "in2");
chassis.add(&chassis, "in3", &converter2, "in2");
chassis.add(&chassis, "in4", &converter3, "in2");
chassis.add(&chassis, "in5", &converter4, "in2");
chassis.add(&chassis, "in6", &converter5, "in2");
chassis.add(&chassis, "in7", &converter6, "in2");

chassis.add(&chassis_orientator, "out", &converter1,
"in1");
chassis.add(&chassis_orientator, "out", &converter2,
"in1");
chassis.add(&chassis_orientator, "out", &converter3,
"in1");
chassis.add(&chassis_orientator, "out", &converter4,
"in1");
chassis.add(&chassis_orientator, "out", &converter5,
"in1");
chassis.add(&chassis_orientator, "out", &converter6,
"in1");
chassis.add(&converter1, "out", &chassis, "out1");
chassis.add(&converter2, "out", &chassis, "out2");
chassis.add(&converter3, "out", &chassis, "out3");
chassis.add(&converter4, "out", &chassis, "out4");
chassis.add(&converter5, "out", &chassis, "out5");
chassis.add(&converter6, "out", &chassis, "out6");

.....
.....
```

Fig. 8. Part of Synthesized Simulation Program.

7. CONCLUSIONS

A methodology for automatic program synthesis for dynamic simulation of mechanical systems is described. The methodology is based on system-theoretic representation of simulation models that distinguishes models structure from their behavior. The APSYS system implementing the methodology was successfully used for dynamic simulation of a tractor system.

REFERENCES

[1] *DADS User's Manual*, CADSI, P.O. Box 203, Oakdale, Iowa 52319.

[2] Wittenburg, J., *Dynamics of Systems of Rigid Bodies,* B.G. Terbner, Stuattgart, 1977.

[3] Zeigler, B.P., "Knowledge Representation From Newton to Minsky and Beyond," *Applied Artificial Intelligence,* vol.1, pp. 87-107, 1987.

[4] Kim, T.G. and Zeigler, B.P., "A Knowledge-Based Environment for Investigating Multicomputer Architectures," *Information and Software Technology,* vol. 31, no. 10, Dec, 1989.

[5] Bae, D.S. and Haug, E.J., "A Recursive Formulation for Constrained Mechanical System Part II, Closed Loop," *Mechanics of Structures and Machines,* vol. 15, no. 4, 1987.

[6] Kim, S.S. and Vanderploeg, M.J.,"A General and Efficient Method for Dynamic Analysis of Mechanical Systems Using Velocity Transformations," *J. of Mechanisms, Transmissions, and Automation in Design,* 1986.

[7] Zeigler, B.P., *Multifacetted Modelling and Discrete Event Simulation,* Academic Press, 1984.

[8] Stroustrup, B., *The C++ Programming Language*, Additional Wesley, 1984.