GENERIC COMPONENT REPRESENTATION FOR EFFICIENT QUALITATIVE MODEL BUILDING

By

ESTHER L. DAVIS

Bachelor of Arts
Cameron University
Lawton, Oklahoma
1982

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1994

GENERIC COMPONENT REPRESENTATION

FOR EFFICIENT QUALITATIVE

MODEL BUILDING

Thesis Approved:

Blayne E. Mayfield

Thesis Adviser

Huizhu Lu

J Chandler

P M Bacon

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I sincerely thank Dr. Blayne Mayfield for the wealth of energy, inspiration, and time he has given me during the course of my dissertation work. Thank you just doesn't seem adequate to express my gratitude for the many ways in which he has helped me.

I also sincerely thank the other members of my committee: Dr. G.E. Hedrick, for sticking with me even through his sabbatical; Dr. Huizhu Lu, for looking out for me and for inviting me to speak before the ACM which gave me much needed practice for my defense presentation; Dr. Charles Bacon, for providing helpful insights and for staying on my committee after his official retirement. I also thank them for all the time they invested to review the materials and make suggestions to improve the quality of this work.

I sincerely thank my former supervisor, Dr. John Chandler, for hiring me as the Head TA which brought me to Stillwater in the first place; and the Amoco Foundation, for their generous funding of my graduate education.

Above all I want to thank my fiance, Robert Steiner, for all his love, caring and support throughout this project. Without his infinite patience I would have never made it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

One of the primary goals of artificial intelligence has been to analyze and imitate human problem solving techniques. These techniques include reasoning with only general knowledge of facts about objects and how they behave. Qualitative reasoning has become an active area in Artificial Intelligence research in recent years [BO93].

The terms **modeling** and **simulation** are often used as interchangeable terms. Although both are usually present in research that deals with physical system design, diagnosis and prediction, modeling and simulation are distinct, yet strongly connected, concepts. Modeling refers to construction of a representation of a system. The representation may contain attributes of physical objects, relationships between objects and their attributes, as well as global information about the system. Simulation refers to performing time- or action-based propagation of changes to the system to predict its behavior according to the limitations specified in the model [PR74].

Numerical computer simulations of physical systems can be performed provided that precise information is available. Exact numerical simulation produces complex, time-consuming

results which are often too detailed or too large for human interpretation. Artificial intelligence researchers have tried to overcome this drawback by developing qualitative modeling and qualitative simulation algorithms [KU86] [FO84].

Background information on qualitative modeling and qualitative simulation, two segments of qualitative reasoning, is given in Chapter 2. Qualitative simulation is an attempt to duplicate human reasoning about a physical system using basic information about its structure and qualitative attributes to predict possible behaviors. Qualitative simulation can provide important insights during the design phase and can aid in the diagnosis of problems in existing systems [WI90a] [KU90].

In order to make modeling and therefore simulation useful in system design, it is important that all reasonable and pertinent knowledge about the physical system be incorporated without making the model unnecessarily large or complex. Qualitative modeling research has provided useful tools for design and diagnosis of physical systems in many areas, but still has a long way to go. There is a need for better representations of systems, faster and easier model building tools, and integration of quantitative and qualitative information to achieve an appropriate level of detail for optimal modeling. The work done in this paper

focuses on the first two areas, model representation and efficient model building.

Qualitative simulation packages often require users to create models by means of tedious, syntactically specific definitions. Emphasis should be on the overall purpose of the system under design and on the general behaviors of each component in the system. Current representations for qualitative modeling of physical systems do not incorporate all useful knowledge of the components, especially with respect to connection capabilities and terminal variable unifiability as defined later. Since qualitative modeling is rapidly becoming a common method for systems design, diagnosis, and tutoring, it would be advantageous to employ all knowledge of components and their connections that can expedite model building, but without placing additional burden on the user.

A representation that allows flexible, efficient, model building has been developed and is documented in Chapter 3. A generic modeling representation is developed to take the burden off the user, thus creating a more intuitive, faster method of constructing model definitions for qualitative simulation.

The representation developed in this work is based on component-connection type modeling and draws from bond graph theory to form a library of fundamental generic components. Through the use of a graphical user interface, these

fundamental generic components can be combined, connected, specialized, and stored as larger components without requiring the user to understand a particular simulation package syntax. To further assist the user in developing qualitative models for simulation, a graphical user interface has also been developed. Chapter 4 outlines the instructions for use of the interface.

CHAPTER II

REVIEW OF LITERATURE

Qualitative Modeling

Qualitative modeling is designed to model physical systems utilizing incomplete or imprecise information. This incompleteness may stem from an actual lack of knowledge about a particular system or a part of the system, or it may be deliberate in that precise values of system variables are deemed unnecessary for determining the qualitative behavior of the system. Qualitative modeling researchers [WE92] [GR92] [BM93] have sought to find methods for determining the minimum amount of information that must be included in the model to produce acceptable results during simulation.

Qualitative modeling attempts to mimic the way humans view real-world objects and actions by concentrating on general concepts rather than on specific numerical information. Even in systems where extensive numerical formulae are available, qualitative modeling is often chosen as a means of making system modeling simpler. In cases where numerical modeling produces unnecessarily complex or intractable simulations, qualitative modeling

can be used to achieve acceptable simulation results without the time consuming details [FI92].

Qualitative Modeling Ontologies

Three distinct ontologies have been developed in qualitative modeling research. The following sections describe briefly these ontologies.

Process-Centered Ontology. In this ontology the model is described in terms of objects, processes that act on those objects, and influences that directly or indirectly affect the values of object attribute values. Qualitative Process Theory (QPT) has been developed by Forbus [FO84].

Behavior of the model is derived by determining which processes are active at each time point and propagating the influences of those processes across objects in the model. QPT is an alternative approach to qualitative modeling of physical systems but will not be discussed further in this work.

Molecular Ontology. In this ontology the model is described in terms of cells or pixels, and has a structure similar to that of the system being modeled. Molecular ontology is found in neural network simulations of brain activity and simulations of chemical reactions. Molecular models are beyond the scope of this paper.

Device-Centered Ontology. This is also referred to as constraint-based or component-based ontology. The model is described in terms of physical objects, components or devices with attribute variables and constraints which determine the relationships among variables. Device-centered ontology has been developed primarily by Kuipers [KU86] [KU92] and investigated by many others [BM93] [FD91] [LS92].

Behavior of the model under given conditions is derived by propagating constraints over time. The basis for the model representation presented in this work has been developed using concepts from device-centered ontology.

Component Connection Models

Another approach to qualitative modeling based on the constraint- or device-centered ontology is the concept of component-connection models [FD91] [KU92]. Component-connection modeling relies on the construction of a model from distinct components. Each component consists of a set of terminals, variables, and constraints. Terminals, also called ports, represent points at which the component may interact with the outside world. Each terminal has its own set of variables, which describe the attributes through which it may be connected to a compatible terminal of another component.

Connections, indicating how components interact with each other, are defined by specifying links between terminals of different components. In each connection, the set of terminal variables and their associated variable types determine how the interactions between components take place, and how constraints can be propagated throughout the system during simulation. Components cannot interact except through the connections and cannot interact with the system globally in any other way.

Figure 1 shows an example for a simple electrical circuit that can be modeled using component-connection models. The circuit is composed of five components: a battery (B), a switch (S), a capacitor (C), a resistor (R),

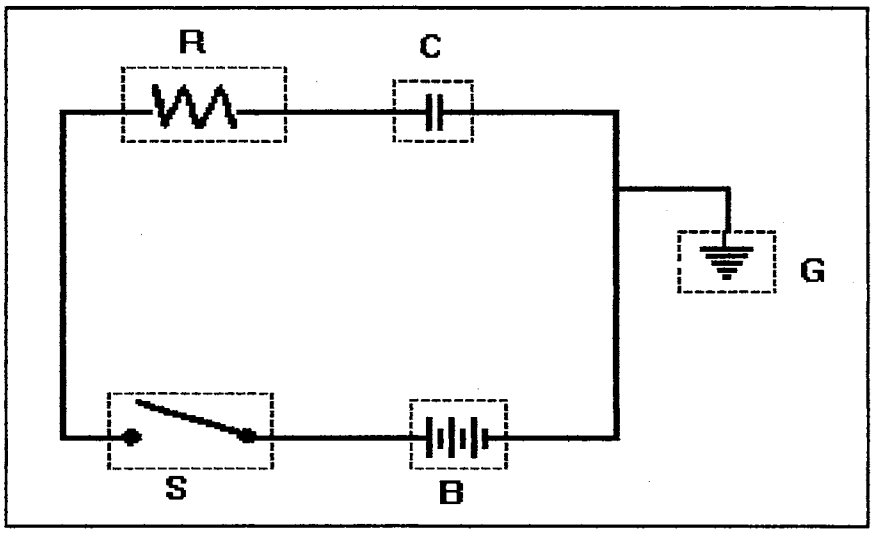


Figure 1. Simple Electrical Circuit

and ground (G), and the associated connections needed to link them together are indicated by solid lines drawn between components.

Component-connection models are highly reusable. Libraries of components for various types of models (for example, a library of electrical components) can be created and saved for later use [FD91] [KU92]. Model building is done by specifying components to be used and the connections between them.

Table I Battery Component Definition

Domain: Electrical

Component Variable		Variable Type
V		Voltage
Terminal	**Variable**	**Variable Type**
T1	V1	Voltage
	I1	Current
T2	V2	Voltage
	I2	Current

Constraints

$V1 - V2 = V$
$I1 = -I2$
$V = V_{bat}$ where V_{bat} is a constant
$V >= \emptyset$

In the example, component B (battery) includes two terminals: T1 and T2, and an additional component variable

which represents voltage across the battery. Each terminal has two variables (voltage and current) associated with it. The sign convention utilized with terminals designates a variable that indicates flow into the component as positive. The Table I illustrates a basic component definition for component B.

Component-connection models provide a less restrictive means of model building in domains where physical systems are clearly composed of individual components. These models have been extensively used in electrical and hydraulic simulations.

Qualitative Simulation

The purpose of qualitative simulation of a physical system is to produce a set of possible behaviors by generating and filtering the set of possible transitions from one qualitative state to another [KU86]. A qualitative state of the system is defined as the collection of all variable (value, direction) pairs where value is a qualitative value of the variable at a given time point and direction represents the direction of value change, i.e. increasing, steady, or decreasing, of that variable. A set of constraints on variables in the system places limitations on the behaviors that can occur.

Qualitative simulation can produce viable answers to "What will happen if...?" questions [KU86] [FO88], even in

systems where certain pieces of information are unknown. For example, in design or diagnosis of a hydraulic system, it may be useful to know "What will happen to the final output pressure if the flow of water at the intake valve is increased?". In a mechanical system design, one of the questions that could be answered by qualitative simulation is "What will happen to the torque on the main shaft if the size of the third gear is reduced?".

The rules governing the progress of a qualitative simulation are those specified in the constraints. No exact functions of variables, no tables of state changes, and no prescribed sequences of events are used. Table II lists

Table II Qualitative Constraint Definitions

Constraint	Equation or Function
(CONSTANT **a**)	the value of **a** is a constant
(EQ **a b**)	**a** = **b**
(ADD **a b c**)	**a** + **b** = **c**
(MULT **a b c**)	**a** * **b** = **c**
(MINUS **a b**)	**a** = -**b**
(M+ **a b**)	**a** is a monotonically increasing function of **b**
(M- **a b**)	**a** is a monotonically decreasing function of **b**
(d/dt **a b**)	**a** is the first-order derivative of **b** with respect to time

some of the constraints that may be used in a qualitative simulation along with the equations they represent.

By propagating the known constraints on variables through the system, qualitative simulation can show the effects of increasing or decreasing the value of a particular variable on the other variables in the system.

Figure 2 shows a simplified drawing of a naval steam propulsion system [FO88]. The system has two primary components: a boiler unit, and a superheater. Water enters the boiler, is converted to steam, which then travels through the superheater.

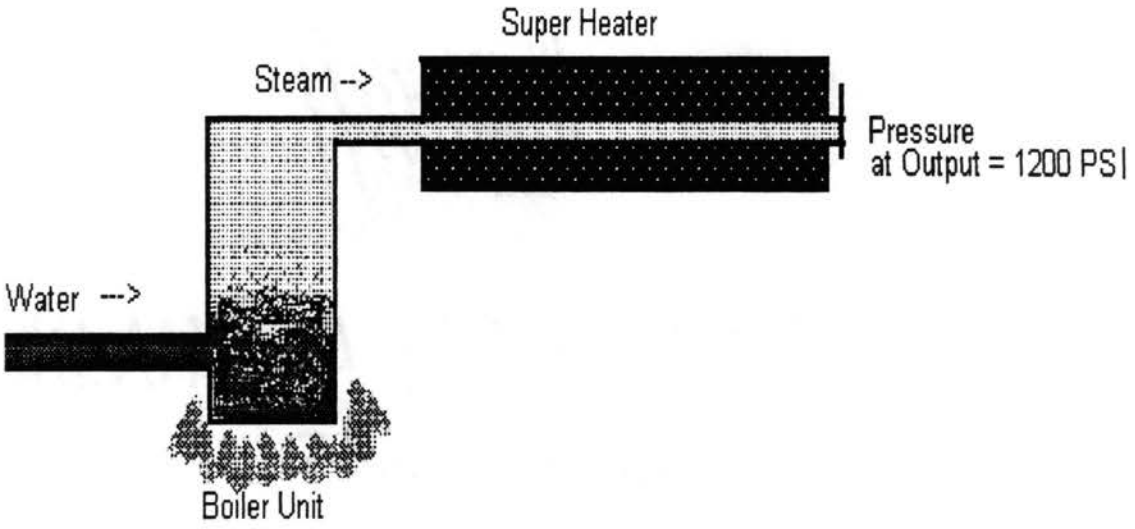


Figure 2. Steam Propulsion System

In training aboard ship, Navy technicians have found answering qualitative questions about the behavior of this type of system very difficult. Qualitative simulation has been used to predict and explain the system's behavior under given conditions.

As an example, suppose the question were "What will happen to the output steam temperature if the temperature of the intake water increases?" This is a legitimate question since the ship may at times sail into warmer parts of the ocean. Naval trainees usually predict an increase in output temperature. However, using the qualitative information and constraints shown in Table III, qualitative simulation shows that the answer is just the opposite.

Table III Qualitative Steam Propulsion Model

Qualitative Information	Constraints
Water is supplied as needed to keep the level in the boiler (LB) constant (C1).	(EQ LB C1) (d/dt EB 0)
Energy supplied by the boiler per unit of time (EB) is a constant (C2).	(EQ EB C2)
Temperature of steam entering the superheater (TSIN) is a constant. It has a quantitative value of 212° F.	(EQ TSIN 212)
The temperature increase (WDIFF) caused by the boiler is a monotonically increasing function of the energy supplied.	(M+ WDIFF EB)

Table III (continued)

Qualitative Information	Constraints
The rate of steam production (RSP) is a monotonically decreasing function of the difference (WDIFF) between the intake temperature (TWIN) and the output boiler temperature (212°).	(M- RSP WDIFF) (ADD WDIFF TWIN 212)
The time the steam takes to travel through the superheater (TSH) is a monotonically decreasing function of the rate of steam production (RSP).	(M- TSH RSP)
The temperature difference between the steam going into the superheater and coming out of the superheater (TDIFF) is a monotonically increasing function of the time it takes to travel through the superheater (TSH).	(M+ TSDIFF TSH) (ADD TDIFF TSIN TOUT)

Given an initial condition (TWIN, TØ, inc) that denotes an increase in the intake water temperature, TWIN, starting from an initial qualitative value of TØ, the simulation propagates constraints and determines that the following sequence of changes will occur in the system:

- Since the intake water temperature is higher, it takes less energy to raise it to boiling.
- The energy needed to raise the water to boiling decreases, but the energy supplied by the boiler is constant, so the rate of steam production will increase.

- As the rate of steam production increases the flow of steam through the superheater increases, thus causing each unit of steam to spend less time in the superheater.
- Since each unit of steam spends less time in the superheater, the temperature difference imparted by the superheater decreases.
- Since the temperature of the steam going in to the superheater is a constant and the temperature difference is decreasing, the output temperature of the superheater decreases.

Thus, through qualitative simulation the answer to the question is shown to be "The output steam temperature **decreases** if the intake water temperature is increased." A possible follow-up question is "What must be done to raise the output steam temperature back to the desired level?". Qualitative simulation could again be used to find the answer.

Landmarks and Quantity Spaces

Instead of using precise numerical values for each variable, qualitative values called **landmarks** are used in qualitative simulation. A landmark represents a value where the variable causes a transition of the system from one qualitative state to another. An ordered set of

landmarks associated with a variable in a qualitative model is called the variable's **quantity space**.

In the Steam Propulsion System example, the intake water temperature (TWIN) has landmark values Ø, TØ, and INFINITY, where TØ represents the initial temperature value. The landmark INFINITY, although unreachable in reality, is included to set a symbolic upper limit on the temperature. The quantity space associated with TWIN would be represented as { Ø TØ INFINITY }.

As another example, a pressure variable (PRES1) in a hydraulic system may have landmarks such as Ø, PMIN, POPT, PMAX, and INFINITY, where PMIN represents the minimum pressure which will produce the desired flow of fluid through the component, POPT the optimal pressure, and PMAX the maximum pressure which can be sustained without bursting the pipe. Again, in reality, the last landmark (INFINITY) cannot be reached. The quantity space associated with PRES1 would be represented as { Ø PMIN POPT PMAX INFINITY }.

Qualitative Differential Equations

An important consideration in qualitative simulation is the use of first-order qualitative derivatives for each variable to determine the direction of change in that variable. Although the set of behaviors produced during the simulation contains all of the possible real behaviors,

it also may contain spurious behaviors, i.e. those which meet specified constraints but cannot occur in the actual system. Inconsistencies detected among first-order and higher-order derivatives during simulation can be used to detect and eliminate spurious behaviors [KC90][KC91].

Qualitative simulation is similar to symbolic simulation in that it can use symbols to represent values of variables without requiring actual numeric values. Qualitative simulation differs from traditional symbolic and numerical simulation methods in several ways, the most important of which is that qualitative simulation can produce a set of possible behaviors in systems where very little information is known. Even in systems where extensive numerical formulae are available, qualitative simulation is often chosen as a means of making the simulation simpler. In cases where numerical simulation produces unnecessarily complex or intractable results, qualitative simulation can be used to obtain useful information without the time consuming details.

Qualitative Simulation Tools

QSIM. Kuipers [KU86][KU93a] and The Qualitative Reasoning Group at the University of Texas at Austin (UTA) have developed a representation based on qualitative differential equations. Using constraint-based ontology, they created an algorithm and an associated implementation

known as the Qualitative Simulation (QSIM) package. QSIM takes as input a lisp-like representation of a physical system that includes variable definitions, quantity space information, and qualitative constraints showing the relationships among the variables critical to the system's behavior. This input must be constructed by the user, with correct syntax, continuity, and complete specification of all variables, constraints, and transitions. From the input and initial perturbation conditions, QSIM produces a set of possible behaviors. A listing of the input required for QSIM to perform the Steam Propulsion System simulation and a sample of the output obtained from the simulation process are given in Appendix A.

QSIM is a well-established program and is based on proven algorithms for qualitative simulation [KU86]. The QSIM algorithm has been shown to provide good results in many branches of physical system modeling, including electrical, hydraulic, chemical, and medical systems. Work is continuing at UTA and other universities to improve the performance of simulations produced by QSIM.

CC. In response to a need for a simpler component-based modeling method, members of The Qualitative Reasoning Group at UTA developed CC [FD91][KU92], a front-end interpreter for QSIM. CC takes as input a set of component and connection specifications, then converts these into

variables, transitions, and constraints that can be used as input for QSIM. This component based format works well for certain domains of modeling, especially electrical and hydraulic systems.

Appendix B shows the CC input needed to create the electrical circuit from Figure 1 as well as the output produced by the CC interpreter. The CC representation format of components and connections has a lisp-like structure. This structure facilitates interpretation of the system specification because the CC to QSIM compiler is written in LISP.

CC allows the use of elemental components (battery, switch, etc.) as well as composed components, i.e. those created by connecting a set of components. The circuit model, once created, could be used as a composed component in a larger system.

Biswas, *et al.* [BM93] have provided a set of extensions to CC that allows the use of global information such as system-wide constraints supplied by the user. These extensions have been tested and shown to be helpful in the use of qualitative models as diagnosis tools.

Crawford, *et al.* [CF90] developed an alternative to CC called the Qualitative Process Compiler (QPC). QPC takes the general approach of Qualitative Process Theory by using a description of a model in terms of views, processes, and influences. The model is compiled into a set of

qualitative differential equations for use in QSIM by identifying active processes and transforming influences into constraints.

Bond Graph Theory

Bond graphs are a means of representing physical systems using a set of basic elements called **multiports** [KA90]. Connections between multiports are termed **bonds**. A bond graph consists of a set of multiports connected by lines or arrows which represent bonds.

Multiports of a bond graph are designed to model the power and energy attributes of components in a physical system; bonds model the interactions that take place among the components. Multiports and bonds do not directly relate to the physical components or connections and no information about sizes or relative locations of components can be obtained from a bond graph.

Power and Energy Variables

Four primary variables are used to describe the behaviors of multiports: effort (e), flow (f), momentum (p), which is the time integral of effort, and displacement (q), which is the time integral of flow. Effort and flow are referred to as the Power Variables, while momentum and displacement are the Energy Variables.

1-Port Elements

Multiport elements are grouped according to the number of ports they have. Each port represents a bond that may be made between the element and another multiport. Elements which consist of a single port (1-ports) are:

Resistor: —R — A resistor element relates the effort variable to the flow variable. It implies that energy is dissipated from the system.

Capacitor: —C — A capacitor element relates the effort variable to the displacement variable. Ideally, no energy is lost. A capacitor element models the concept of potential energy.

Inertia: —I — An inertia element relates momentum to the flow variable. Ideally, no energy is lost. An inertia element models the concept of kinetic energy.

Source: S— — A source element adds energy to the system. The element may be designated as either an effort source S_e or a flow source S_f to indicate the variable being supplied.

2-Port Elements

Elements which consist of two ports (2-ports) are:

Gyrator:

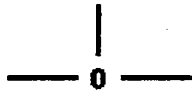
A gyrator changes the relationship between an effort variable and a flow variable. The relationships are defined by the equations $e_1 = rf_2$ and $e_2 = rf_1$ where e_n is the effort variable at port n, f_n is the flow variable at port n, and r is a constant. Ideally, power is conserved. A gyrator usually converts energy from one domain to another, such as from mechanical to electrical.

Transformer:

A transformer changes the relationship between the effort variable and the flow variable in a different manner. The relationships are defined by the equations $e_1 = me_2$ and $mf_1 = f_2$, where m is a constant. Ideally, power is conserved. Although a transformer can be constructed by placing two gyrator elements in series, it is a convenient, frequently used element. A transformer usually converts within the same domain, such as an electrical transformer.

Junctions

Additionally, the following multiport elements are used to represent junctions where two or more multiports are connected by bonds.

0-junction: 

Also referred to as a common effort junction, this multiport represents a connection of multiports where the efforts on all bonds of the junction are identical and the flows sum to zero. In electrical domains, the 0-junction represents a parallel connection.

1-junction: 

Also referred to as a common flow junction, this multiport represents a connection of multiports where the flows on all bonds of the junction are identical and the efforts sum to zero. In electrical domains, the 1-junction represents a series connection.

0-junctions and 1-junctions may be considered to have as few as two, or as many incident bonds as necessary to appropriately describe the physical system. Although the junction elements do not correspond to actual components in a physical system, they are a convenient way to represent

the interactions between multiports in terms of the relationships among the Power Variables.

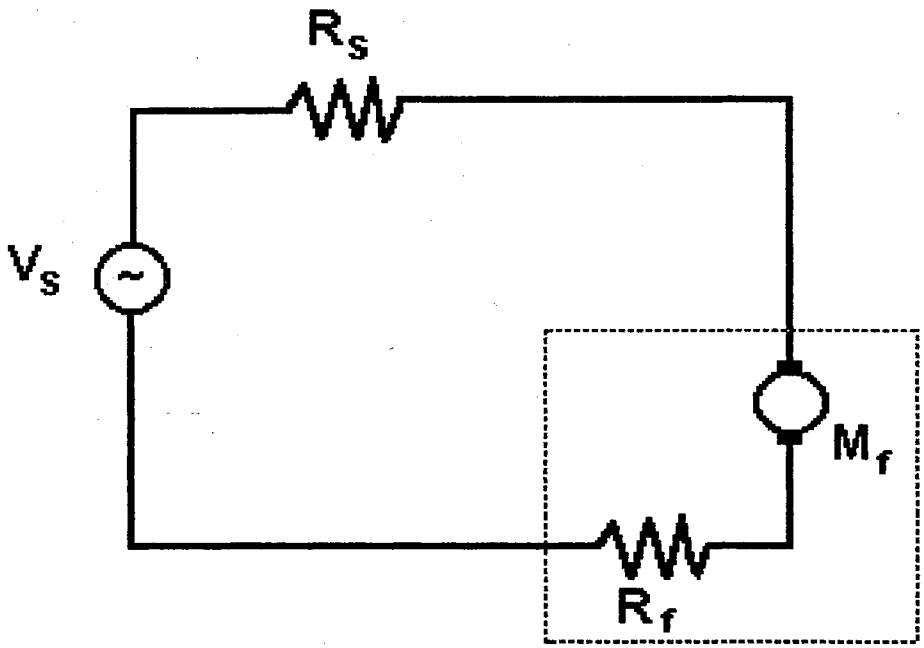


Figure 3. Schematic for Electric Fan Circuit

Figure 3 shows a schematic for a simple electrical circuit containing a power source and an electric fan. For the purpose of this example, the schematic contains only a minimum representation of the circuit elements. The dashed box indicates the electrical elements that comprise an equivalent circuit for the fan with M_f

representing the fan motor, and R_f representing the combined equivalent resistance of the resistive elements in the fan. In actuality, the fan would contain many more electrical and mechanical components than can be represented here.

Figure 4 shows the bond graph representation of the electric fan system including the electrical circuit shown in Figure 3 and the mechanical element of the fan blades. The multiport symbol **S** represents the power source. The junction among **S**, **R**, and **GY** represents the electrical connections from the power source to the electric motor contained in the fan.

The multiport **GY** represents the motor, which converts electrical energy from the power source into mechanical energy to drive the fan blades. The connection of the motor to the moving blades is represented by the junction **GY**, **I**, and **R**.

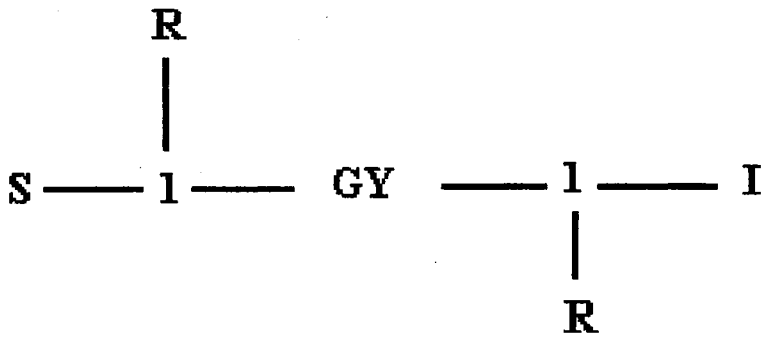


Figure 4. Bond Graph for Electric Fan System

The bond graph representation provides a convenient way to view physical system models in terms of a small number of basic component types, and it can be used to represent large, complex systems. The library of components developed in this work and outlined in the next chapter is based on the multiport elements shown above.

CHAPTER III

GENERIC COMPONENT REPRESENTATION

Purpose

Current representations for qualitative modeling of physical systems do not incorporate all useful knowledge of the components, especially with respect to connection capabilities and terminal variable unifiability. Since qualitative modeling is rapidly becoming a common method for system design, diagnosis, and tutoring, the generic component representation developed in this work has been designed to utilize all knowledge of components and connections that can expedite model building, yet does not place an additional burden on the user.

Modeling with the generic component representation is faster and easier than previous methods. Through the use of the accompanying graphical user interface, the user is not required to know the syntax of either CC or QSIM. The knowledge required of the user is limited to general concepts of physical systems, beginning level modeling concepts, and an understanding of the purpose of each of the components in the generic component library.

Component Types

To provide compatibility with the component-connection ontology utilized in CC [FD91], the generic component representation is designed around two types of components: primitive and composed.

Primitive Components

Primitive components are those components whose definitions are self-contained, i.e. no references are made to variables, constraints or other information outside the component definition.

Composed Components

Composed components are those components whose definitions consist of combinations of other components (either primitive or composed) and the connections between those components. A qualitative model is represented as a composed component.

Component Classes

An object-oriented hierarchy was chosen for the internal representation of component structures in PCL. The four classes in the hierarchy and their relationships to one another are shown in Figure 5.

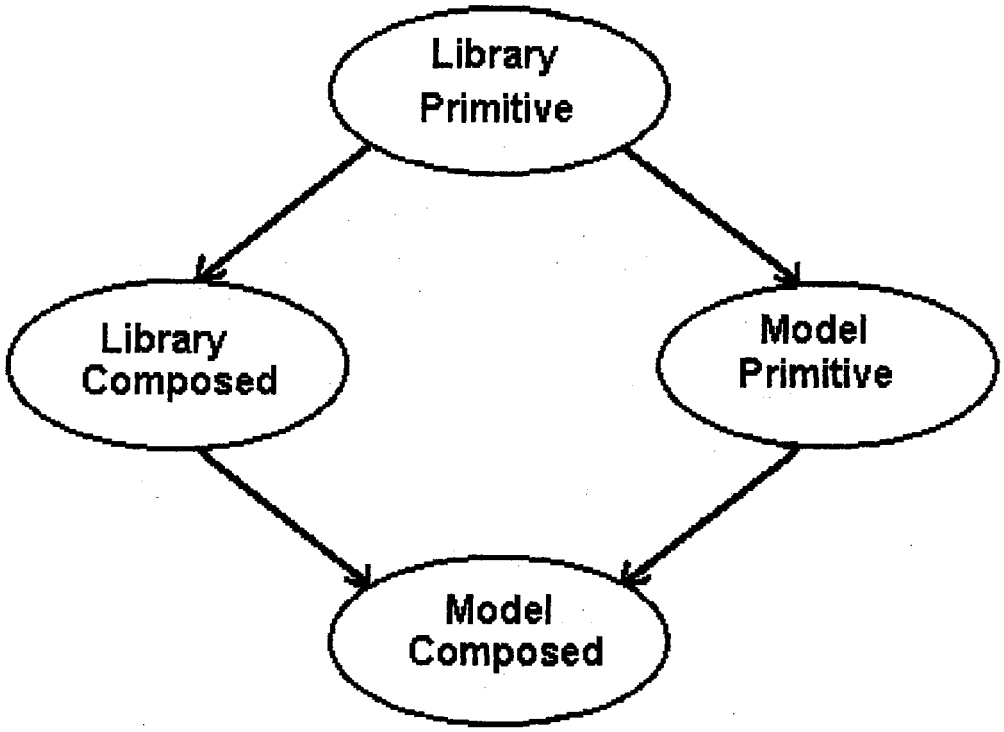


Figure 5. Component Class Hierarchy

Library Components

The library component class definitions contain information needed to represent the variables, terminals, and behavior of the component. Library components may be either primitive or composed. A set of library component definitions is loaded into the model building system from a file during the system startup phase. The user may select a component to be added to the model from the set of library components.

Model Components

The model component class definitions contain all of the information from the library component class definition and additional elements to represent its position in the model, its assigned domain, and customized constraints. Model components may be either primitive or composed.

When the user selects a component from the library, a copy of the library component definition is placed into a model component instance. The model component may then be customized to reflect its behavior to the specific model. Model component definitions are used to save and load qualitative models.

When the user selects a component from the set of previously saved composed component definitions, a reference copy of the composed component is added to the model. Any subsequent changes to the saved composed component definition will be reflected in each model where it is used.

Element Descriptions

Component Elements

The generic component representation consists of the set of elements listed in Table IV. The "Used In" column indicates whether a particular element is used in the library component definition, the model definition, or

both. Complete descriptions of the purpose and use of each of the elements are given in the following sections.

Table IV Generic Component Definition

Element Name	Default Value/ Type	Used In
Name	"" String	L, M
Variables	() List of Variable Objects	L, M
Terminals	() List of Terminal Objects	L, M
Constraints	() List of Constraint Forms	L, M
Domain	"unknown" String	M
X-coordinate	∅ Integer	M
Y-coordinate	∅ Integer	M
Bitmap	"" String	L, M
Restrictions	() List of Constraint Forms	L, M
Modifications	() List of Constraint Forms	M

Name. In a library component, the name field represents a predefined string assigned to the component.

In a model component, the user may enter a string to use as the external name for each component instance.

Variables. This field contains a list of variable object definitions which denote the component-level variables. The layout of the variable object definition is given in Table V.

Terminals. This field contains a list of terminal object definitions which denote the points through which the component may be connected to other components in the model. The layout of the terminal object definition is given in Table VI.

Constraints. This field contains a list of constraint forms which specify the constraints that apply to the component-level and terminal-level variables. Constraint forms have the following syntax:

(KEYWORD arg_1 arg_2 ...)

where KEYWORD is one of { CONSTANT, EQ, ADD, MULT, MINUS, M+, M-, d/dt } and arg_n is either a variable specification or a constant. Refer to Table II, Qualitative Constraint Definitions, for a description of each of the constraint keywords.

Domain. This field denotes the default domain for the component instance. Its value is one of { unknown, electrical, mechanical, hydraulic }.

X- and Y-coordinates. These contain the horizontal and vertical coordinates of the component within the graphical model display. The coordinates are given with respect to the upper left corner of the display grid.

Bitmap. This field is the string name of a bitmap file which contains a 25 pixel x 25 pixel component icon definition used in the graphical model display.

Restrictions. This field contains clauses which specify additional requirements placed on the component which cannot be given in constraint form. These clauses provide two forms of information needed by the generic model building system. First, they imply additional constraints on the component and the model to which the component belongs. Second, they act as signals to specify methods of consistency and completeness checking that will be performed as the model is built. The following restriction clauses have been defined as part of the generic component modeling system.

(DOMAIN-EQ *component1 component2*)

This clause establishes a requirement on a composed component (or qualitative model) that *component1* and *component2* always have the same domain. This clause prevents the inadvertent assignment of different domains to specific components in a model. For example, in certain

models certain combinations of electrical components and hydraulic components would not be permitted and can be prevented by adding this clause to the set of model restrictions.

(DOMAIN-EQ *terminal1 terminal2*)

This clause establishes a requirement that *terminal1* and *terminal2* of the same component have the same domain. This prevents the inadvertent change of domains across a single component. For example, both terminals of a resistor element must have the same domain since a resistor cannot be used to convert usable energy from one domain to another.

(OPTIONAL-TERMINAL *terminal*)

This clause permits the definition of terminals that may or may not be needed in a particular model. When the model is saved, checked, or translated, an optional terminal which has not been connected is discarded from the component definition.

Modifications. This field contains a list of constraint forms which the user has added to customize the behavior of the component for the specific model.

Variable Elements

Components may have two different types of variables: component variables or terminal variables. The terminal variables are those variables which are assigned to specific terminals of the component. Component variables correspond to other variables of interest in the model and are related to other variables of the component (either component or terminal) through constraints [KU92]. The representation of both types of variables is the same and is shown in Table V.

Table V Variable Object Definition

Element Name	Default Value/ Type
Name	"" String
Type	"effort" String
Quantity Space	() List of Landmark Values
Domain	"unknown" String

Terminal Elements

A terminal object represents a means through which a component may interact with other components. Connections in a composed component are given by specifying the component name and terminal name for each end point of the connection. Terminal names within each component must be unique but terminal names may be reused between components. The terminal object definition is shown in Table VI.

Table VI Terminal Object Definition

Element Name	Default Value/ Type
Name	"" String
Variables	() List of Variable Objects
Position	none Integer
Direction	IN $\in$ {IN, OUT, NONE}
Connected Flag	false Boolean

Generic Component Library

The initial set of generic components developed in this work is based on the concepts of basic multiports

described in Chapter 2. Each generic component contains the minimum information necessary to embody the purpose of the multiport but without any extra information that might restrict the user from utilizing the component as they wish in a specific model. The library definition of each component is fixed. Copies of the component definition are included in the user's model and can be customized as desired.

Table VII Library Component Descriptions

Component Name	Terminals	Behavior
Effort Source	T1, T2	SE >= Ø T1.effort + SE = T2.effort
Flow Source	T1, T2	SF >= Ø T1.flow + SF = T2.flow
Resistor	T1, T2	EACROSS >= Ø M+(EACROSS, T1.flow) T2.effort + EACROSS = T1.effort T1.flow = T2.flow
Capacitor	T1, T2	M+(T1.effort, displacement) d/dt(displacement, T1.flow)
Inertia	T1, T2	M+(T1.flow, momentum) d/dt(momentum, T1.effort)
Gyrator	T1, T2, T3, T4	M+(T1.flow, momentum) d/dt(momentum, T1.effort) T1.flow = T2.flow
Transformer	T1, T2, T3, T4	T1.flow * T2.flow = M T1.effort * T2.effort = M T1.effort * T1.flow = power T2.effort * T2.flow = power

The generic components in the initial test library and their definitions are outlined in Table VII. Appendix C contains a complete listing of the generic component test library definitions.

Terminal Compatibility

To aid in the model building process several checks for terminal compatibility have been defined. Each of these checks provides a means of detecting possible errors in the qualitative model structure early in the modeling process rather than during the simulation phase. Terminal compatibility checking is a feature of the generic modeling system that is not available in other qualitative modeling systems.

Compatibility Warnings and Errors

Same Component Warning. If the user attempts to connect a terminal on a component to another terminal on the same component, a warning message is issued. The user may elect whether to proceed with the connection or cancel the attempt.

Different Domains Error. If the user attempts to connect a terminal on a component to a terminal with a different domain, an error message is issued. The connection attempt is cancelled.

Direction Conflict Warning. If the user attempts to connect a terminal on a component to a terminal with the same relative direction, for example, connecting a terminal with direction IN to another terminal with direction IN, a warning is issued. Although this type of conflict can be resolved by CC in some cases, deliberately connecting components in this manner is not recommended.

CHAPTER IV

GENERIC MODEL BUILDING SYSTEM

Implementation

The implementation of concepts described in this work has been developed as a series of programs collectively called the "Generic Model Building System" (GMBS). The purpose of the implementation is to allow a user quickly to build a qualitative model, translate it to a format acceptable to QSIM, then invoke the QSIM package to simulate the model, all from a single user interface screen. A description of each of the programs that comprise GMBS and the options available through the graphical user interface of the system is included in this chapter.

Hardware and Software Requirements

GMBS has been developed on a Sequent Dynix/PTX system, using a flavor of Unix that somewhat resembles System V, version 3 but has several known inconsistencies. Due to the particular instructions used to start the execution of the GMBS component programs, GMBS is not currently portable to operating systems other than Unix, and is not guaranteed

to work properly on Unix systems other than Sequent Dynix/PTX.

The graphical user interface was developed using X11R5 and the Motif widget set. An X Windows terminal is required for use of the interface.

Program Descriptions

A combination of LISP (specifically Austin Kyoto Common Lisp -- AKCL) and C programs was used in the development of the Generic Model Building System for several reasons. First, because CC and QSIM are both written in LISP, using LISP as the language for creating and maintaining internal model structures provides a more efficient basis for translating the model to CC and subsequently simulating it with QSIM. Second, although an X Windows interface development system written in LISP is currently available, it is not currently portable to Dynix/PTX for use with AKCL. As a more convenient and familiar language, C was chosen for the X Windows interface because of the availability of Intrinsics library and Motif widget set. Finally, C provides appropriate commands for linking the input and output of executables from different programming languages. The major programs that comprise the GMBS package are:

Program 1: GMstart. This executable is compiled from a C source program. Its purpose is to invoke (by forking) separate processes for the LISP routines and the X Windows interface, load the LISP programs, and establish linkage between the standard input and output (stdio/stdin) pairs of LISP and X Windows.

Program 2. Gmbs. This executable is compiled from a C source program. Its purpose is to establish the X Windows environment and invoke the main X Windows interface program.

Program 3. gmbs. This executable is compiled from a C source program. It contains the main X Windows interface routines.

Program 4. clos. This system wide command invokes an executable which contains Portable Common Loops (PCL), an object-oriented programming system, layered over AKCL.

Complete instructions for building the GMBS executables are included with the source code distribution in the file "README.TXT".

Starting GMBS

GMBS is invoked by typing GMstart at the Unix prompt. This should be done without any window manager running to avoid having any window decorations overlap the GMBS display. Several lines of text will be displayed while the

system is loading, then the main window will appear as shown in Figure 6.

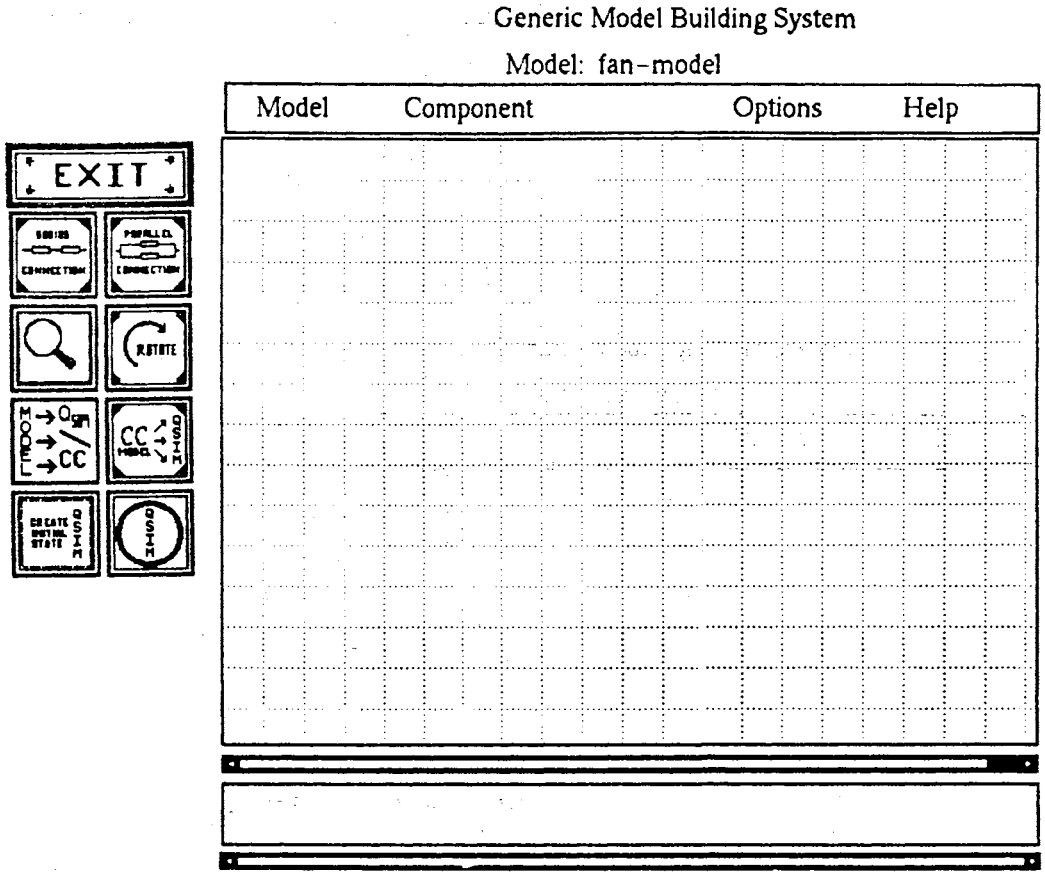


Figure 6. Generic Model Building System Main Window

GMBS Main Window Sections

The main window contains five important sections: a title area at the top of the screen, a toolbar with nine

buttons on the left side, a menu bar with four pulldown menus, a large drawing area with grid, and a message text area along the bottom of the screen.

GMBS Pulldown Menus

The menu bar in the GMBS main window contains pulldown menus for four groups of options: Model Options, Component Options, Option Options and Help Options. The options available on these menus are described in the following sections.

Model Menu

The Model Menu is a collection of options designed to allow the user perform various options at the model level. Using the mouse pointer, the user may pull down the model menu as shown in Figure 7.

Model Menu Options

New. This option allows the user to create a new model with zero components. The user is prompted for a model name and the default domain for the model. The model display area is cleared and the system is ready for the user to begin adding components to the model.

If a model already exists in the display area and it has been modified since the last save operation, the user

is prompted for whether to save the existing model before creating a new model.

Load. This option allows the user to load a model from a previously saved model definition. The user is prompted for a file name from which to load the model. The model definition is loaded into the system and the model is drawn in the display area.

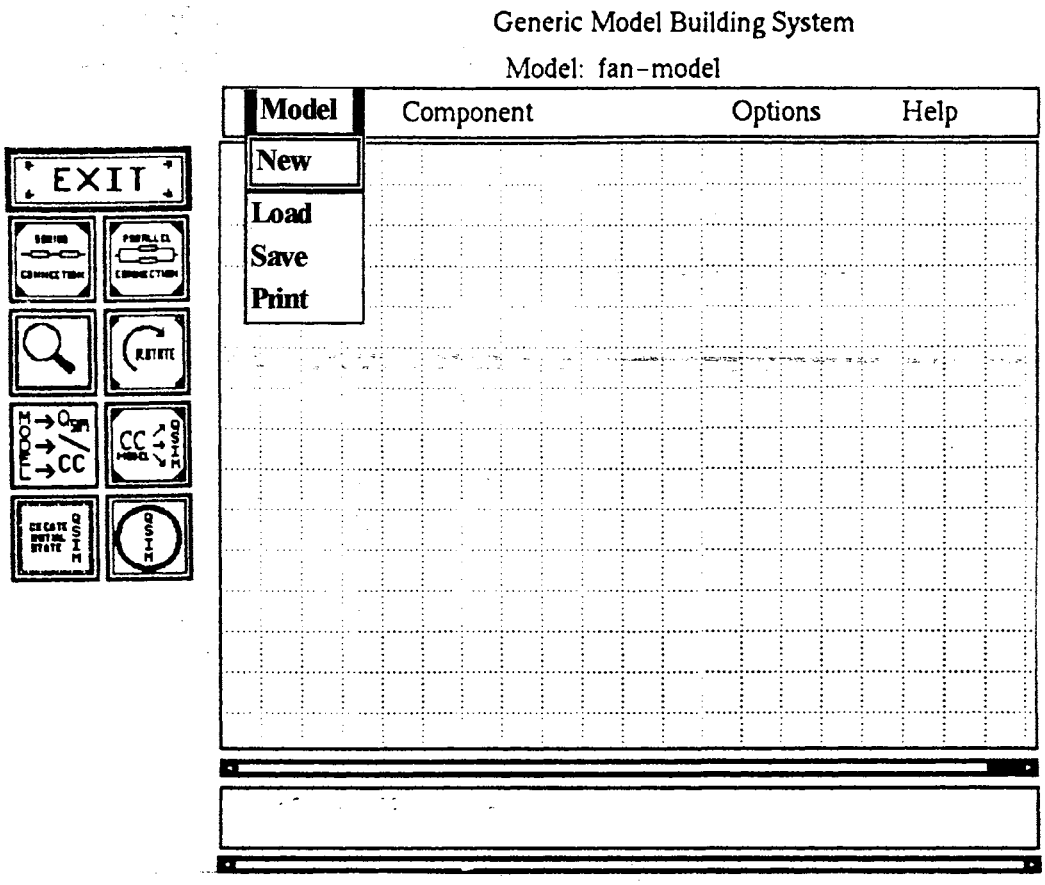


Figure 7. Model Menu Pulldown

Save. This option allows the user to saves a LISP-readable version of the model description to a file. The user is prompted for the file name.

Print. This option allows the user to obtain a printed copy of the model description. A temporary file containing a GMBS-readable version of the model description is created, then spooled to a printer. The user is prompted for the printer name.

Component Menu

The Component Menu is designed to allow the user to perform various options at the component level. The appearance of the component menu when it is pulled down is shown in Figure 8.

Component Menu Options

Select. This option allows the user to select a component from the list of components in the current library or the list of previously stored composed component definitions. The user is prompted for the component selection, the component name, and a location at which to place the component icon in the model display.

Modify. This option allows the user to modify a component in the current model. Elements of the component

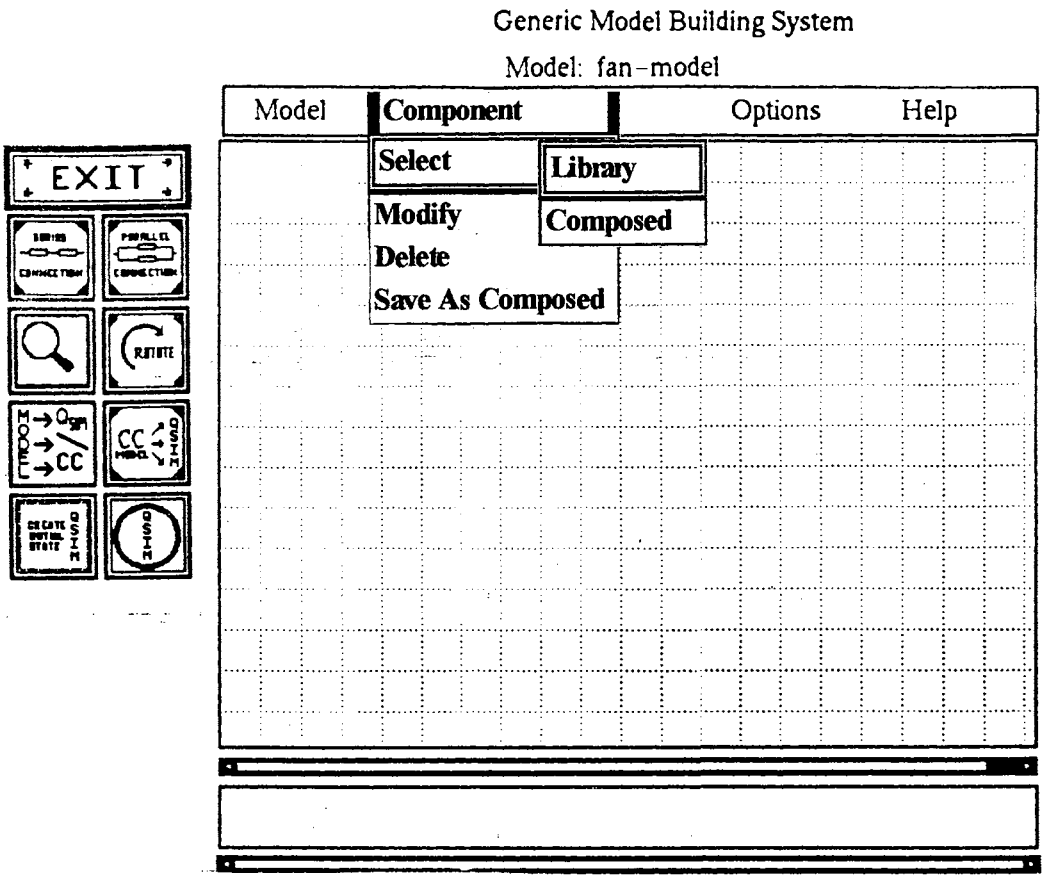


Figure 8. Component Menu Pulldown

that may be modified include the name, terminal positions, component variables, and constraints.

Delete. This option allows the user to remove a component from the current model. The component icon and any connections drawn to it are removed from the model display. The component definition and any connection

definitions referencing the component are removed from the model definition.

Save As Composed. This option allows the user to save the current model definition as a composed component. The user is prompted for a file name in which to store the composed component definition. The composed component may be reused in future model building and the component will appear as a single icon in the model display.

Options Menu

The Options Menu is designed to allow the user to select which operations to be automatically performed on the model. Having the appropriate options selected can speed up model building significantly since less input is required from the user when adding components or connections to the model. However, the user may want to exercise control over choices made in the model rather than allow the system to generate them.

The appearance of the options menu with the default settings is shown in Figure 9. A filled box appears next to options that are enabled or set "ON".

Option Settings

Snap Lines to Grid. When this option is set, any points selected for connection end points are moved to

intersections on the display area grid. This option is available primarily for aesthetic considerations.

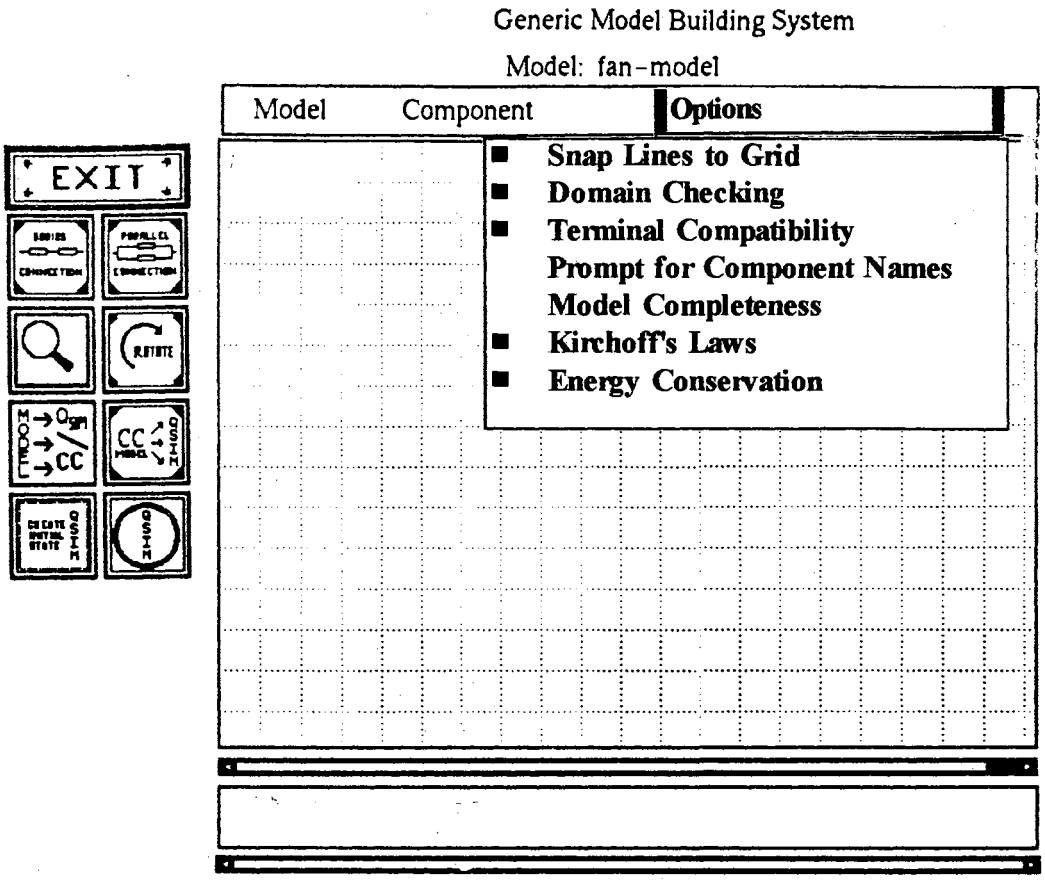


Figure 9. Options Menu Pulldown

Domain Checking. When this option is set, the system compares the default domain of the model to the domain requirements of each component added to the model. Domain

values for components, terminals and variables in the sytem are generated automatically whenever possible. The user is prompted only when a domain selection cannot be resolved.

Terminal Compatibility. When this option is set, the system compares the attributes of each terminal involved when a connection request is made. The user is prompted for needed information or given appropriate warning or error messages.

Prompt for Component Names. When this option is set, the user is prompted for a string to use for the component name each time a component is added to the model. When the option is disabled, GMBS automatically generates a string name for each component.

Model Completeness. When this option is set, the system performs a series of operations to determine the completeness status of the model when any of the following options are selected: Model Save, Save Component as Composed, or Translate to CC/QSIM Format.

Kirchoff's Laws. When this option is set, the system automatically adds appropriate constraints to the model when the option Translate to CC/QSIM Format is selected. These constraints are designed to ensure that the model conforms to Kirchoff's Voltage and Current Laws.

Energy Conservation. When this option is set, the system automatically adds appropriate constraints to the model when the option Convert Model to CC/QSIM Format is selected. These constraints are designed to ensure that the model obeys the laws of Energy Conservation.

Toolbar Functions

At the left side of the GMBS Main Window a series of buttons appears. These buttons represent the toolbar functions which allow the user to rapidly perform many of the more commonly used operations in model building.

Toolbar Button Descriptions

Exit

Activating this button initiates a process to terminate execution of the model building system. If the model display has been modified since the last Save operation, the user is prompted to save the model before exiting.

Series Connection

Activating this button allows the user to create a series connection between two components in the model. The user is prompted for the two terminals to use as end points for the connection. Adding

additional connections to terminals which are already connected is not allowed.

Parallel Connection

Activating this button allows the user to insert two parallel junctions then connect the junctions to the appropriate components in the model. The user is prompted for the locations of the junctions and the end point terminals to be connected to the junctions.

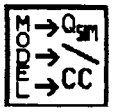
Examine Component

Activating this button allows the user to obtain information about a particular component in the model display. The user is prompted for the component. A popup window with a description of the selected component including name, constraints, terminals, and modifications is displayed.

Rotate Component

Activating this button allows the user to rotate a particular component in the model display 90 degrees clockwise. Rotation of a component must be done prior to making any connections to that component.

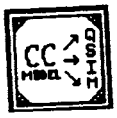
Translate to CC/QSIM Format



Activating this button initiates the process of translating the current model to a format acceptable to CC and/or QSIM. The user is

prompted for a file name in which to store the translated model.

Convert CC to QSIM



Activating this button initiates execution of CC to convert the CC formatted file to a QSIM input format file. The user is prompted for a file name in which to store the QSIM input.

Create QSIM initial state

Activating this button allows the user to select initial values for variables and the direction of value change of those variables. The user is prompted for a file name to store the QSIM initial state.

Execute QSIM

Activating this button initiates execution of QSIM to simulate the model and display the resulting behaviors. The user is prompted for the file names of the QSIM input and the QSIM initial state.

CHAPTER V

SUMMARY AND FUTURE WORK

Evaluation of Generic Component Representation

This paper has presented the development of a new approach to qualitative model building using a generic component representation. The generic representation and the accompanying implementation of the Generic Model Building System provide several benefits over previously existing software. Concentration has been placed on improving the modeling phase of qualitative reasoning where previously it has been primarily on the simulation phase.

The ease and efficiency of model building have been increased, especially for users unfamiliar with CC or QSIM syntax. Automated checking for possible errors during the model building stage has been added.

The generic component representation contains elements for including information for constructing component-based qualitative models for simulation and for displaying those models through a graphical user interface.

Eight basic library components have been developed using the generic component representation. These components provide a means for constructing models for

virtually any physical system consisting of components from any combination of the electrical, mechanical, and hydraulic domains.

A prototype generic model building system has been developed, linking a LISP-based processing system and an X Windows graphical user interface. This prototype can be used to build simple models in electrical, mechanical, and hydraulic domain combinations. With the user interface one can rapidly construct, translate and simulate a model, as well as quickly revise the model to improve the results of simulation.

Several simple models were constructed using the generic model building system to test the viability of the representation and its compatibility for translation to CC/QSIM formats. The details of the construction of one model used as a test are given in Appendix D.

Future Work

Several possible avenues for further research into generic component use in qualitative modeling have been recognized at this time. The development of components suitable for use in the thermodynamic and medical domains would greatly expand the types of physical systems that could be modeled. The addition of multi-domain terminal capabilities, where a single terminal may have

qualitatively important variables from two or more domains, would allow for creation of integrated components.

The design of multi-mode or multi-function components, those where the value, either symbolic or numeric, of a component variable determines which set of constraints will be used to model the component's behavior, could be used to develop multi-purpose qualitative models.

Further testing by individuals familiar with the concepts of qualitative simulation, component connections modeling, CC and QSIM can be used to provide feedback for further enhancements of the system.

Also important to making the generic model building system a viable method for modeling non-trivial physical systems is the development of a larger set of library components and a set of frequently used composed component model definitions. This will allow even faster model building because the user will be able to draw from the library of predefined components rather than constructing large models strictly from library primitives.

Implementing various operating system dependent mechanisms to provide portability to other hardware and software can lead to the development of a distributable package for qualitative modeling.

BIBLIOGRAPHY

[AS90] Asente, P. and Swick, R. 1990. *X Window System Toolkit, The Complete Programmer's Guide and Specification.* Digital Press, Boston, MA.

[BB91] Broenink, J.; J. Bekkink; A. Kok; and P. Breedveld. 1991. "Multibond-graph Version of the CAMAS Modeling and Simulation Environment." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 1086-1087.

[BK92] Berleant, D. and B. Kuipers. 1992. "Qualitative-Numeric Simulation with Q3". In *Recent Advances in Qualitative Physics.* B. Faltings and P. Struss (eds). MIT Press. Cambridge, MA.

[BM93] Biswas, G.; S. Manganaris; and X. Yu. 1993. "Extending Component Connection Modeling for Analyzing Complex Physical Systems." *IEEE Expert* 8, no. 1: 48-57.

[BO93] Bobrow, D.G. 1993. "Qualitative Reasoning About Physical Systems: An Introduction." *Artificial Intelligence* 24: 1-5.

[CE??] Crawford, J. and D. Etherington. "Formalizing Reasoning about Change: A Qualitative Reasoning Approach".

[CE90] Cellier, F. 1990. "General System Problem Solving Paradigm for Qualitative Modeling."

[CF90] Crawford, J.; A. Farquhar; and B. Kuipers. 1990. "QPC: A Compiler from Physical Models into Qualitative Differential Equations." In *AAAI-90 Proceedings of the Eighth National Conference on Artificial Intelligence* (Jul. 29 - Aug. 3). MIT Press, Cambridge, MA, 365-372.

[CF91] Charles, A.; P. Fouché; and C. Mélin. 1991. "Recent Improvements in Qualitative Simulation." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 1745-1746.

[DE93] de Kleer, J. 1993. "A View on Qualitative Physics." *Artificial Intelligence* 59: 105-114.

[DR90a] Dague, P.; O. Raiman; and P. Deves. 1990. "Troubleshooting: when Modeling is the Trouble." In Readings in Qualitative Reasoning about Physical Systems. D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[DR90b] Dormoy, J. and O. Raiman. 1990. "Assembling a Device." In *Readings in Qualitative Reasoning about Physical Systems.* D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[FA82] Faber, Rodney B. 1982. *Applied Electricity and Electronics for Technology.* John Wiley and Sons, New York, NY.

[FD91] Franke, D. and D. Dvorak. 1991. *CC: Component Connection Models for Qualitative Simulation. A User's Guide.* Internal Documentation. Department of Computer Sciences, University of Texas at Austin.

[FF90] Falkenhainer, B. and K. Forbus. 1990. "Setting up Large-Scale Qualitative Models." In *Readings in Qualitative Reasoning about Physical Systems.* D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[FF92] Falkenhainer, B. and K. Forbus. 1992. "Compositional Modeling of Physical Systems." In *Recent Advances in Qualitative Physics.* P. Struss (ed). MIT Press, Cambridge, MA.

[FI92] Fishwick, P. 1992. "An Integrated Approach to System Modeling Using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies." *ACM Transactions on Modeling and Computer Simulation* 2, no. 4 (Oct.): 285-306.

[FK92a] Fouche, P. and B. Kuipers. 1992. "Reasoning about Energy in Qualitative Simulation". *IEEE*

Transactions on Systems, Man, and Cybernetics 22, no. 1 (Jan./Feb.): 47-63.

[FK92b] Farquhar, A.; B. Kuipers; J. Rickel; and D. Throop. 1992. *QSIM: The Program and its Use.* Internal Documentation. Department of Computer Sciences, University of Texas at Austin.

[FL91] Fishwick, P. and P. Luker (eds). 1991. *Qualitative Simulation, Modeling, and Analysis.* Springer-Verlag, New York, NY.

[FM91] Feray-Beaumont, B. and A. Morris. 1991. "Towards Qualitative Control Engineering." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 1734-1735.

[FO84] Forbus, K.D. 1984. "Qualitative Process Theory." *Artificial Intelligence* 24: 85-168.

[FO88] Forbus, K.D. 1988. "Qualitative Physics: Past, Present, and Future." In *Exploring Artificial Intelligence.* Morgan Kaufman, New York, NY.

[FO92] Forbus, K. and B. Falkenhainer. 1992. "Self-Explanatory Simulations: Integrating Qualitative and Quantitative Knowledge". In *Recent Advances in Qualitative Physics.* B. Faltings and P. Struss (eds). MIT Press. Cambridge, MA.

[FZ92] Fishwick, P. and B. Zeigler. 1992. "A Multimodel Methodology for Qualitative Model Engineering." *ACM Transactions on Modeling and Computer Simulation* 2. no. 1 (Jan.): 52-81.

[GR72] Greenberg, S. 1972. *The GPSS Primer.* John Wiley and Sons, New York, NY.

[GR92] Guariso, G.; A. Rizzoli; and H. Werthner. 1992. "Identification of Model Structure via Qualitative Simulation." *IEEE Transactions on Systems, Man, and Cybernetics* 22, no. 5 (Sept./Oct.): 1075-1086.

[KA90] Karnopp, D. 1990. *System Dynamics: A Unified Approach.* John Wiley & Sons, New York, NY.

[KC90] Kuipers, B.J. and C. Chiu. 1990. "Taming Intractible Branching in Qualitative Simulation."

In *Readings in Qualitative Reasoning about Physical Systems.* D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[KC91] Kuipers, B.J.; C. Chiu; D. Molle; and D. Throop. 1991. "Higher-Order Derivative Constraints in Qualitative Simulation." *Artificial Intelligence* 51: 343-379.

[KU86] Kuipers, B.J. 1986. "Qualitative Simulation." *Artificial Intelligence* 29: 289-338.

[KU90] Kuipers, B.J. 1990. *Diagnosis and Model-Building for Dynamic Systems.* Draft of Proposal Submitted to NASA. Department of Computer Science. The University of Texas at Austin.

[KU92] Kuipers, B.J. 1992. "Component-Connection Models." Draft. Department of Computer Science. The University of Texas at Austin.

[KU93a] Kuipers, B.J. 1993. "Qualitative Simulation: Then and Now." *Artificial Intelligence* 59: 133-140.

[KU93b] Kuipers, B.J. 1993. "Reasoning with Qualitative Models." *Artificial Intelligence* 59: 125-132.

[LE91] Leitch, R. 1991. "Themes and Variations in the Development of Qualitative Reasoning." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 966-968.

[LF90] Liu, Z. and A. Farley. 1990. "Shifting Ontological Perspectives in Reasoning about Physical Systems." In *Proceedings of the Eighth National Conference on Artificial Intelligence* (Jul. 29 - Aug. 3). MIT Press, Cambridge, MA, 395-400.

[LM90] Lawless, J. and M. Miller. 1990. *Understanding CLOS: The Common Lisp Object System.* Digital Press, Boston, MA.

[LN93] Lackinger, F. and W. Nejdl. 1993. "Diamon: A Model-based Troubleshooter Based on Qualitative Reasoning." *IEEE Expert* 8, no. 1 (Feb.): 33-39.

[LS92] Lee, Y. and J. Stascavage. 1992. "Multitasking Simulation of a Boiler System Using Qualitative Model-Based Reasoning." *ACM Transactions on Modeling and Computer Simulation* 2, no. 4 (Oct.): 285-306.

[MF92] Miller, D.; R. Firby; P. Fishwick; D. Franke; and J. Rothenberg. 1992. "AI: What Simulationists Really Need to Know". *ACM Transactions on Modeling and Computer Simulation* 2, no. 4 (Oct.): 269-284.

[MM91] Makarovic, A. and N.J.I. Mars. 1991. "Fundamental Limitations of Qualitative Simulation." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 1743-1744.

[NO92] Noyes, J. 1992. *Artificial Intelligence with Common Lisp.* D. C. Heath and Company, Lexington, MA.

[PR74] Pritsker, A. 1974. *The GASP IV Simulation Language.* John Wiley and Sons, New York, NY.

[RO93] Rosenberg, R. 1993. "Reflections on Engineering Systems and Bond Graphs." Journal of Dynamic Systems, Measurement, and Control 115 (Jun.): 242-251.

[SL91] Shen, Q. and R. Leitch. 1991. "Diagnosing Continuous Dynamic Systems Using Qualitative Simulation." In *Proceedings of the 1991 International Conference on Control* (Edinburgh, UK, Mar. 25-28). IEE, London, 1000-1006.

[ST90a] Steele, G. 1990. *Common Lisp: The Language.* Second Edition. Digital Equipment Corporation. Boston, MA.

[ST90b] Struss, P. 1990. "Global Filters for Qualitative Behaviors". In *Readings in Qualitative Reasoning about Physical Systems.* D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[WD91] Williams, B. and J. de Kleer. 1991. "Qualitative Reasoning about Physical Systems: A Return to the Roots." *Artificial Intelligence* 51: 1-9.

[WE92] Weld, D. 1992. "Reasoning about Model Accuracy." *Artificial Intelligence* 56: 255-300.

[WI90a] Williams, B. 1990. "Temporal Qualitative Analysis: Explaining How Physical Systems Work." In Readings in Qualitative Reasoning about Physical Systems. D. Weld and J. deKleer (eds). Morgan Kaufmann, San Mateo, CA.

[WI90b] Williams, B. 1990. "Interaction-based Invention: Designing Novel Devices from First Principles." In *Proceedings of the Eighth National Conference on Artifical Intelligence* (Jul. 29 - Aug. 3). MIT Press, Cambridge, MA, 395-400.

[XL91] Xia, S.; D. Linkens; and S. Bennett. 1991. "Integration of Qualitative Reasoning and Bond Graphs: Project Introduction." In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics* (Dublin, Ireland, Jul. 22-26). Trinity College, Dublin, 1079-1080.

[YO89] Young, D. 1989. *The X Window System: Applications and Programming with Xt (Motif Version).* Prentice-Hall.

APPENDIX A

STEAM PROPULSION SYSTEM

QSIM Input

```
;; Qualitative Model Definition
;;
(define-QDE steam-plant

; Documentation
   (text "Steam plant with two major components")

   (quantity-spaces
       (SV (0 V* inf)                "Steam Velocity")
       (EB (0 E* inf)                "Boiler Energy")
       (TIN (0 T* inf)                "SH Input Temp")
       (TOUT (0 TOUT*  inf)           "SH Output Temp")
       (TSDIFF (0 TS*  inf)          "SH temp Difference")
       (WIN  ( 0 W*  inf )           "Water Input Temp")
       (WDIFF (0 WD* inf)            "Temp Diff in Boiler")
       (QSP (0 Q*  inf)              "Steam Produced")
       (SHTIME (0 ST* inf)           "Time in SH")
   )

   (constraints
       ((M- WDIFF WIN) (WD* W*))
       ((M+ SV QSP) (0 0) (V* Q*) (inf inf))
       ((M- QSP WDIFF) (0 inf) (Q* WD*) )
       ((ADD WDIFF WIN TIN))
       ((ADD TIN TSDIFF TOUT))
       ((M- SV SHTIME) (inf 0) (V* ST*) (0 inf))
       ((M+ SHTIME TSDIFF) (ST* TS*) (0 0) (inf inf))
       ((constant EB))   ; energy supplied by the boiler
                         ; is a constant
       ((constant TIN))  ; temperature of steam going in
                         ; is a constant T* = 212
   )

   (layout   (WIN  TIN WDIFF)
             (QSP  SV   SHTIME)
             (TOUT nil  nil)
   )
```

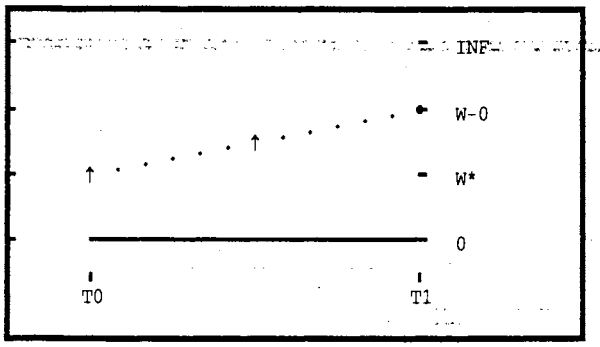
```
   (other    (qspace-hierarchy ((WIN TIN TOUT) -> (*seq 0 W*
               T* TOUT* inf)))
             (unreachable-values (SV inf) (QSP inf)(WDIFF 0)
             (TIN inf) (TOUT inf))
))
```

QSIM Initial State

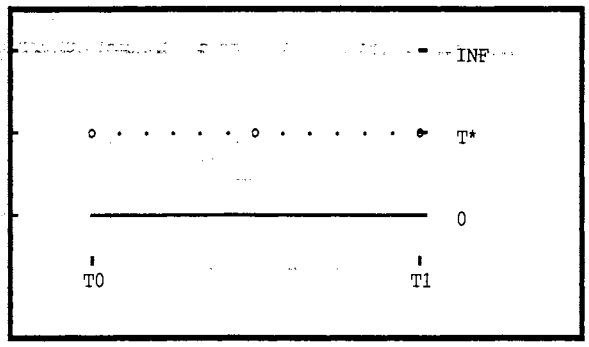
```
;;
;; Initial State Definition
;;
(defun warmer-water ()
   (let ((initial-state
             (make-new-state
               :from-qde steam-plant
               :assert-values
                    '((WIN (W* inc))
                      (TIN (T* std))
                      (TOUT (TOUT* nil))
                      (QSP (Q* nil))
                      (SHTIME (ST* nil))
                      (WDIFF (WD* nil))
                      (TSDIFF (TS* nil))
                      (SV (V* nil))
                      (EB (E* std)))
               :text "Increase the water intake temperature"
             )
          )
         )

;; execute the simulation for this initial state
;; and display the results
(qsim initial-state)
(qsim-display initial-state)))
```

QSIM Sample Output

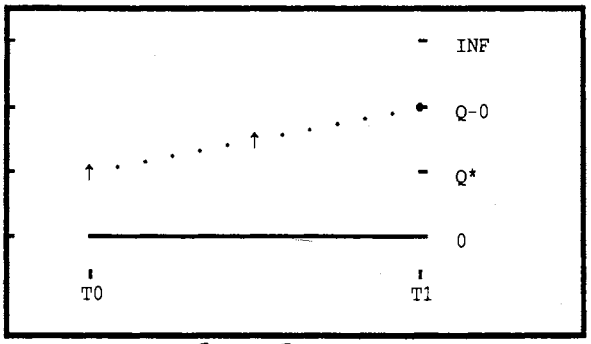
Structure: Steam plant with two major components.
Initialization: Increase the water intake temperature (S-0)
Behavior 1 of 1: (S-0 S-1 S-3).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.



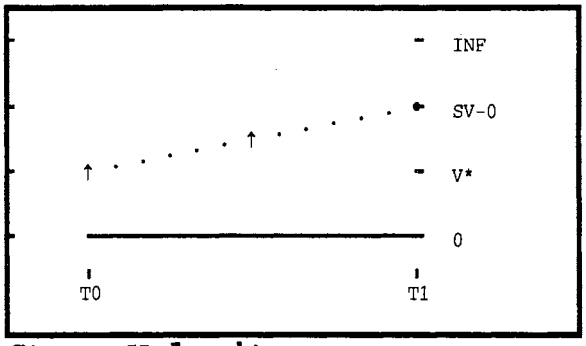
Water Input Temp



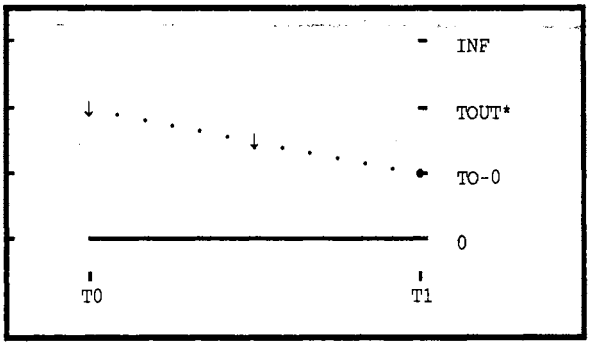
SH Input Temp



Steam Produced



Steam Velocity



SH Output Temp

APPENDIX B

ELECTRICAL CIRCUIT EXAMPLE

CC Input

```
(define-component Resistor electrical
     "Resistor:  Ohm's law"
     (terminal-variables      (t1  (v1 voltage)
                              (i current))
                              (t2  (v2 voltage)
                              (i2 current))
     (component-variables     (v voltage)
                              (r resistance
                              (quantity-space (0 R* inf))))
     (mode-variables (mode working burnout))
     (constraints ((add v v2 v1))
               ((mult i r v))
               ((minus i i2))
               ((mode working) -> ((constant r R*)))
               ((mode burnout) -> ((constant r inf)))))

(define-component Battery electrical
     "Battery: constant voltage source"
     (terminal-variables      (t1  (v1 voltage)
                              (i current))
                              (t2  (v2 voltage)
                              (i2 current))
     (component-variables     (v voltage
                         (quantity-space (0 vbat inf))))
     (constraints ((add v v2 v1))
               ((minus i i2))
               ((constant v vbat))))

(define-component Capacitor electrical
  "Capacitor:  container for charge"
     (terminal-variables      (t1  (v1 voltage)
                              (i current))
                              (t2  (v2 voltage)
                              (i2 current)))
```

```
     (component-variables     (v voltage)
                              (c capacitance
                              (quantity-space (0 C* inf)))
                              (q charge))
     (constraints   ((add v v2 v1))
               ((mult v c q))
               ((d/dt q i))
               ((minus i i2))
               ((constant c C*))))

(define-component Ground electrical
     "Ground:  constant voltage (current sink)"
     (terminal-variables      (t   (v voltage)
                                   (i current)))
     (constraints   ((constant v 0))
               ((constant i 0))))

(define-component Switch electrical
     "Switch:  externally opened or closed"
     (mode-variables (mode open closed))
     (terminal-variables      (t1  (v1 voltage)
                               (i1 current))
                               (t2  (v2 voltage)
                               (i current)))
     (component-variables      (v voltage))
     (constraints ((add v v2 v1))
               ((minus i i1))
               ((mode open)   ->  ((constant i 0)))
               ((mode closed)   ->  ((constant v 0)))))

(define-component RC electrical
     "Resistor-Capacitor Circuit"
     (components    (B battery)
                    (R resistor)
                    (C capacitor)
                    (S switch)
                    (G ground))
     (connections   (n1  (R t1)    (S t1))
                    (n2  (R t2)    (C t1))
                    (n3  (C t2)    (B t2)    (G t))
                    (n4  (B t1)    (S t2))))
```

CC Output

```
(make-QDE
  :name
     RC_UNIQUE_WITH_S.MODE=CLOSED_WITH_R.MODE=WORKING

  :qspaces        ((TIME t0 inf)
                   (B.V 0 vbat inf)
                   (R.V minf 0 inf)
                   (R.R 0 r* inf)
                   (C.V minf 0 inf)
                   (C.C 0 c* inf)
                   (C.Q minf 0 inf)
                   (S.V minf 0 inf)

                   (N1.EFFORT_ELECTRICAL
                       minf 0 inf)
                   (R.I minf 0 inf)
                   (S.I1 minf 0 inf)
                   (N2.EFFORT_ELECTRICAL
                       minf 0 inf)
                   (R.I2 minf 0 inf)
                   (C.I minf 0 inf)
                   (N3.EFFORT_ELECTRICAL
                       minf 0 inf)
                   (C.I2 minf 0 inf)
                   (B.I2 minf 0 inf)
                   (G.I minf 0 inf)
                   (N4.EFFORT_ELECTRICAL
                       minf 0 inf)
                   (B.I minf 0 inf)
                   (S.I minf 0 inf)
                   (S.MODE open
                           closed)
                   (R.MODE working
                           burnout))

  :constraints    ((CONSTANT R.MODE WORKING)
                   (CONSTANT R.R R*)
                   (CONSTANT S.MODE CLOSED)
                   (CONSTANT S.V 0)
                   (ADD B.V N3.EFFORT_ELECTRICAL
                    N4.EFFORT_ELECTRICAL)
                   (MINUS B.I B.I2)
                   (CONSTANT B.V VBAT)
                   (ADD R.V N2.EFFORT_ELECTRICAL
                    N1.EFFORT_ELECTRICAL)
                   (MULT R.I R.R R.V)
                   (MINUS R.I R.I2)
                   (ADD C.V N3.EFFORT_ELECTRICAL
                    N2.EFFORT_ELECTRICAL)
```

```
                (MULT C.V C.C C.Q)
                (D/DT C.Q C.I)
                (MINUS C.I C.I2)
                (CONSTANT C.C C*)
                (ADD S.V N4.EFFORT_ELECTRICAL
                 N1.EFFORT_ELECTRICAL)
                (MINUS S.I S.I1)
                (CONSTANT N3.EFFORT_ELECTRICAL 0)
                (CONSTANT G.I 0)
                (MINUS R.I S.I1)
                (MINUS R.I2 C.I)
                (SUM-ZERO C.I2 B.I2 G.I)
                (MINUS B.I S.I))

:independent    NIL

:history        NIL

:transitions    NIL

:print-names    ((TIME NIL NIL)
                 (B.V "(RC B V)" B)
                 (R.V "(RC R V)" R)
                 (R.R "(RC R R)" R)
                 (C.V "(RC C V)" C)
                 (C.C "(RC C C)" C)
                 (C.Q "(RC C Q)" C)
                 (S.V "(RC S V)" S.)
                 (N1.EFFORT_ELECTRICAL "(RC R V1)" N)
                 (R.I "(RC R I)" R)
                 (S.I1 "(RC S I1)" S.)
                 (N2.EFFORT_ELECTRICAL "(RC R V2)" N)
                 (R.I2 "(RC R I2)" R)
                 (C.I "(RC C I)" C)
                 (N3.EFFORT_ELECTRICAL "(RC C V2)" N)
                 (C.I2 "(RC C I2)" C)
                 (B.I2 "(RC B I2)" B)
                 (G.I "(RC G I)" G)
                 (N4.EFFORT_ELECTRICAL "(RC B V1)" N)
                 (B.I "(RC B I)" B)
                 (S.I "(RC S I)" S.)
                 (S.MODE NIL NIL)
                 (R.MODE NIL NIL))

:text           ()

:layout         ()

:other          ((DERIVED-ENERGY-CONSTRAINT
                    :NO-ENERGY-CONSTRAINT)
                 (IGNORE-QDIRS)
```

```
                    (NO-NEW-LANDMARKS)
                    (CC-INFO
                     . (RC (impl UNIQUE)))
                    (CC-MODE-ASSUMPTIONS (S.MODE CLOSED)
                        (R.MODE WORKING)))
)
```

APPENDIX C

GENERIC COMPONENT LIBRARY LISTING

```
;;;  gendefs.lsp
;;;
;;;  GENERIC COMPONENT BASIC LIBRARY
;;;
;;;
;;;  Variables are listed in the form (name (domain type))
;;;
;;;  Valid domain choices are:    unknown
;;;                               electrical
;;;                               mechanical
;;;                               hydraulic
;;;
;;;  Valid types are:             effort
;;;                               flow
;;;                               displacement
;;;                               momentum
;;;                               power
;;;                               constant, no units, no ;;;
 quantity space
;;;
(library-primitive effort-source "effort-source"
    ;; association list pair for component variables
    ;; the list of variable specifications is also an
    ;; association list
    ;;
    ;; key = "c-variables"
    ;; datum = sequence of variable list forms
    ;;
    (c-variables   (SE unknown effort)
    )
    ;; association list pair for terminals
    ;;
    ;; key = "terminals"
    ;; datum = sequence of terminal list forms
    ;;
    (terminals     (T1 180 TERMINAL-IN (E unknown effort)
                                       (F unknown flow))
                   (T2 0  TERMINAL-OUT (E unknown effort)
                                       (F unknown flow))
    )
```

```
    ;; association list pair for component constraints
    ;; key = "constraints"
    ;; datum = sequence of constraint list forms
    ;;
    (constraints    (constant SE)
                    (ADD T1.E SE T2.E)
    )
    ;; association list pair for component restrictions
    ;; key = "constraints"
    ;; datum = sequence of restrictions list forms
    ;;
    (restrictions    (domain-eq T1 T2)
                     (>= SE 0)
    )
    (bitmap "effort.bit")
)
(library-primitive flow-source  "flow-source"
    ;; association list pair for component variables
    ;; the list of variable specifications is also an
    ;; association list
    ;;
    (c-variables  (SF unknown flow)
    )
    ;; association list pair for terminals
    ;; each terminal specification is also an association
    ;; list
    ;;
    (terminals    (T1 180 TERMINAL-IN (E unknown effort)
                                      (F unknown flow))
                  (T2 0  TERMINAL-OUT (E unknown effort)
                                      (F unknown flow))
    )
    ;; association list pair for component constraints
    ;; a general list of lists is used for the constraints
    ;;
    (constraints  (constant SF)
                  (ADD T1.F SF T2.F)
    )
    (restrictions (domain-eq T1 T2)
                  (>= SF 0)
    )
    (bitmap "flow.bit")
)
(library-primitive resistor "resistor"
    (c-variables  (EACROSS unknown effort)

    )
    (terminals    (T1 180 TERMINAL-IN (E unknown effort)
                                      (F unknown flow))
                  (T2 0  TERMINAL-OUT (E unknown effort)
                                      (F unknown flow))
```

```
    )
    (constraints   (M+ EACROSS T1.F)
                   (ADD T2.E EACROSS T1.E)
                   (EQUAL T1.F T2.F)
    )
    (restrictions (domain-eq T1 T2)
                   (>= EACROSS 0)
    )
    (bitmap "resistor.bit")
)
(library-primitive capacitor "capacitor"
    (c-variables   (Q unknown displacement)
    )
    (terminals     (T1 180 TERMINAL-IN (E unknown effort)
                                       (F unknown flow))
                   (T2 0 TERMINAL-OUT (E unknown effort)
                                      (F unknown flow))
    )
    (constraints   (M+ T1.E Q)
                   (d/dt Q T1.F)
    )
    (restrictions (domain-eq T1 T2)
    )
    (bitmap "capacitor.bit")
)
(library-primitive inertia "inertia"
    (c-variables   (P unknown momentum)
    )
    (terminals     (T1 180 TERMINAL-IN (E unknown effort)
                                       (F unknown flow))
                   (T2 0  TERMINAL-OUT (E unknown effort)
                                       (F unknown flow))
    )
    (constraints   (M+ T1.F P)
                   (d/dt P T1.E)
    )
    (bitmap "inertia.bit")
)
(library-primitive gyrator "gyrator"
    (c-variables   (P unknown momentum)
    )
    (terminals     (T1 150 TERMINAL-IN (E unknown effort)
                                       (F unknown flow))
                   (T2  210 TERMINAL-OUT (E unknown effort)
                                         (F unknown flow))
                   (T3  30 TERMINAL-IN (E unknown effort)
                                       (F unknown flow))
                   (T4 330 TERMINAL-OUT (E unknown effort)
                                        (F unknown flow))
    )
```

```
    (constraints  (M+ T1.F P)
                  (d/dt P T1.E)
                  (EQUAL T1.F T2.F)
                  (EQUAL T1.E T2.E)
    )
    (bitmap "gyrator.bit")
)
(library-primitive transformer "transformer"
    (c-variables  (P unknown momentum)
                  (TPOWER unknown power)
    )
    (terminals    (T1 150 TERMINAL-IN (E unknown effort)
                                      (F unknown flow))
                  (T2  210 TERMINAL-OUT (E unknown effort)
                                        (F unknown flow))
                  (T3  30 TERMINAL-IN (E unknown effort)
                                      (F unknown flow))
                  (T4 330 TERMINAL-OUT (E unknown effort)
                                       (F unknown flow))
    )
    (constraints  (MULT M T1.F T2.F)
                  (MULT M T2.E T1.E)
                  (MULT T1.E T1.F TPOWER)
                  (MULT T2.E T2.F TPOWER)
    )
    (bitmap "transformer.bit")
)
(library-primitive parallel-junction "parallel-junction"
    (c-variables (F unknown flow)
                 (E unknown effort)
    )
    (terminals    (T1 180 TERMINAL-NONE (E unknown effort)
                                        (F unknown flow))
                  (T2 0 TERMINAL-NONE (E unknown effort)
                                      (F unknown flow))
                  (T3 90 TERMINAL-NONE (E unknown effort)
                                       (F unknown flow))
                  (T4 270 TERMINAL-NONE (E unknown effort)
                                        (F unknown flow))
    )
    (constraints  (EQ T1.F T2.F T3.F T4.F)
                  (ADD T1.E T2.E T3.E T4.E E)
                  (EQ E 0)
    )
)
```

APPENDIX D

CONSTRUCTING AN EXAMPLE MODEL

Physical System Description

In the mechanical domain, a commonly recognized physical system is the drive train of an automobile. The system involves the engine, a power source, and a series of linkages from the engine through the driveshaft, differential, and axle to the wheels. With appropriate selections for initial states, a qualitative model of this system can be used to answer questions such as "If the friction on the tires is decreased, what will happen to the torque on the driveshaft?" and "What effects will increasing the rotational speed of the driveshaft have on the differential?".

Constructing the Model with GMBS

The first step in constructing the model using GMBS is to select NEW from the Model Menu. When the text prompt for model name appears, the name "Drive-Train-Model" is entered. When the selection list prompt for default model domain appears, the domain "mechanical" is selected. The

display area is cleared and the model building system is ready for components to be added to the model.

Using the Select/Library option from the Component Menu, the library component "effort-source" is selected to represent the engine. The component icon is placed near the left side of the display area at position (1, 4). Next, the library component "inertia" is selected to represent the turning of the driveshaft and is placed at position (3, 3). There is also a resistance applied to the driveshaft so a "resistor" component is selected and placed at position (4, 3). Using the Series toolbar button, the inertia and the resistor are connected in series.

In a similar manner, a "transformer" component is selected to represent the converting effects of the differential, two more "transformer" components are selected to represent the conversion from the axle to the rotation of the wheels. Each wheel is then represented by a series combination of "resistor" and "inertia".

Using the Parallel toolbar button, two "parallel-junction" components are added to represent the parallel combination of the two wheel sections. The effects of stored energy from road elevation and resistance from air and friction are represented by adding a series combination of a "capacitor" component, a "resistor" component, and an "inertia" component in parallel with the

wheel sections. The display of the completed model appears as shown in Figure 10.

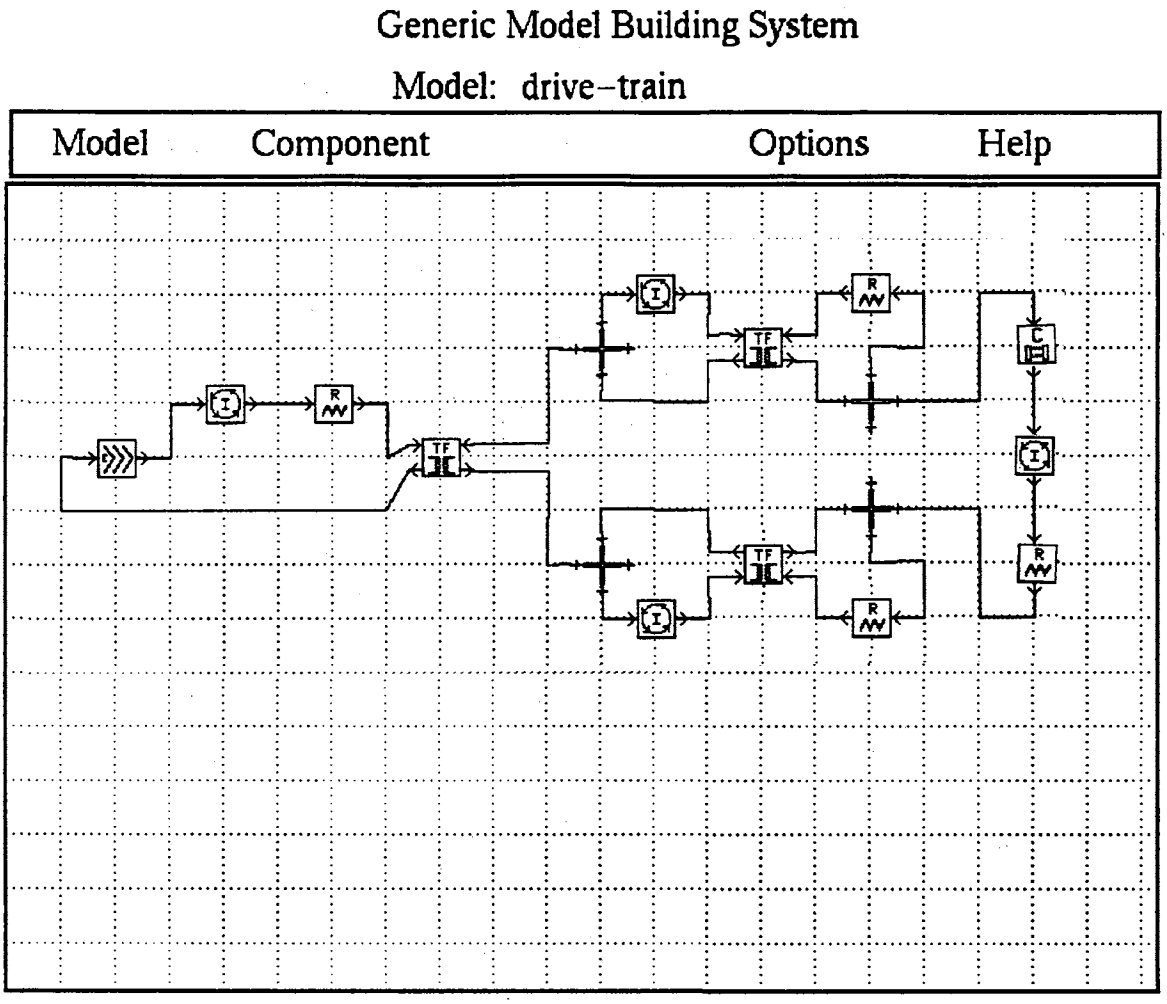


Figure 10. Drive Train Model Display

VITA 2

Esther L. Davis

Candidate for the Degree of

Doctor of Philosophy

Thesis: GENERIC COMPONENT REPRESENTATION FOR EFFICIENT QUALITATIVE MODEL BUILDING

Major Field: Computer Science

Biographical:

Personal Data: Born in Lawton, Oklahoma, September 22, 1960, the daughter of Robert G. and Alma L. Davis.

Education: Graduated from Douglas MacArthur Senior High School, Lawton, Oklahoma, in May 1978; received Bachelor of Arts Degree in Math and Physics from Cameron University in May 1982; received Master of Science degree in Computer Science from Oklahoma State University in May 1992; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in May, 1994.

Professional Experience: Programmer/Analyst, Random House Publishers, June 1983 to December 1985; Programmer I, Oklahoma College of Osteopathic Medicine and Surgery, February 1986 to December 1986; Data Processing Manager, OCOMS, December 1986 to March 1988; Director of Computing and Telecommunications, COM-OSU, March 1988 to January 1990; Programmer/Analyst, Blue Cross & Blue Shield of Oklahoma, January 1990 to July 1990; Teaching Assistant, Computer Science Department, Oklahoma State University, August 1990 to July 1993.