Component Connection Models*

David W. Franke, Daniel L. Dvorak†

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

The relation between *part* and *whole* is the key to describing the structure of a mechanism. Different modeling methods have different concepts of what should count as a "part" of a system, and how the parts should relate to each other. The mathematical, differential-equation-based approach to modeling taken in QSIM essentially says that the "parts" of a mechanism are the continuous variables that characterize its state, and their relations are mathematical constraints inherited from the physical structure of the system.

However, a physical system frequently consists of a set of components that relate through explicit connections (a form of description that is frequently more meaningful to a domain expert than the differential equations). This paper describes CC, a model-building program that accepts a component-connection description of a physical system and translates it to the qualitative differential equations of QSIM. CC provides facilities for component abstraction and hierarchical component definition, raising the level of abstraction for modeling via QSIM. Component *modes* can be specified, and are translated into QSIM operating regions. CC uses the general variable types of bond graphs (a technique for dynamic physical system modeling). Finally, this component-connection paradigm provides the framework for information utilized in other model-based reasoning tasks such as diagnosis.

1 Introduction

The component-connection paradigm [Abelson, Sussman 1985], [de Kleer 1985] provides a declarative, conceptually uniform approach to specifying hierarchical system models. This paradigm states that models are constructed from (less complex) components and connections among terminals of these components. Further, connections among terminals (more precisely, variables or parameters associated with each terminal) are the only manner in which components of the model can interact. We describe the CC system [Franke, Dvorak 1989] which supports the definition of models via the component-connection paradigm in conjunction with the QSIM system [Kuipers 1985], [Kuipers 1986]. CC supports the definition of components in terms of QSIM primitives and the construction of models from these primitives. Model construction in CC uses the notion of variable types for physical systems [Rosenberg, Karnopp 1983] in conjunction with conservation laws (*cf.* [Rosenberg, Karnopp 1983], [de Kleer, Brown 1985], [de Kleer 1985]). A component definition can specify several modes which can be either static (e.g. Working, Broken) or dynamic (determined by a model variable).

Section 2 discusses the component-connection paradigm. Section 3 discusses the concepts of CC. Sections 4 and 5 describe component and configuration definitions, respectively. Section 6 describes future work planned for CC.

2 Component Connection Models

Many systems of interest allow models composed of components and connections among these components. Further, these components need not themselves be primitives of the domain, and can be decomposed into less complex (possibly primitive components) and associated connections. This representation is common in Computer Aided Design (CAD) systems for mechanical and electrical design. Models specified in this way provide a declarative specification of a design which can be used for multiple purposes.

A system modeling the flow of fluids will involve tanks and valves, and a system modeling an electrical circuit will involve capacitors, resistors, and transistors. Each occurrence of one of these components will introduce variables and constraints which can be characterized once, but require unique instantiations for each occurrence of the component. The ability to characterize a component once provides a mechanism where complex models can be described more concisely than enumerating the variables and constraints for each occurrence of the component. To complete the model, the connections between components must be specified, and any implications of these connections must be considered. In particular, components may interact only at these connections, and constraints which enforce conservation of flow at a connection must be added.

*This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grant IRI-8602665 and by NASA grants NAG 2-507 and NAG 9-200.

†Supported by the AT&T Doctoral Support Program.

3 CC Concepts

Two important high-level concepts of model definition in CC are *hierarchical component definitions* and *component abstraction*. Hierarchical component definitions permit components to be defined in terms of (less complex) components and connections between these components. A component definition expressed in terms of other components is called *composed*. These recursive definitions terminate with a component defined in terms of QSIM primitives, namely variables and constraints among these variables. A component definition expressed in terms of QSIM constraints and variables is called *primitive*.

The component abstraction concept is realized by requiring a component interface definition independent of any definition of the implementation of the component, either a composed or a primitive definition. A component interface declaration defines the component type name and the terminals and parameters of the component type. One or more component implementations can then be defined for a component interface, using either primitive definitions, composed definitions, or both. This distinction of interface and implementation is modeled after the entity and architecture constructs of the VHSIC Hardware Description Language (VHDL) [VHDL 1987].

Given the ability to define multiple implementations of a component interface, a model builder may wish to reference a component interface (i.e. a component type) in a composed definition without specifying a particular implementation for that component instance. This allows the modeller to delay the selection of a particular implementation from model definition time to model building time. The construct through which such implementations are specified is called a *configuration*. (The configuration construct is also borrowed from VHDL.) This notion of configuration is common in electrical CAD systems, allowing alternative implementations and parameter changes to be easily specified.

The final concept described here is that of general variable types for physical systems, taken from [Rosenberg, Karnopp 1983]. The *bond graph* modeling approach described in [Rosenberg, Karnopp 1983] is a technique for describing dynamic physical systems independent of their domain (e.g. electrical, mechanical, or hydraulic). Hence, the approach also characterizes the model variables in a domain independent manner. CC types model variables in this way, with the following types:

- Effort
- Flow
- Momentum (the time integral of Effort)
- Displacement (the time integral of Flow)
- Energy
- Resistance
- Capacitance

To facilitate component definitions for particular domains, CC provides mappings between these general variable types and domain specific type names, such as *voltage* as the electrical domain specific name for effort. The associated general types are important in CC in that the processing of connections requires knowledge of variable types for such things as applying the conservation of flow law. In CC, the variable type names for the electrical domain might be mapped as follows:

```
(effort        voltage        "V")
(flow          current        "I")
(resistance    resistance     "R")
(capacitance   capacitance    "C")
(displacement  charge         "Q")
(inductance    inductance     "L")
(momentum      flux-linkage   "Fl")
```

4 Component Definition

A component definition is comprised of an interface and one or more implementations. A component definition can be subsequently referenced in other component definitions to provide hierarchical definitions. The content of these definitions is described below, with examples of each.

4.1 Component Interface

Component interface definitions define a component type and terminals and parameters of the type's interface and the domain (e.g. electrical) of the component. Additional model information such as terminal direction (e.g. input) can be specified in the interface definition. Below is the interface definition for a resistor component in the electrical domain.

```
(define-component-interface resistor "Resistor"
  electrical
  (terminals t1 t2))
```

4.2 Component Implementation

Component implementations can be either primitive (expressed in QSIM primitives) or composed (expressed in terms of other components. The primitive implementation is described and an example of the resistor is given. The composed implementation is also described and an example using the resistor component is given.

Primitive Implementation

A primitive component implementation consists of variable and constraint declarations which an instance of the component introduces into the model. Variables declared for a component are associated either with a terminal of the component or with the component itself, and are called *terminal variables* and *component variables*, respectively. The names of terminal and component variables must be unique for a component,

since both terminal variables and component variables can be referenced in the constraints of the component. Information required for each variable declaration is the variable name (a symbol) and the variable type (a symbol corresponding to the appropriate name for the domain).

In the domain of fluid flows, assuming that a tank is defined as a component, examples of *terminal variables* would be Inflow at the input terminal and Outflow at the drain terminal. In the domain of electrical circuits, assuming that a resistor is defined as a component, examples of terminal variables would be V1, the voltage at one terminal, and V2, the voltage at the other terminal. Variables which correspond to Flow in the domain of interest are typically defined as terminal variables. Depending on the component being defined, variables corresponding to Flow may also be defined as component variables.

Component variables correspond to quantities of interest for the component, and are related to other component variables and to terminal variables via constraints. For example, in the fluid flow domain, a tank would have terminal variables Inflow and Outflow, and a component variable Netflow, which is the difference of the Inflow and Outflow. In the electrical domain, a resistor has component variables Voltage, Resistance, and Current. Voltage interacts with the terminal variables V1 and V2 (the definition of voltage across a resistor), and also with the component variables Resistance and Current (Ohm's Law).

4.2.1 Constraints

The constraints introduced by a component's inclusion in a model are specified in the same syntax as required by the Define-QDE form (i.e. they are QSIM specific). Constraints can be partitioned into groups which represent *modes* of the component. Modes can be static (e.g. Working or Broken) in which case the component is characterized by one set of constraints throughout simulation. Modes can also designate dynamic characterizations of the component, with the active constraints for the model determined by the value of a model variable. These dynamic modes are realized via QSIM operating regions.

Following is an example component definition from the electrical domain.

```
(define-component-implementation
  primitive resistor "Resistor in primitives"
  (terminal-variables
    (t1 (v1 voltage)
        (i  current))
    (t2 (v2 voltage)
        (i2 current)))
  (component-variables
    (v voltage display)
    (r resistance independent))
  (constraints
    ((ADD v v2 v1))    ; Voltage across terminals
    ((MULT i r v))     ; Ohm's Law
    ((MINUS i i2) (minf inf) (0 0) (inf minf))))
```

The definition of two *static* modes of a resistor, `Working` and `Shorted`, are as follows:

```
(define-component-implementation
  ...
  (constraints
    (working nil
      ...)                    ; as before
    (shorted nil
      ((= v2 v1))             ; Voltage = at terminals
      ((= v 0))               ; No voltage drop
      ((MINUS i i2) (minf inf) (0 0) (inf minf))
    )))
```

The definition of *dynamic* modes is accomplished via a condition on a variable. For example, suppose the resistor model changes when the current (`i`) is greater than the landmark `I*`. The associated dynamic modes are specified as:

```
(define-component-implementation
  ...
  (constraints
    (normal (<= i I*)
      ...)
    (abnormal (> i I*)
      ...)))  ; Corresponding constraints
```

Composed Implementation

A composed implementation consists of (sub)component and connection declarations which an instance of the component introduces into the model. Each (sub)component must be uniquely named within the component implementation definition, and references the component type of which it is an instance. Additionally, a (sub)component instance declaration can specify information (such as the quantity space) for variables introduced by the instance. Given component definitions for a battery and a capacitor, the definition of a series resistor-capacitor circuit can be specified as follows:

```
(define-component-interface RC
  "Resistor-capacitor circuit")

(define-component-implementation composed RC
  "Composed resistor-capacitor circuit" electrical
  (components
    (B battery (landmarks (v V*) (v1 V*)
                          (v2 (0)) (i (0 inf)))
               (initable i))
    (R resistor (landmarks (r R*)))
    (C capacitor (landmarks (c C*))
                 (initable q)))
  (connections
    (w1 (b t1) (r t1))
    (w2 (r t2) (c t1))
    (w3 (c t2) (b t2))))
```

4.2.2 Generated Variables and Constraints

A model variable of type effort is generated for each connection to represent the equivalence class of terminal variables of type effort associated with a connection. The flow variables for terminals involved in a connection are constrained by the flow conservation law (Kirchhoff's Current Law, in the electrical circuit domain). To add the constraint representing flow conservation, CC adopts the convention that flows are positive into the component. It is important to keep this convention in mind when specifying the constraints (involving terminal variables of type flow) in primitive implementations of components.

No explicit constraint representing Kirchoff's Voltage Law (sum of voltage drops in a loop is zero) is added by CC. Rather, this law is implicitly enforced by QSIM's constraint propagation if every terminal has an associated effort variable, and effort variables of a each component instance are appropriately related (e.g. Ohm's Law for a resistor, V = IR, and V = V1 - V2).

5 Configuration Definition

A configuration defines particular implementations for component instances in the model. This allows the model builder to succinctly specify alternative models. An implementation can be specified for all instances of a component type, or for specific instances of a component type. In addition to component implementations, a configuration can specify a mode and parameter values for all instances or specific instances of a component type. A configuration is associated with either a component interface or with a component implementation. For example, consider a circuit model which contains two instances of `resistor`, such as the RC ladder network in [Williams 1985]. In this model, the resistor instance names are `R1` and `R3`. A configuration specifying static modes for these instances would be:

```
(define-component-configuration RC-Ladder-Config1
  "RC ladder, R1 working, R3 shorted" RC-Ladder
  (r1 (impl primitive) (mode working))
  (r3 (impl primitive) (mode shorted)))
```

Specific implementation and/or mode information concerning other (sub)components of the can be given also. An alternative configuration which specifies a mode for all (sub)components of type resistor would be:

```
(define-component-configuration RC-Ladder-Config1
  "RC ladder, all resistors working" RC-Ladder
  (resistor (impl primitive) (mode working)))
```

6 Future Work

6.1 Hierarchy

CC takes a hierarchical description and generates a "flat" description of the system. The description is termed "flat" since it expresses the system in QSIM primitives, namely variables and constraints. Given that a system can be described hierarchically, it is interesting to consider simulating the system without completely decomposing the hierarchy. One approach [Christian 1987] considers a component as an arbitrary constraint. Time scale abstraction [Kuipers 1987] also addresses simulation of a hierarchy.

6.2 Quantitative Information

The mixed qualitative-quantitative reasoning system Q2 [Kuipers, Berleant 1988] makes it possible to incorporate incomplete quantitative information into the derivation of behaviors. Specification of this quantitative information via a CC model description will extend the breadth of models comprehended by CC.

6.3 A Language for Model Specification

Currently, CC is specific to QSIM and addresses those aspects of a model which are required to develop the qualitative differential equation definition used by QSIM. We plan to examine modifications to the syntax of CC so that models specified in this syntax can be 1) mapped to other simulation environments (with appropriate changes in primitives) and 2) used for other purposes such as causal reasoning for model-based troubleshooting or teleological reasoning [Franke 1989].

7 Conclusions

The component-connection paradigm provides a declarative, conceptually uniform approach to specifying hierarchical system models. The initial implementation of CC supports this model definition paradigm and provides a means for translating a structural model into a computational model. When extended in the ways described above, this model definition language will support other model-based reasoning tasks, as well as other underlying computational languages.

Acknowledgements

This work is progressing under the guidance of Prof. Benjamin Kuipers. Valuable comments and suggestions have also been contributed by the qualitative reasoning community at the UT-Austin AI Lab, especially Adam Farquhar and David Throop.

References

[Abelson, Sussman 1985] Harold Abelson, Gerald J. Sussman, *The Structure and Interpretation of Computer Programs*, MIT Press, 1985.

[Christian 1987] Jim Christian, "Component Hierarchies for QSIM", Dec. 1987.

[de Kleer 1985] Johan de Kleer, "How Circuits Work", in *Qualitative Reasoning About Physical Systems*, Daniel G. Bobrow, ed., (The MIT Press, Cambridge, MA 1985), pp. 205-280. Reprinted from *Artificial Intelligence* Vol. 24, 1984.

[de Kleer, Brown 1985] Johan de Kleer, John Seely Brown, "A Qualitative Physics Based on Confluences", in *Qualitative Reasoning About Physical Systems*, Daniel G. Bobrow, ed., (The MIT Press, Cambridge, MA 1985), pp. 7-83. Reprinted from *Artificial Intelligence* Vol. 24, 1984.

[Franke 1989] David W. Franke, "Representing and Acquiring Teleological Descriptions" in *Proceedings of the IJCAI-89 Workshop on Model-Based Reasoning*, 1989.

[Franke, Dvorak 1989] David W. Franke, Daniel L. Dvorak, "CC: Component Connection Models for Qualitative Simulation, A User's Guide", AI technical report (forthcoming), Dept. of Computer Sciences, University of Texas at Austin.

[Kuipers 1985] Benjamin J. Kuipers, "Commonsense Reasoning about Causality: Deriving Behavior from Structure", in *Qualitative Reasoning About Physical Systems*, Daniel G. Bobrow, ed., (The MIT Press, Cambridge, MA 1985), pp. 169-203. Reprinted from *Artificial Intelligence* Vol. 24, 1984.

[Kuipers 1986] Benjamin J. Kuipers, "Qualitative Simulation", in *Artificial Intelligence*, Vol. 29, No. 3, (September 1986), pp. 289-338.

[Kuipers 1987] Benjamin Kuipers, "Abstraction by Time-Scale in Qualitative Simulation", in Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, July 1987, pp. 621-625.

[Kuipers, Berleant 1988] Benjamin Kuipers, Daniel Berleant, "Using Incomplete Quantitative Knowledge in Qualitative Reasoning", in Proceedings of the Seventh National Conference on Artificial Intelligence, Saint Paul, August 1988, pp. 324-329.

[Rosenberg, Karnopp 1983] Ronald C. Rosenberg, Dean C. Karnopp, Introduction to Physical System Dynamics, McGraw-Hill, 1983.

[VHDL 1987] IEEE Standard VHDL Language Reference Manual, IEEE Std. 1076-1987.

[Williams 1985] Brian C. Williams, "Qualitative Analysis of MOS Circuits", in *Qualitative Reasoning About Physical Systems*, Daniel G. Bobrow, ed., (The MIT Press, Cambridge, MA 1985), pp. 281-346. Reprinted from *Artificial Intelligence* Vol. 24, 1984.