19930/6883

N93-26072

A STUDY OF MAPPING EXOGENOUS KNOWLEDGE REPRESENTATIONS INTO CONFIG

Final Report

NASA/ASEE Summer Faculty Fellowship Program - 1992

Johnson Space Center

Prepared by:	Blayne E. Mayfield
Academic Rank:	Assistant Professor
University and Department:	Oklahoma State University Computer Science Department Stillwater, Oklahoma 74078-0599

NASA/JSC

Directorate:	Engineering
Division:	Automation and Robotics
Branch:	Intelligent Systems
JSC Colleague:	Jane T. Malin
Date Submitted:	August 4, 1992
Contract Number:	NGT-44-005-803

ABSTRACT

Qualitative reasoning is reasoning with a small set of qualitative values that is an abstraction of a larger and perhaps infinite set of quantitative values. The use of qualitative and quantitative reasoning together holds great promise for performance improvement in applications that suffer from large and/or imprecise knowledge domains. Included among these applications are the modeling, simulation, analysis, and fault diagnosis of physical systems.

Several research groups continue to discover and experiment with new qualitative representations and reasoning techniques. However, due to the diversity of these techniques, it is difficult for the programs produced to exchange system models easily. The availability of mappings to transform knowledge from the form used by one of these programs to that used by another would open the doors for comparative analysis of these programs in areas such as completeness, correctness, and performance.

A group at the Johnson Space Center (JSC) is working to develop CONFIG, a prototype qualitative modeling, simulation, and analysis tool for fault diagnosis applications in the U.S. space program. The availability of knowledge mappings from the programs produced by other research groups to CONFIG may provide savings in CONFIG's development costs and time, and may improve CONFIG's performance. The study of such mappings is the purpose of the research described in this paper.

Two other research groups that have worked with the JSC group in the past are the Northwest University Group and the University of Texas at Austin group. The former has produced a qualitative reasoning tool named SIMGEN, and the latter has produced one named QSIM. Another program produced by the Austin group is CC, a preprocessor that permits users to develop input for eventual use by QSIM, but in a more natural format. CONFIG and CC are both based on a component-connection ontology, so a mapping from CC's knowledge representation to CONFIG's knowledge representation was chosen as the focus of this study.

A mapping from CC to CONFIG was developed. Due to differences between the two programs, however, the mapping transforms some of the CC knowledge to CONFIG as documentation rather than as knowledge in a form useful to computation.

The study suggests that it may be worthwhile to pursue the mappings further. By implementing the mapping as a program, actual comparisons of computational efficiency and quality of results can be made between the QSIM and CONFIG programs. A secondary study may reveal that the results of the two programs augment one another, contradict one another, or differ only slightly. If the latter, the qualitative reasoning techniques may be compared in other areas, such as computational efficiency.

INTRODUCTION

Qualitative Reasoning

Quantitative reasoning is reasoning with precise, usually numeric, values. Applications such as expert systems, modeling, and simulation typically have been developed using quantitative reasoning. Due to the combinatorial and sometimes imprecise nature of the data in these applications, however, their success has been limited by the overwhelming amount of computation necessary to obtain the desired results. Qualitative reasoning can help ease this problem.

Qualitative reasoning can be used to reason over the same domains as quantitative reasoning, but a domain is represented as a (usually) small set of qualitative values rather than a large (and possibly infinite) set of quantitative values. This abstraction of a domain must be chosen carefully so that the set of qualitative values represents the important qualities of the domain while suppressing or ignoring other qualities. If the quantitative domain is continuous, the qualitative values chosen usually represent contiguous intervals of the continuous space. For example, a number line can be represented quantitatively by the real numbers; however, in some applications the same number line could be represented by the qualitative values "negative", "zero", and "positive". Since the number of elements in a qualitative domain is smaller than the number in the corresponding quantitative domain, combinatorial problems can be solved with less effort. A good introduction to qualitative reasoning in physical systems can be found in [6].

Although qualitative reasoning methods are still mostly experimental, their application to modeling, simulation, and analysis of physical systems holds great promise for use in fault diagnosis of those systems. Using qualitative reasoning techniques, fault diagnosis can be performed not only after a physical system has been built, but also during each phase of its design. The potential savings in time, effort, and money are enormous. Research groups, such as those headed by Forbus [2,4,7,8] and Kuipers [5,9,11,12], have developed experimental qualitative reasoning techniques and systems. A Johnson Space Center group headed by Dr. Jane Malin is developing a prototype qualitative reasoning program named CONFIG [13,14] for use in model-based fault diagnosis applications within the U.S. space program.

Goals and Rationale

These groups have taken (sometimes radically) different approaches to qualitative modeling, simulation, and analysis. Vigorous research is necessary to advance qualitative reasoning techniques to a level where they can be applied to solve a wide variety of real-world problems. However, sometimes the diverse methods utilized by different groups present problems: different knowledge representation schemes can make it difficult to share knowledge between groups, and incompatible reasoning techniques can make it difficult to incorporate new techniques into existing programs. The thrust of this project is to study the former, and perhaps to gain insight into the latter.

The major goal of this project is to develop a mapping from the knowledge representations used by the Kuipers group (in the CC [9] and QSIM [5] systems) to that used by the Malin group (in the CONFIG system). A program could be developed from this mapping to mechanize the transformation of CC and/or QSIM knowledge to a form compatible with CONFIG. In addition to facilitating the sharing of physical systems models between research groups, such a program may serve as the basis for comparisons between reasoning methods used by the programs.

It is important that the mapping produced be correct and complete; i.e., all of the knowledge in the source representation should be transformed in such a way as to accurately retain its original meaning, and only that knowledge should be transformed (nothing should be added). This presents a problem when the target representation contains no counterpart for particular elements of knowledge embodied in the source representation. It was decided that these elements and all documentation elements would be transformed to documentation in the target representation, and all other knowledge would be transformed to its counterpart in the target representation.

CONFIG

General Description

CONFIG is a system that integrates object-oriented modeling, quantitative and qualitative mathematics, discrete event simulation, and digraph analysis to support the modeling, simulation, and analysis of physical systems. The initial application of CONFIG will be model-based fault diagnosis. CONFIG is based on a component-connection ontology [1,3]; i.e., a physical system is modeled as a collection of components and the connections between them. A component-connection model is extremely modular and, thus, can be modified easily. A CONFIG model can be viewed as a directed graph in which components, called devices, are the graph nodes and connections, called relations, are the graph edges. During simulation, changes in the value of device variables are propagated along the edges of the graph (i.e., through the component-to-component connections).

Knowledge Representation

Each device class is comprised of several elements (or parts), some of which may also be made up of parts. This arrangement of parts within parts is called a parts hierarchy. Figure 1 shows the parts hierarchy of a CONFIG device. The parts of a device are the internal variable clusters (VCs), port VCs, mode independent (MI) processes, a mode transition digraph (MTD), and a Device Control Digraph (DCD). In turn, the MTD is made up of modes, and modes are composed of mode dependent (MD) processes and mode transition (MT) processes. These parts are detailed in [13] and [14].

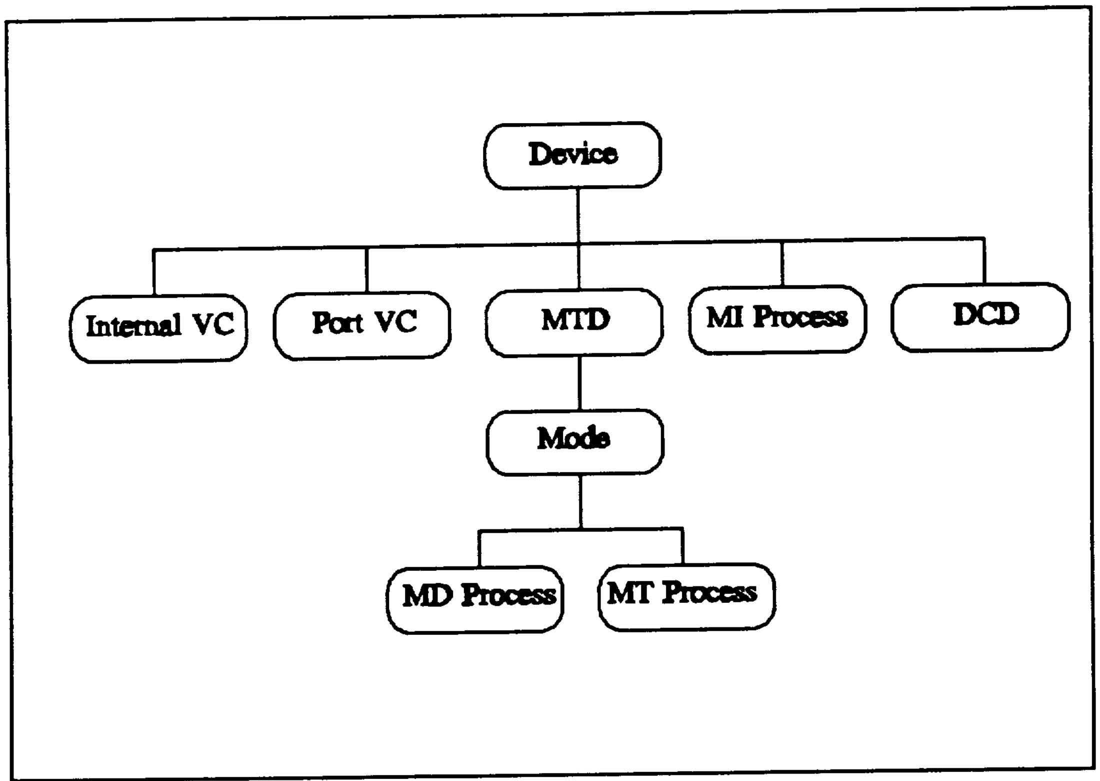

Figure 1.- Parts hierarchy of a CONFIG device class.

The purpose of a variable cluster (VC) is to aggregate logically related variables into a common data structure. The variables in the internal and port VCs contain state information about the device. An internal VC is similar to a private structure in a C++ class: it contains variables that are accessible only by the other parts of the device. A port VC, by contrast, is designed to be a device interface to other devices.

MI processes are one of three types of processes defined for use in discrete simulation (the others are described below). Semantically, a process has the general form

if *precondition* then *action* after *time-delay*.

If the precondition of a MI process becomes true during simulation, the action specified is placed on the simulation schedule to be executed after the time delay specified. The preconditions are checked each time the simulation clock is updated.

Conceptually, a device mode defines an operating region of the device and describes the behavior of the device when it is in that mode. By defining modes for each operating region of the device, the behavior of the device can be modeled completely. For example, the modes for a switch device might be named "open", "closed", and "stuck" and describe how a switch behaves in each mode. The modes of an MTD describe a digraph in which the nodes are mode objects and the edges are MT processes. MT processes have the same general semantic form as MI processes. The MTD also contains a variable that indicates the current device mode. Each time the simulation clock is updated, the preconditions of the MT processes of the current mode are checked. If the precondition of one of the MT processes is true, then the corresponding action, which is a transition to another mode, takes place. The MTD also contains a collection of MD processes; these processes operate much like the MI processes, with the exception that their preconditions are checked only when the mode in which they are defined is the current mode.

The purpose of a DCD is to describe the submodel (i.e., subcomponent) structure within a device class.

A relation (i.e., an inter-device connection) can be thought of as a cable or pipe between a subset of the port VCs of two devices. The flow of data through the relation is generally one-way, though there are two-way relations (these can be viewed as two one-way relations). A relation may be defined to connect any two devices, and within these devices, any subset of their port VCs, and within the port VCs, any subset of the variables.

Simulation

CONFIG performs discrete event simulation. In discrete event simulation, events are assumed to occur at points in time rather than continuously over intervals of time. When the simulation begins, a sequence of events is placed in a schedule, ordered chronologically by their arrival times (i.e., the times at which the events will occur). The simulation clock is advanced to the arrival time of the first event, and the event takes place. As a consequence of event occurrence, other events may be scheduled. The simulation clock is advanced to the next event arrival time, and the process continues until the schedule is empty (or until a specified amount of clock time has passed). The purpose of using discrete event simulation is to simulate a numerical sample of the set of possible system state sequences, rather than generating all of them. (The method of generating all sequences is the basis of envisionment, a form of which is used in QSIM).

When CONFIG begins a simulation, each device is set to its specified initial mode. As the simulation progresses, the process conditions are checked, and if they are true, their actions are scheduled for execution. Data values are propagated from device port to device port through relations.

QSIM

General Description

QSIM [5] is a qualitative simulation system for model-based reasoning produced by the Kuipers research group at University of Texas at Austin. Although still considered an experimental system, QSIM has matured to the point that other research groups use it as a basis for their work. The modeling approach used in QSIM is radically different from that used in CONFIG. In QSIM, a system model is represented by a single qualitative differential equation (QDE)[1]. Contrast this to CONFIG, in which a system is modeled as a collection of components and connections. Furthermore, in QSIM it is possible to have multiple models of the same physical system, and to switch between models as needed during simulation. For example, the QSIM User's Guide [5] presents an example of modeling a bouncing ball; the system is modeled as an accelerating mass while in flight and as a spring when it rebounds the floor. QSIM's use of multiple models and model switching is similar in concept to CONFIG's use of device modes and mode transitions.

Knowledge Representation

A QDE specifies the structure of the system being modeled. A QDE is comprised of four major parts: a set of qualitative variables, a quantity space for each variable, a set of constraints among the variables, and a set of transition rules. Details about these may be found in [5].

A quantity space is a qualitative abstraction of a quantitative range. It consists of an ordered sequence of landmark values within the range; landmarks are borders between qualitative intervals. A QSIM user can specify any collection of landmarks as long as the collection includes the landmark 0 (zero).

The value of a qualitative variable is comprised of a qualitative magnitude, which is a value from the variable's quantity space, and the direction of the magnitude's change, which can either be decreasing, steady, or increasing.

A constraint specifies important relationships among the variables of the QDE. It consists of the constraint specification and an optional list of corresponding values. A constraint specification is a relation name and its variable arguments. A corresponding value is a list of variable values that further constrain the variables. For example,

((M+ A B) (0 0) (inf inf))

[1] A QSIM QDE construct is actually a set of qualitative differential equations, in the traditional sense of the term. However, to maintain consistency with QSIM documentation, the construct will continue to be referred to as a QDE for the remainder of this paper.

is a constraint in which "(M+ A B)" is a constraint specification, and "(0 0)" and "(inf inf)" are corresponding values. This constraint states that the relation M+ (positive monotonicity) holds between the variables "A" and "B". Further, the corresponding values state that when "A" is zero, "B" is zero, and when "A" is $+\infty$, "B" is $+\infty$.

Transition rules provide a mechanism for switching between models during simulation. Each transition rule consists of a condition and a QDE identifier; if the QDE containing the transition rule is active and the condition is true, the QDE specified in the rule becomes the active QDE.

Simulation

A simulation in QSIM differs drastically from one performed in CONFIG since QSIM uses a form of envisionment to perform simulation. Envisionment, as described in the section on CONFIG, is a process by which all possible system state sequences are generated. Beginning with the initial model in its initial state, QSIM generates a tree containing these state sequences, although QDE constraints and other constraining mechanisms are used to prune away those state sequences that violate constraints, thus saving time and computation. This tree is called a behavior tree. The user can request information about any path in the tree from QSIM. Even with pruning, though, building a behavior tree can be a computationally expensive process.

QSIM to CONFIG Mapping

The desirability of mapping QSIM knowledge to CONFIG was examined early in this study. As a result of this examination, it was decided that a QSIM-to-CONFIG mapping would not be developed. QSIM models all devices of the system in a single QDE. A QDE cannot reliably be separated into discrete devices. Thus, a QSIM model would be mapped to CONFIG as a single "mega-device". One of CONFIG's strong points is its component-connection representation. This representation is extremely modular; components and connections can easily be added or deleted. This modularity would be lost in the mapping.

CC

General Description

CC [9] is a preprocessor for QSIM; it takes CC input language as its input, and produces QSIM input language as its output. CC, like CONFIG, is based on a component-connection ontology. The CC input language is styled after that of VHDL [10], which separates a component's interface definition from its implementation definition so that multiple component implementations can be defined without the necessity of replicating the interface information. CC elements can be arranged into part and inheritance hierarchies, even though QSIM does not support hierarchies; this is because CC "flattens out" the hierarchies.

Knowledge Representation

There are four major modeling elements in CC: quantity spaces, component interfaces, component implementations, and component configurations. Details about these elements can be found in [9].

Quantity spaces can appear either globally or within a component interface or implementation. The CC quantity space concept is an extension of the QSIM quantity space concept. Both contain an ordered list of qualitative landmark values, but a CC quantity space may also specify a parent quantity space and a list of conservation correspondences. A conservation correspondence is a list of qualitative values from the quantity space that sum to zero; thus, a conservation correspondence is a form of constraint on the quantity space values.

A CC component class definition is comprised of two parts: a component interface and a component implementation. A component interface is a high-level abstraction of a component that describes a family of component classes. The specific details of particular component classes are described by component implementations. Multiple implementations may be defined for a given interface. Each interface/implementation pair defines a component class. A component class with subcomponents is called a composed component class, and a component class without subcomponents is called a primitive component class.

The parts of a component class (i.e., an interface/implementation pair) include a knowledge domain name, terminals, component (local) variables, mode variables, subcomponents, and connections between the subcomponents.

The knowledge domain name is one of a list of predefined areas such as "hydraulic", "electrical", etc. A knowledge domain is a list of variable types commonly used in components within the domain; for example, the "hydraulic" domain contains variable types such as "flow" and "pressure". The authors of CC have taken a bond graph approach [15], so these domain-specific variable types simply are synonyms for a set of generic type names.

A terminal is an interface that can be connected to a terminal of another device. It contains a collection of terminal variables. Terminal and component variables are variables like those found in QSIM; i.e., their values are composed of a qualitative magnitude and a direction of change. A mode variable is a QSIM variable and a collection of condition/value pairs. During simulation, the conditions are checked, and if any is found to be true, its associated value is assigned as the value of the mode variable.

The final major CC element, the component configuration, might be called a "composed component macro". Its purpose is to create similar composed component class definitions from a single component class that has some subcomponent information omitted. The component configuration does this by supplying the missing subcomponent information. For example, assume that there exists a composed component class named "Black Box" with three subcomponents for which implementations (and perhaps interfaces) have not been specified. Any number of similar "Black Box-like" component classes can be created by defining component configurations that specify different subcomponent class information for the three subcomponents.

CC does not appear to support QSIM model switching. Although multiple QDE's can be generated, the CC User's Guide does not document any way of generating the QDE transition relations needed to switch between the models during simulation.

CC-TO-CONFIG

Order of Mapping

The order in which CC elements are mapped to CONFIG elements is bottom-up. That is, a CC construct is converted only if all of its subparts have already been converted, or if it contains no subparts. In this way, a given construct can be transformed in a single pass, whereas with a top-down approach multiple passes might be required to complete the transformation of a given construct. This leads to the following transformation order:
1) Global quantity spaces with no parent specified;
2) Global quantity spaces whose parent has already been transformed;
3) Primitive component classes;
4) Component configurations are transformed to CC composed component classes; and
5) Composed component classes whose subcomponents have already been transformed.

Note that step 4 is a CC-to-CC transformation, not a CC-to-CONFIG transformation. The purpose of this step is to convert component configurations to a form that is easier to transform to CONFIG.

Mapping Quantity Spaces

Quantity spaces, regardless of whether they appear globally or within a component class definition, most closely resemble qualitative data types in the CONFIG process specification language. The main elements of a quantity space are an ordered set of landmark values and a set of conservation correspondences on those values. A CONFIG qualitative data type consists of an unordered set of qualitative interval names and a collection of operations on those qualitative names. Transformation of the landmark values to interval names is straightforward; since landmarks represent the boundaries of qualitative intervals, interval names will be generated by joining adjacent landmark names. For example, the set of landmark values

(L1 L2 L3 ...)

can be transformed to the set of interval names

(L1_L2 L2_L3 ....).

The transformation of conservation correspondences to operations on the newly generated qualitative type is complicated by the fact that the conservation correspondences are operations on landmarks rather than on intervals. However, this can be overcome by generating additional interval names that correspond to the intervals between 0 (which is a landmark in every quantity space) and the landmarks in the conservation correspondences. Then the sum of these intervals can be viewed as zero, and these interval operations can be transformed to CONFIG qualitative data type operations. For example, the quantity space,

(minf a b 0 c d inf) (0 0) (minf inf) (b c d))

would be transformed to the qualitative type

(minf_a a_b b_0 0_c c_d d_inf 0_0 minf_0 0_inf b_0 0_c 0_d)

and the transformed conservation correspondences (i.e., those intervals whose sum is the interval 0_0) would be (in functional form):

```
(SUMS-TO-ZERO 0_0 0_0)
(SUMS-TO-ZERO minf_0 0_inf)
(SUMS-TO-ZERO b_0 0_c 0_d)).
```

In addition, the original list of landmark values will be copied to CONFIG to document the original landmark ordering.

Quantity Spaces with Inheritance

Communication with Dr. Kuipers about the inheritance mechanism for quantity spaces indicates that there are still many unanswered questions about how the mechanism will operate. Thus, it was decided to take the view that the landmarks and conservation correspondences of parent quantity spaces are recursively inherited by child quantity spaces. Thus, the transformation of a quantity space whose parent quantity space has already been transformed is achieved by taking the union of the parent's interval set and the child's interval set, and the union of the parent's operation set and the child's operation set. This view assumes, however, that particular landmarks have the same qualitative meaning at each level of inheritance.

Mapping Primitive Components

Primitive components are component classes with no subcomponents. They will be transformed to a CONFIG device class. This transformation involves many elements, which are described below.

The implementation and interface of the component class contain descriptive text strings. These will be included as documentation in the CONFIG device class. Other items transformed as documentation will be the component class knowledge domain name and component parameters.

CC terminals and CONFIG port variable clusters serve the same purpose, and each is comprised of a collection of variables. Thus each terminal will be transformed to a CONFIG port variable cluster class, and the CONFIG device class will contain an instance of each variable cluster class as a port.

In a similar fashion, the component (local) variables will collectively be transformed to a CONFIG variable cluster class, and the CONFIG device class will contain an instance of this variable cluster class as an internal variable cluster. CC terminal and component variables may also have several options specified, such as "display" and "no-new-landmarks"; these are operational flags that have no counterpart in CONFIG, and so will be copied as documentation for the variable.

A CC mode variable is a variable and a collection of condition/value pairs. If one of the conditions is true, the variable takes on the corresponding value. Mode variables have no relationship to modes in a CONFIG device. Their variables will collectively be transformed to a CONFIG variable cluster class, and the CONFIG device class will contain an instance of the variable cluster class as an internal variable cluster. Their condition/value pairs will be transformed to MI processes.

CONFIG does not use constraint propagation. Thus, transformation of CC constraints to CONFIG may have limited utility. The relations specified in most CC constraints have predefined meanings; for example, M+ is a relation that between two variables, one of which affects the value of the other in a monotonically positive manner. The predefined meanings will not be translated, but the list of corresponding values in the constraint could be transformed to CONFIG process language operations on the types of the constraint arguments.

CC constraints define the value of one variable as a function of another; i.e., whenever the domain variable value changes, the range variable value changes according to a function of the domain variable. This relationship could be transformed to CONFIG MI or MD processes in which the change of domain variable from one qualitative value to another triggers an action that changes the range variable value accordingly. However, since only the name of the function rather than the function itself is encoded in the constraint, the transformation from constraint to MI or MD processes would have to be performed by the human modeler. If the modeler chooses not to perform this task, then the constraints would be transformed to CONFIG device class documentation.

Mapping Composed Components

The conversions described for primitive component classes also hold for composed component classes. There are two additional elements that must be converted for composed components: subcomponents and connections between subcomponents. By taking a bottom-up approach, the subcomponents already will have been converted, so instances of the subcomponent device classes can be included in the composed device class DCD. Subcomponent connections will be transformed to CONFIG relations, since both describe device-to-device connections, and will also be included in the DCD.

CONCLUSIONS AND FUTURE WORK

A CC-to-CONFIG knowledge representation mapping has been described. Although it is complete (i.e., it describes the transformation of all CC knowledge to the CONFIG representation), several elements are translated as documentation, due to differences in CC/QSIM and CONFIG. These gaps represent areas of further study for potential extension of CONFIG's capabilities. The mapping also highlights the need for further studies concerning mappings from other qualitative reasoning programs to CONFIG, and also the mapping of CONFIG representations to other programs. For example, even though a mapping from QSIM to CONFIG is not desirable, a mapping from CONFIG to QSIM might be. In summary, the current study has revealed more questions to be investigated.

A good starting point for further study would be the implementation of the mapping described here. Doing so will open the doors for empirical comparisons of CONFIG to QSIM. It is important to know if the results achieved by CONFIG are complimentary, contradictory, or essentially the same as the results of other programs. If complimentary, the programs could be used together to get better results. If contradictory, why? If essentially the same, comparisons could be made to find out if one program has a better computational efficiency than the others.

REFERENCES

1. Abelson, H.; Sussman, G.J.: The Structure and Interpretation of Computer Programs, MIT Press, 1985.

2. DeCoste, D.: Dynamic Across-Time Measurement Interpretation. *Artificial Intelligence*, vol. 51, 1991, pp. 273-341.

3. de Kleer, J.; and Brown, J.S.: A Qualitative Physics Based on Confluences. Qualitative Reasoning About Physical Systems, Daniel G. Bobrow, ed., MIT Press, 1985, pp. 7-83.

4. Falkenhainer, B.; and Forbus, K.D.: Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, vol. 51, pp. 95-143.

5. Farquhar, A.; Kuipers, B.; Rickel, J.; Throop, D.; and The Qualitative Reasoning Group: QSIM: The Program and its Use. Draft. Department of Computer Sciences, University of Texas at Austin, Austin, Texas, 1992.

6. Forbus, K.D.: Qualitative Physics: Past, Present, and Future. Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence, H.E. Shrobe, ed., Morgan Kaufmann, 1988, pp. 239-296.

7. Forbus, K.; and Falkenhainer, B.: Self-Explanatory Simulations: Scaling Up to Large Models. Proceedings of the Tenth National Conference on Artificial Intelligence, AAAI Press, 1992.

8. Forbus, K.D.; Nielsen, P; and Faltings, B.: Qualitative Spatial Reasoning: The CLOCK Project. *Artificial Intelligence*, vol. 51, 1991, pp. 417-471.

9. Franke, D.W.; and Dvorak, D.L.: CC: Component Connection Models for Qualitative Simulation, A User's Guide. Draft. Department of Computer Sciences, University of Texas at Austin, Austin, Texas, 1992.

10. IEEE Standard VHDL Language Reference Manual. IEEE Std. 1076-1987.

11. Kuipers, B.: Component-Connection Models. Draft. University of Texas at Austin, 1992.

12. Kuipers, B.J.; Chiu, C.; Dalle Molle, D.T.; and Throop, D.R.: Higher-Order Derivative Constraints in Qualitative Simulation. *Artificial Intelligence*, vol. 51, 1991, pp. 343-379.

13. Malin, J.T.; and The CONFIG Design Team: CONFIG User's Guide. Draft. Intelligent Systems Branch, Automation and Robotics Division, Johnson Space Center, 1990.

14. Malin, J.T.; Basham, B.D.; and Harris, R.A.: Use of Qualitative Models in Discrete Event Simulation to Analyze Malfunctions in Processing Systems. Artificial Intelligence in Process Engineering, Academic Press, 1990, pp. 37-79.

15. Rosenberg, R.C.; Karnopp, D.C.: Introduction to Physical System Dynamics. McGraw-Hill, 1983.