

# Generating Explanations of Device Behavior Using Compositional Modeling and Causal Ordering

Patrice O. Gautier and Thomas R. Gruber

Knowledge Systems Laboratory  
Stanford University  
701 Welch Road, Building C  
Palo Alto, CA 94304  
gautier@ksl.stanford.edu

## Abstract

Generating explanations of device behavior is a long-standing goal of AI research in reasoning about physical systems. Much of the relevant work has concentrated on new methods for modeling and simulation, such as qualitative physics, or on sophisticated natural language generation, in which the device models are specially crafted for explanatory purposes. We show how two techniques from the modeling research—compositional modeling and causal ordering—can be effectively combined to generate natural language explanations of device behavior from engineering models. The explanations offer three advances over the data displays produced by conventional simulation software: (1) causal interpretations of the data, (2) summaries at appropriate levels of abstraction (physical mechanisms and component operating modes), and (3) query-driven, natural language summaries. Furthermore, combining the compositional modeling and causal ordering techniques allows models that are more scalable and less brittle than models designed solely for explanation. However, these techniques produce models with detail that can be distracting in explanations and would be removed in hand-crafted models (e.g., intermediate variables). We present domain-independent filtering and aggregation techniques that overcome these problems.

## 1. Introduction

This paper presents a method for generating explanations of device behavior characterized by systems of mathematical constraints over continuous-valued quantities. Such models are widely used in engineering for dynamical systems, such as electromechanical and thermodynamic control systems. Given such a model and initial conditions, conventional simulation software can predict and plot the values of these quantities over time. However, the data can be difficult to interpret because conventional simulators do not explain how the predicted behavior arises from the structure of the modeled system and physical laws.

What we call *explanations* are presentations of information about the modeled system that satisfy three requirements. First, an explanation offers a meaningful *interpretation* of the simulation data, explaining how and why and not just what happened. For engineering tasks

such as design and diagnosis, it is useful to provide causal and functional interpretations. In this paper we focus on causal interpretations. Second, an explanation should present information at appropriate levels of abstraction. What is appropriate depends on the system being modeled, the purpose of the model, and the modeling primitives. For our tasks, we need explanations at the level of physical mechanisms and component operating modes, rather than graphs of numeric variables. Third, an explanation is a presentation of information in a format that is comprehensible to the human user. In the context of natural language generation, relevant design issues include choosing an appropriate level of detail, summarizing data, and adapting to information needs of users.

We have developed a system that generates explanations with these properties. It is part of the Device Modeling Environment (DME) [12], which integrates model formulation support, qualitative and numerical simulation, and explanation generation. A separate report introduces the explanation architecture and describes the text generation and human interface techniques [11].

In this paper, we focus on how two techniques from qualitative reasoning research, compositional modeling [6] and causal ordering [13] are applied and combined to produce explanations that satisfy the three criteria outlined above without imposing ad hoc or unscalable modeling formalisms. In Section 3 we present a series of example explanations generated in an interactive session. In Section 3 we describe how compositional modeling is used, and in Section 4 we describe the use of causal ordering. In Section 5 we analyze why it works, explaining how combining the two techniques makes it possible to achieve the three design requirements, and how problems that arise from this design are addressed. The final section compares related work.

## 2. A Running Example

We will demonstrate the explanation technique using a model of the space shuttle's Reaction Control System (RCS). The RCS is a system of thrusters that are used to steer the vehicle. The system consists of tanks, regulators, valves, thrusters, pipes, and junctions. The RCS system model comprises 160 model fragments that generate 150 equations relating 180 parameters. Figure 1 shows the topological structure of the RCS system. A similar picture

Funding was provided by NASA Grant NCC2-537 and NASA Grant NAG 2-581 (under ARPA Order 6822).

is displayed on the DME user's screen, providing a user interface for monitoring and editing capabilities.

At any time, the user may ask for explanations by clicking the mouse on text or graphics on the screen. Figure 2 shows a sequence of explanations produced in response to user queries. The first presentation, labeled (a), is an explanation of the salient events of the current state. This "what just happened" explanation uses heuristics to filter irrelevant information. Of the 180 variables, only one was mentioned in the explanation of Figure 2a.

All text and graphics presented in DME explanations are mouse-sensitive. By clicking on an icon, word, phrase, or sentence, the user can ask follow-up questions about any part of an explanation. In Figure 2a, the user clicks on the sentence stating that the check valve is now in blocking mode. The system produces a menu of possible queries relevant to this fact, as shown in Figure 2b. The user asks for an explanation by selecting a query from the menu.

The system then generates the explanation shown in Figure 2c, which explains why the selected behavior was observed. In this case, the behavior is determined by a single variable, the pressure differential of the quad check valve. The user then asks for the influences on that variable by clicking on the sentence describing the value of the variable, and selecting the desired query from the menu shown in Figure 2d.

The resulting explanation, shown in presentation 2e, summarizes the causal influences on the pressure differential variable. The value of this variable is determined by two chains of influences: the upstream pressures from the helium tank through the pressure regulators and the downstream pressures coming from the oxygen tank (see Figure 1). Using heuristics described in Section 4, the system simplifies these chains into two influences. In the ex-

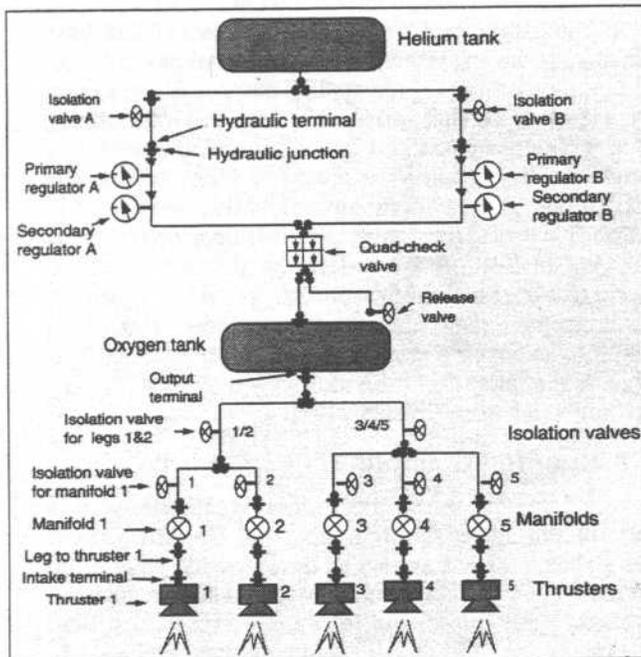


Figure 1: Schematic of the RCS system

planation, it says that the value of the variable in question, pressure differential, is determined by the input pressure at the oxygen tank and the nominal pressure of the primary pressure regulator. It explains that these were the salient influences because the secondary regulator was in pass-through mode and the primary regulator was in normal mode. It shows the equation that results from this simplification at the bottom of the display.

In Figure 2f, the user follows up by asking for the influences on the input pressure of the oxygen tank. This produces the explanation shown in Figure 2g, which extends the previous explanation. In this case, the system explains that the oxygen pressure is determined by the amount of gas and two exogenous constants "by the ideal gas law for the oxygen tank." Clicking on this phrase would result in an explanation that the oxygen tank is modeled as an ideal gas container, which is why the ideal gas law is governing in this situation.

These are a few of the explanation types that can be generated on demand (others are described in [11]). We now look at the roles of compositional modeling and causal ordering in generating these explanations.

### 3. Compositional Modeling and Explanation

Engineering models used to describe and predict the behavior of systems like the RCS are typically mathematical models. A set of continuous variables represents the values of physical quantities such as pressures and temperatures. The behavior of the system is defined by a set of equations that constrain these variables. Each model is based on a set of approximations, abstractions, and other assumptions about what is relevant to produce the desired data at a reasonable cost. Model formulation is the task of constructing a model from available primitives to answer some question. In electromechanical domains, these models are often constructed by hand from background knowledge of physics and engineering.

In the compositional modeling approach [6] to model formulation, engineering models are constructed from modular pieces, called *model fragments*. A model fragment is an abstraction of some physical domain, mechanism, structure, or other constituent of a model that contributes constraints to the overall behavior description. Model fragments can represent idealized components, such as resistors, transistors, logical gates, electrical junctions, valves, and pipes, and physical processes such as flows.

Each model fragment has a set of *activation conditions* that specify when an applicable model holds in a given simulation (e.g., the model of a boiling process can only hold when the temperature of the water is above some threshold). Each model fragment contributes a partial description of the behavior that occurs when the conditions hold. The behavior is specified in terms of algebraic and logical constraints on the values of simulation variables.

DME uses a compositional modeling approach for assembling a mathematical model from a library of model fragments. Model formulation and simulation are inter-

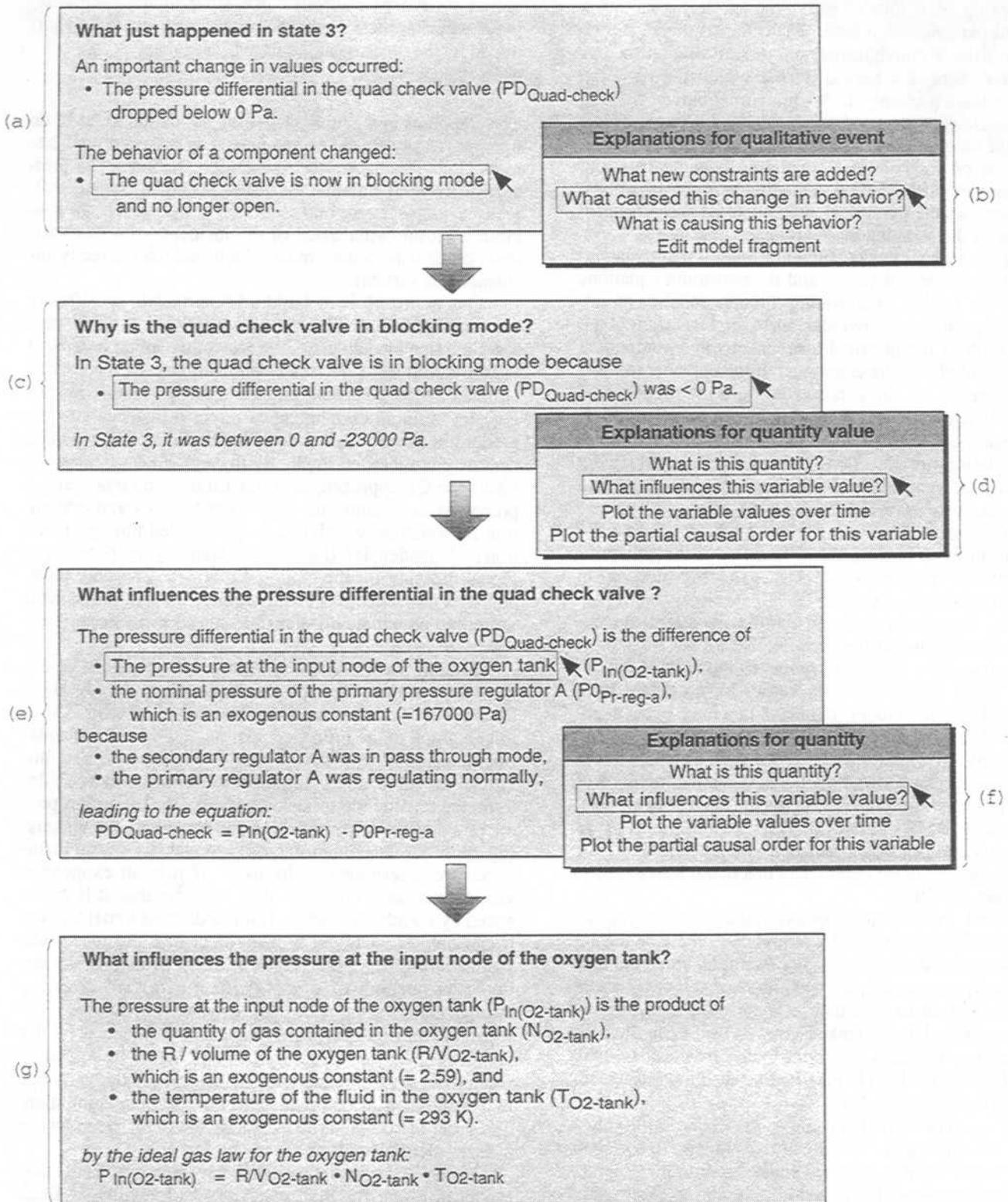


Figure 2: A sequence of explanations produced in response to user queries

leaved. During a simulation, DME monitors the activation conditions of model fragments; at each state, the system combines the equations of active model fragments into a set called the *equation model*. The equation model is then used to drive a conventional numerical simulation. A *qualitative state* is a period during which the equation model remains unchanged. Within a qualitative state, the numeric values of modeled parameters can change. When parameter values cross certain limit points, the activation conditions of some model fragments become true and others become false, leading to qualitative state transitions. DME monitors the numerical simulation for such changes, and updates the equation model for each new state.

The data predicted by simulation are a mathematical consequence of initial values and the constraint equations given in the model. Interpreting the data requires an understanding of the knowledge used in formulating the model, such as the physical mechanisms and component structure underlying the equations. If the engineer looking at the output is not the person who built the model, or if the model is complex and contains hidden assumptions, then it can be difficult for the engineer to make sense of the simulation output. DME's explanation services are intended to address this problem by relating predicted data to the underlying modeling choices.

Compositional modeling plays an essential role for explanation by providing the *derivation* of the equations from model fragments. DME uses the derivation information in several ways.

First, transitions between qualitative states are explained as a change in the set of active model fragments. For example, the summary of salient events (Figure 2a) describes those variables whose values have crossed limit points and lists the model fragments that have become active (quad check valve closed) and inactive (quad check valve open). To explain what caused such a change in behavior, the system shows how the activation conditions of the model fragment were satisfied (Figure 2c). Furthermore, DME uses the analysis of limit points in the activation conditions as a *salience heuristic*, focusing the summary on just those parameters that could lead to qualitative state transitions.

Second, the principles or assumptions underlying an equation can be explained by reference to the model fragments that contributed them. For example, when the user asked the system to describe the influences on the pressure at the input of the oxygen tank, it showed the ideal gas law equation applied to the tank (Figure 2g). It knew that the ideal gas law equation was contributed by the ideal-tank model fragment, which is inherited by the model fragment representing the oxygen tank.

Derivation knowledge is also used when simplifying a causal influence path. As shown in Figure 2e, the system relates the pressure at the quad check valve to the pressure at the primary regulator by explaining that the secondary regulator (which is between the other two components) is in pass-through mode. The model fragment for the pass-through mode specifies that the input and output pressures

are equal, and the system removes this equation in the explanation (see Section 4). Knowing the source of this equation—the pass-through operating mode—helps explain why the pressure at the quad check valve is determined by the pressure at the primary regulator.

#### 4. Causal Ordering

The equations in the model used for simulation in DME do not specify any causal orientation; they are purely mathematical constraints on the values of variables. The pressure differential variable, for instance, is related mathematically to almost every other variable in the model. To generate a causal explanation of the influences on this variable, one needs to determine which variables directly influence this variable.

One approach is to build ad hoc models specifically for explanation, in which the causal influences on all variables are fixed in advance. We reject this option because it is brittle and does not scale with the size of the model. Another approach would be to use compositional modeling, but build an assignment of direct influences into the model fragments. This is done in QP theory [8], in which causality is specified explicitly through *direct influences*. Using the QP approach, the model fragments representing processes each contribute causal dependency and orientation information, which can be propagated through functional dependencies (*indirect influences*) to produce a global ordering for the composed model. The only problem with this scheme is that it requires the model fragment writers to anticipate all plausible causal influences in advance.

Instead, we assume that causal influences can be *inferred* at run time. We use an adaptation of the causal ordering procedure developed by Simon and Iwasaki [13] to infer a graph of the influence relation. Given a set of equations constraining a set of model parameters, and the knowledge of which parameters are exogenous (i.e., determined outside the modeled system), the algorithm produces a dependency graph over the remaining parameters. For each variable in an equation model, its causal influences are determined as follows. If it is an exogenous variable and/or a constant, then by definition it is influenced by no other variable. If it is a discrete variable, then it can only be changed by the effect of a discrete mode fragment (e.g., the triggering of a relay), by an operator action (the opening of a valve), or by forward chaining through rules from one of these events. If a variable is integrated from its derivative, then it is influenced by the derivative (e.g., acceleration causes change in velocity). Otherwise, the influences on the variable are the variables that were used to compute it in the numeric simulation. The order of computation is exactly the order given by the causal ordering graph.

Figure 3 shows the causal order graph of the equation model in effect for the explanations of Figure 2. Each node in the graph corresponds to a variable in the model. Each arc represents an influence given by an equation; equations relating more than two variables appear as

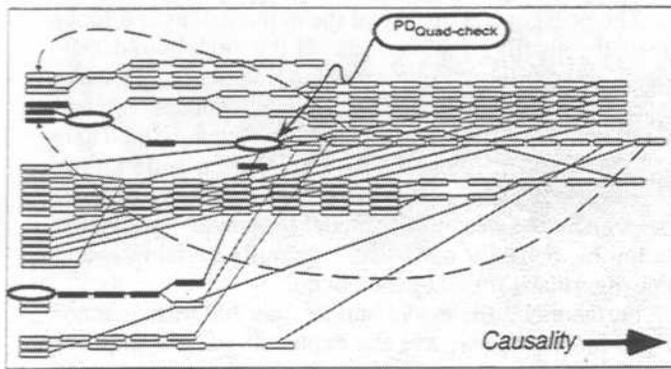


Figure 3: The causal order graph for the RCS in state 3. Nodes are variables. Solid arcs are direct influences, and dashed arcs denote integration relationships.

branching arcs. Influences go roughly from left to right; nodes on the left boundary are either exogenous, constant, or integrated variables. The integrated variables are computed from their derivatives on each numeric integration loop, as depicted by dashed arcs in the figure.

The direct influences on a variable such as  $PD_{Quad-check}$  are those connected by single arcs leading into the variable. The subgraph of all influences on a variable is formed by recursively following influence arcs to the left.

To generate the causal explanation of Figure 2e, the system computed the subgraph of influences for the variable to explain,  $PD_{Quad-check}$ . The subgraph of influences on this variable and the associated equations are shown in Figure 4.

A straightforward application of causal ordering to explanation would be to output the entire subgraph of influences. However, when this subgraph is large the resulting output would not be an effective presentation of information in textual form (violating our third requirement). A second design alternative would be to only output the immediately adjacent nodes in the influence graph. For example, the explanation of Figure 2e would read “ $PD_{Quad-check}$  is the difference of the pressure at the output node of the quad check valve ( $PD_{In(Quad-check)}$ ) and the pressure at the input of the quad check valve ( $PD_{In(Quad-check)}$ ) by the definition of pressure differential.” Then the user could traverse the next node in influence graph by clicking on one of these two variables and asking for causal influences on it. This approach can distribute a single explanation over several presentations, and requires the user to sort out irrelevant detail.

To overcome this problem, DME applies salience heuristics to select a subset of influences to report in an explanation. The system works back through the subgraph from the influenced variable, collapsing paths called influence chains to a single-step influence. For example, in the influence graph of Figure 4, the chain from  $P_{Out(O2-tank)}$  to  $PD_{Quad-check}$  and the chain from  $P_{OPr-reg-A}$  to  $PD_{Quad-check}$  were collapsed. Instead of describing the graph of 12 potential influences, the explanation says that  $PD_{Quad-check}$  is simply the difference between  $P_{Out(O2-tank)}$  and  $P_{OPr-reg-A}$ .

DME currently uses two salience heuristics to select causal influences, both of which collapse a chain of influences into a virtual, single-step influence.

- **Collapsing equality chains:** Chains in the influence subgraph of the form  $v_1=v_2, v_2=v_3 \dots =v_n$  are transformed into the equation  $v_1 = v_n$ , and only these two influences are presented to the user. The path between nodes  $P_{OPr-reg-A}$  and  $PD_{Quad-check}$  in Figure 4 was collapsed using this heuristic.
- **Collapsing paths of same dimension:** Chains of variables of the same physical dimension are collapsed into single-step influences. The value of a variable is a physical quantity of some dimension, such as pressure, temperature, or flow. If a sequence of variables all have the same dimension, then they are presumed to be driven as a whole by some influence. For example, the path of fluid flow in the RCS corresponds to sequences of influences of the same dimension (flow).

The influences to report to the user are those connected to the influenced variable by a collapsed sequence or an adjacent node in the influence graph. The resulting influences need not be terminals in the original influence

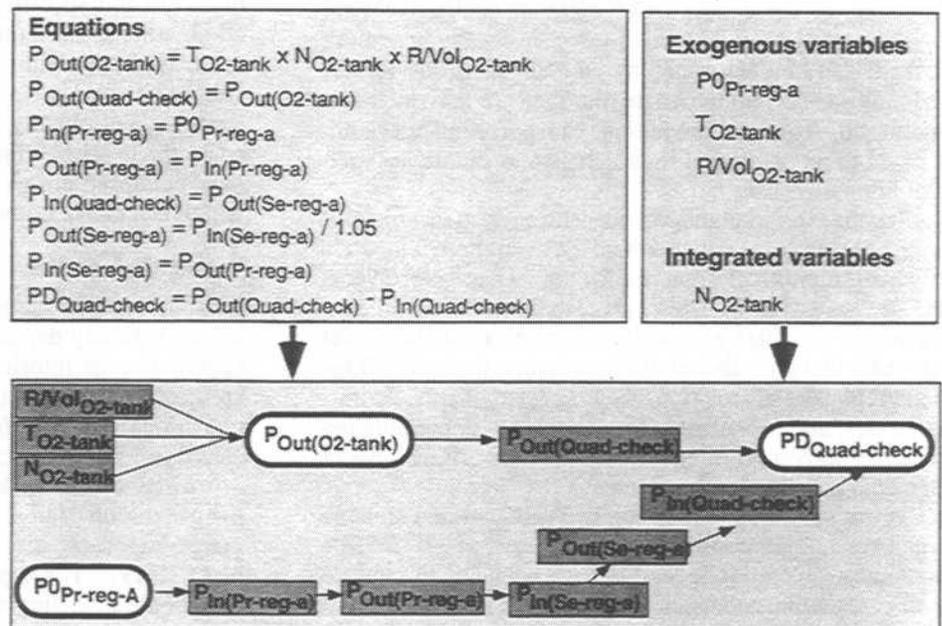


Figure 4: The subgraph of influences on  $PD_{Quad-check}$ , which are shown in black in Figure 3. The circled variables appear in the explanation of Figure 2e.

graph (i.e., variables that are not influenced). For example, the variable  $P_{\text{Out(O2-tank)}}$  is reported as an influence on  $PD_{\text{Quad-check}}$ , but the former is in turn influenced by three other parameters of the oxygen tank. The user can ask for these influences by invoking a follow-up question, as shown in parts f and g of Figure 2.

In explanations where sequences are collapsed, the system displays the equation that results from symbolically solving the set of relevant equations. For example, the equations shown in Figure 4 are reduced to the equation  $PD_{\text{Quad-check}} = P_{\text{Out(O2-tank)}} - P_{\text{O(Pr-reg-a)}}$ .

Presenting a chain of influences as a single-step influence imposes a view of the data that suggests a causal interpretation. The system gives the "reasons" for this causal relationship by listing the model fragments that contributed a collapsed equation and that have activation conditions dependent on time-varying variables. Typically these are operating modes of components. For example, the system justified the single-step jump to the nominal pressure of the primary regulator "because the secondary regulator A was in pass through mode and the primary regulator A was regulating normally" (from Figure 2e).

## 5. Summary and Analysis

The use of compositional modeling and causal ordering techniques is responsible for several desired properties of the explanation approach we have presented.

First, it is possible to generate causal interpretations (our first design requirement) from models that are designed for engineering analysis and simulation, rather than being crafted specially for explanation. Because causal influences can be inferred from equation models using causal ordering, they need not be built in to the model. Because explanation is integrated with compositional modeling, explanations of the causes of changes in qualitative state (e.g., discrete events like the quad check valve becoming blocked) can be determined by an analysis of the logical preconditions of model fragments that are activated and deactivated. The set of conditions to report need not be anticipated in advance, and it can change as the model fragment library evolves.

Second, the explanations can be presented at useful levels of abstraction (requirement 2), even though they are driven by numerical simulation data. Low-level events such as changes in variable values and mathematical constraints on variables are explained in terms of the model fragments that contributed the constraints; the model fragments represent abstractions such as components, operating modes, and processes. This capability is possible because the derivation of equations from the original model fragments is known to the system.

Third, the explanations can be presented in a suitable format for human consumption (requirement 3). DME's simple text generation procedures are adequate to produce small explanations; the capability to ask interactive follow-up questions gives the user control over the level of detail and lessens the need for advanced text planning and user modeling.

The pedagogical quality of the explanations is a function of the quality of the models. If the model builder divides a system model into modular fragments that make sense in the domain (e.g., components, connections, and operating modes), then the explanation system will be able to describe them. None of the explanation code knows anything about flows, pressures, or even junctions. It knows about the structure of model fragments—activation conditions, behavior constraints, quantity variables—and some algorithms for text generation.

Furthermore, the model builder may add textual annotations incrementally, and the explanations will improve gracefully. For example, if components are not named with text annotations, the system will generate names based on the Lisp atoms used to specify the model fragments. As the textual annotations are added, the system can compose them into more useful labels, such as the subscripted variable notation. This capability is possible because of the modularity and compositionality enabled by the compositional modeling.

The major problem with the use of causal ordering and compositional modeling is the introduction of irrelevant detail. A model built specially for explanation can include only those variables and causal relations that are relevant to the target explanations. A model composed from independent model fragments includes intermediate variables and equations such as those modeling the flows and pressures at junctions. Since the causal ordering algorithm is domain-independent and works bottom-up from equations, rather than top-down from model fragments, these intermediate variables are included in the determination of causal influences on a variable.

The solution taken in DME was the application of salience heuristics, as described in Section 4. Although these are not specific to any domain, they are aimed at eliminating the intermediate variables and equations that occur when modeling circuits using constraint equations. Additional heuristics may be needed in other classes of models. Fortunately, it is possible to degrade smoothly when the heuristics fail. If an influence chain should have been collapsed but was not, the user can easily traverse it with a few clicks of the mouse.

## 6. Related Work

Existing systems for generating explanations of device behavior typically depend on models built explicitly for the explanation or tutoring task [10,19]. When explanations are generated from more general behavior models, the explanations typically follow from hard-coded labeling of causal influence [21] or component function [14].

Much of the work in explanation has concentrated on the generation of high-quality presentations in natural language based on discourse planning and user modeling [7,17,18,19]. These presentation techniques are independent of the modeling method or technique for determining causal influence, and so could be adapted for the explanation approach presented in this paper.

The task of giving a causal interpretation to device behavior has been addressed from several perspectives [3,13,20]. Alternatives to the causal ordering algorithm have been developed, such as context sensitive causality [16] and a method based on bond graphs [20]. These methods differ in the information they require and the class of models they accept. Given a model like the RCS and the same exogenous labeling, these methods should all produce the same ordering graph. Any of these methods could be used by the explanation technique we have described.

The idea of using such a causal interpretation for generating explanations has been previously proposed. In QUALEX [4], a causal graph is computed from a set of confluences [2], and the graph is used to explain the propagation of perturbations ("if X goes up, Y goes down"). This system is limited by the modeling representation: the confluence equations can only predict the sign of the first derivative and do not scale.

Qualitative models have been used to generate explanations in tutoring and training systems [10,21]. DME's explanation system can also generate explanations on such models (using QSIM [15] for simulation). Qualitative models have known limitations of scale.

Work on the SIMGEN systems [5,9] was the first to achieve the effect of qualitative explanation using numerical simulation models. SIMGEN also uses a compositional modeling approach, and explains simulation data using the derivation of equations from model fragments. The SIMGEN strategy is to build parallel qualitative and quantitative model libraries, analyze the possible qualitative state transitions for a given scenario description, and compile out an efficient numeric simulator. While DME determines model fragment activation and assembles equation models at run time, SIMGEN precomputes and stores the information relating the quantitative model and the qualitative model (which we call the derivation information). For causal explanations, the SIMGEN systems use the causal assignment that is built into the direct and indirect influences of the qualitative model. If the directly influenced variables are exogenous, this produces the same causal ordering as the Iwasaki and Simon algorithm. To answer questions such as "what affects this variable?", SIMGEN currently shows the single-step influences and does not summarize chains of influences.

In principle, the DME explanation method could use the model derivation information from a SIMGEN model library, and SIMGEN could use DME's text composition, causal ordering, user interface, and filtering techniques. Furthermore, QSIM-style qualitative models can be derived from quantitative models as used in DME. We are working with the authors of SIMGEN and QPC [1] (which is similar to DME and uses QSIM) on a common modeling formalism that might make it possible to exchange model libraries and test these conjectures.

## Acknowledgments

The DME system is the product of members of the How Things Work project, including Richard Fikes, Yumi Iwasaki, Alon Levy, Chee Meng Low, Fritz Mueller, James Rice, and Pandu Nayak. Brian Falkenhainer and Ken Forbus have been very influential.

## Bibliography

- [1] J. Crawford, A. Farquhar, & B. Kuipers. QPC: A Compiler from Physical Models into Qualitative Differential Equations. *AAAI-91*, pp. 365-371, 1990.
- [2] J. de Kleer & J. S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7-83, 1984.
- [3] J. de Kleer & J. S. Brown. Theories of Causal Ordering. *Artificial Intelligence*, 29(1):33-62, 1986.
- [4] S. A. Douglas & Z.-Y. Liu. Generating causal explanation from a cardio-vascular simulation. *IJCAI-89*, pp. 489-494, 1989.
- [5] B. Falkenhainer & K. Forbus. Self-explanatory simulations: Scaling up to large models. *AAAI-92*, pp. 685-690, 1992.
- [6] B. Falkenhainer & K. D. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95-143, 1991.
- [7] S. K. Feiner & K. R. McKeown. Coordinating text and graphics in explanation generation. *AAAI-90*, pp. 442-449, 1990.
- [8] K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85-168, 1984.
- [9] K. D. Forbus & B. Falkenhainer. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. *AAAI-90*, pp. 380-387, 1990.
- [10] K. D. Forbus & A. Stevens. Using qualitative simulation to generate explanations. *Proceedings of the Third Annual Conference of the Cognitive Science Society*, 1981.
- [11] T. R. Gruber & P. O. Gautier. Machine-generated explanations of engineering models: A compositional modeling approach. *IJCAI-93*, 1993.
- [12] Y. Iwasaki & C. M. Low. Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes. *Intelligent Systems Engineering*, 1(2)1993.
- [13] Y. Iwasaki & H. Simon. Causality in device behavior. *Artificial Intelligence*, 29:3-32, 1986.
- [14] A. M. Keuneke & M. C. Tanner. Explanations in knowledge systems: The roles of the task structure and domain functional models. *IEEE Expert*, 6(3):50-56, 1991.
- [15] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289-388, 1986.
- [16] M. Lee, P. Compton, & B. Jansen. Modelling with Context-Dependent Causality. In R. Mizoguchi, Ed., *Proceedings of the Second Japan Knowledge Acquisition for Knowledge-Based Systems Workshop*, Kobe, Japan, pp. 357-370, 1992.
- [17] J. D. Moore & W. R. Swartout. Pointing: A way toward explanation dialog. *AAAI-90*, pp. 457-464, 1990.
- [18] C. Paris. *The Use of Explicit User Models in Text Generation: Tailoring to a User's Level of Expertise*. PhD Thesis, Columbia University, 1987.
- [19] D. Suthers, B. Woolf, & M. Cornell. Steps from explanation planning to model construction dialogues. *AAAI-92*, pp. 24-30, 1992.
- [20] J. Top & H. Akkermans. Computational and Physical Causality. *IJCAI-91*, pp. 1171-1176, 1991.
- [21] B. White & J. Frederiksen. Causal model progressions as a foundation for Intelligent learning. *Artificial Intelligence*, 42(1):99-155, 1990.

## Appendix A : Example Model Fragments

```
(define-model fluid-terminal
:it-is-"a terminal with pressure and flow"
:type terminal
:activation-condition default
:qty-slots (
  (pressure-at-terminal
:it-is-"the pressure at the terminal"
:the-"the-object ("pressure" "at")
:abbreviation "P"
:value-restriction (0 plus-infinity)
:dimension pressure)
(flow-thru-terminal
:abbreviation "F"
:the-"the-object ("flow" "through")
:it-is-"the flow through the terminal"
:dimension mass-flow)))
```

```
(define-model
  hydraulic-component-with-polarity
:type hydraulic-component
:it-is-"an hydraulic asymmetric (i.e. with a
  privileged direction of flow ) component
  with terminals"
:component-slots (
  (input-terminal
:range-class fluid-terminal
:abbreviation "In"
:the-"the-object ("input terminal" "of"))
  (output-terminal
:range-class fluid-terminal
:abbreviation "Out"
:the-"the-object ("output terminal" "of"))))
```

```
(define-model quad-check-valve
:it-is-"a valve that ensures that there is no backflow"
:type hydraulic-component-with-polarity
:activation-condition default
:qty-slots (
  (valve-resistance
:abbreviation "R"
:the-"the-object ("fluid resistance" "of")
:it-is-"the fluid resistance of
  the quad-check-valve"
:value-restrictions (0 plus-infinity)
:dimension fluid-resistance)
  (pressure-differential
:abbreviation "PD"
:the-"the-object ("pressure differential" "in")
:it-is-"the pressure differential between
  output and input terminals"
:dimension pressure))
:operating-modes
  (quad-check-valve-open
  quad-check-valve-closed)
:behavior-constraints (
  (quad-check-pressure-differential
:expression
```

```
(= (pressure-differential $m)
  (- (pressure-at-terminal
      (input-terminal $m))
     (pressure-at-terminal
      (output-terminal $m))))
:it-is-"the definition of the pressure
  differential")
```

```
(define-model quad-check-valve-operating-mode
:type operating-mode-model
:participant-slots (
  (quad-check-valve-involved
:range-class quad-check-valve)))
```

```
(define-model quad-check-valve-open
:type (quad-check-valve-operating-mode
  continuous-behavior-model)
:component-is-"open"
:it-is-"an operating-mode of the quad-check valve,
  in which the gas flows in the correct direction, i.e
  from the input node to the output node"
:instantiation-condition default
:activation-condition (
  (q> (pressure-differential
      (quad-check-valve-involved $m))
      1e-5))
:behavior-constraints (
  (quad-check-input-node-open
:expression
  (= (flow-thru-terminal
      (input-terminal
      (quad-check-valve-involved $m)))
     (/ (- (pressure-at-terminal
          (input-terminal
          (quad-check-valve-involved $m)))
          (pressure-at-terminal
          (output-terminal
          (quad-check-valve-involved $m))))
        (valve-resistance
          (quad-check-valve-involved $m))))))
  :it-is-"Ohm's law applied to the valve modeled
  as a linear resistive device")
  (quad-check-flow-conservation
:it-is-"the conservation of flow in the valve"
:expression
  (= 0
    (+
     (flow-thru-terminal
      (input-terminal
      (quad-check-valve-involved $m)))
     (flow-thru-terminal
      (output-terminal
      (quad-check-valve-involved $m)
      ))))))))
```

```

(define-model quad-check-valve-closed
  :type(quad-check-valve-operating-mode
        continuous-behavior-model)
  :component-is-"in blocking mode"
  :it-is-"an operating mode, in which the quad
    -check valve is closed, preventing backflow"
  :instantiation-condition default
  :activation-condition (
    (q<= (pressure-differential
          (quad-check-valve-involved $m))
         0))
  :behavior-constraints (
    (input-node-closed
     :expression
      (= (flow-thru-terminal
          (input-terminal
            (quad-check-valve-involved $m)))
         0)
      :it-is "the conservation of flow when
        the input node is closed")
    (output-node-closed
     :expression
      (= (flow-thru-terminal
          (output-terminal
            (quad-check-valve-involved $m)))
         0)
      :it-is "the conservation of flow when the output
        node is closed")))

```

```

(define-model generic-tank
  :type (hydraulic-component-leak-capable
        hydraulic-component-with-polarity)
  :activation-condition default
  :it-is-"a tank containing fluid"
  :qty-slots (
    (quantity-contained
     :abbreviation "N"
     :it-is-"the quantity of fluid contained in the tank"
     :the-object ("quantity of gas" "in")
     :value-restrictions (0 plus-infinity)
     :dimension mass)
    (tank-temperature
     :abbreviation "T"
     :it-is-"the temperature of the fluid in the tank"
     :the-object ("temperature" "in")
     :value-restrictions (0 plus-infinity)
     :dimension temperature)
    (ideal-gas-law-constant
     :abbreviation "R/V"
     :it-is-"the ideal gas law constant for the tank
      (R/V in PV = nRT)"
     :the-object ("R divided by the volume" "of")
     :Value-restrictions (0 plus-infinity)
     :dimension gas-law-constant)
    )
  :behavior-constraints(
    (tank-flow-equation
     :it-is-"the flow conservation for the tank"
     :expression (= (d/dt quantity-contained)
                    (+
                     (flow-thru-terminal (input-terminal $m))
                     (flow-thru-terminal (output-terminal $m))))))

```

```

(tank-pressure-conservation
 :expression
  (= (pressure-at-terminal (output-terminal $m))
     (pressure-at-terminal (input-terminal $m))))
:it-is-"the pressure propagation in the tank")
(tank-ideal-gas
 :expression
  (= (pressure-at-terminal (output-terminal $m))
     (* quantity-contained
        ideal-gas-law-constant
        tank-temperature)))
:it-is-"the ideal gas law for the tank")
)
)

```