Automated Model Selection using Context-Dependent Behaviors

P. Pandurang Nayak*
Knowledge Systems Lab.,
701 Welch Road, Bldg. C,
Palo Alto, CA 94304.

Leo Joskowicz
IBM, Watson Res. Ctr.,
P.O. Box 704,
Yorktown Heights, NY 10598.

Sanjaya Addanki
IBM, Watson Res. Ctr.,
P.O. Box 704,
Yorktown Heights, NY 10598.

Abstract

Effective reasoning about complex engineered devices requires device models that are both adequate for the task and computationally efficient. This paper presents a method for constructing simple and adequate device models by selecting appropriate models for each of the device's components. Appropriate component models are determined by the context in which the device operates. We introduce *context-dependent behaviors* (CDBs), a component behavior model representation for encapsulating contextual modeling constraints. We show how CDBs are used in the model selection process by exploiting constraints from three sources: the structural and behavioral contexts of the components, and the expected behavior of the device. We describe an implemented program for selecting a simplest adequate model. The inputs are the structure of the device, the expected device behavior, and a library of CDBs. The output is a set of component CDBs forming a structurally and behaviorally consistent device model that achieves the expected behavior.

Introduction

Effective reasoning about complex engineered devices requires device models that are adequate for the task and computationally efficient. Producing such models requires identifying relevant device features and applicable simplifications. In most existing applications, the user constructs the device model appropriate for the task. This is a difficult, error-prone, and time-consuming activity requiring skilled and experienced engineers. Automating the model construction process overcomes these drawbacks and provides future intelligent programs with a useful modeling tool.

*Pandurang Nayak was supported by an IBM Graduate Technical Fellowship. Additional support for this research was provided by the Defense Advanced Research Projects Agency under NASA Grant NAG 2-581 (under ARPA order number 6822), by NASA under NASA Grant NCC 2-537, and by IBM under agreement number 14780042.

Model-based reasoning systems embody an important advance in automating the construction of device models—the device model is automatically constructed from a description of the structure of the device. However, currently these systems have a single model for each component, thus limiting both the modeling scope and the problems that can be solved. Allowing multiple component models adds flexibility and extends the scope. Producing a simplest, adequate device model then consists of selecting component models that are mutually compatible, globally consistent, and incorporate appropriate simplifying assumptions.

In this paper we present a method for constructing simple and adequate device models by selecting appropriate models for each of the device's components. The key insight is that adequate component models are determined by the context in which they operate. We have identified three types of contexts: the *structural* and *behavioral* contexts of the components, and the *expected behavior* of the device. The structural context of a component consists of its physical properties and the components to which it is structurally related. The behavioral context of a component consists of its behavior and the behavior of related components. Expected behaviors are abstract descriptions of device behavior.

We introduce *context-dependent behaviors* (CDBs), a behavior model representation for encapsulating contextual modeling constraints. We describe an implemented program that uses the structural and behavioral contexts of components, and the expected behavior of the device, to select adequate component CDBs. The inputs are the structure of the device, the expected device behavior, and a library of CDBs. The output is a set of component CDBs forming a structurally and behaviorally consistent device model that minimally achieves the expected behavior.

Example: a temperature gauge

This section presents an example of a device with multiple models for its components, and defines the properties of a good model. Figure 1 shows the schematic of a temperature gauge from [Macaulay, 1988], consisting of a battery, a wire, a bimetallic strip, a pointer,

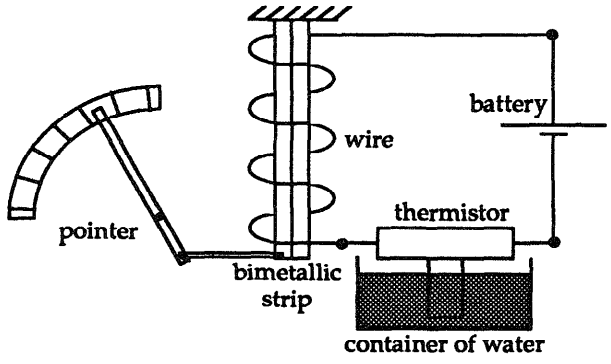

Figure 1: A temperature gauge

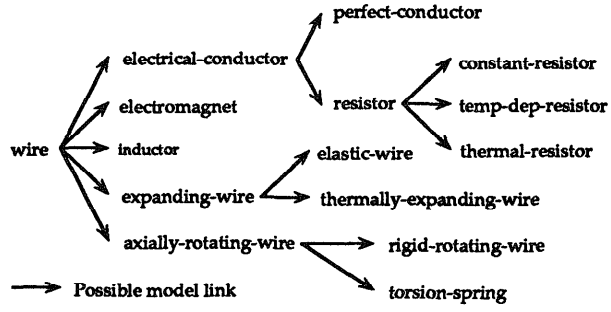

Figure 2: The possible models of a wire.

and a thermistor. A thermistor is a semiconductor device; a small increase in its temperature causes a large decrease in its resistance. A bimetallic strip has two strips made of different metals welded together. Temperature changes cause the two strips to expand by different amounts, causing the bimetallic strip to bend. The temperature gauge works as follows: the thermistor senses the water temperature. The thermistor temperature determines the thermistor resistance, which determines the circuit current. This determines the amount of heat dissipated in the wire, which determines the bimetallic strip's temperature. This determines the bimetallic strip's deflection, which determines the pointer's angular position.

To model the temperature gauge, we use a component model library that contains multiple models for each component. Figure 2 shows some of the wire's possible models. For example, the wire can be modeled as an `electrical-conductor`, which can be a `perfect-conductor` or a `resistor`. The `resistor` can be modeled as a `constant-resistor`, a `temp-dep-resistor`, or a `thermal-resistor` (which models the heat generated in the resistor). All components can be modeled as `physical-things` with various thermal, mass, and motion models.

The simplest model that explains the workings of the temperature gauge models the wire both as a `thermal-resistor` and a `constant-resistor`, the bimetallic strip as a `thermal-bms` (which models the bending of the bimetallic strip due to temperature changes), the pointer as a `rotating-object`, the battery as a `voltage-source`, and the thermistor as a `thermal-thermistor` (which models the temperature dependence of the thermistor's resistance).

This model satisfies the two important properties of a good model: adequacy and simplicity. Adequacy guarantees that the model correctly captures the device's behavior. For example, ignoring the thermal properties of the wire (i.e., modeling it only as a `constant-resistor`, which does not model the heat generated due to current flow) fails to account for the bimetallic strip bending and consequently the pointer's displacement. If the pointer does not move, the device does not measure temperature changes. Simplicity guarantees that the model captures only the physical phenomena necessary for explaining the device's behavior. For example, modeling the wire's magnetic and kinematic properties in addition to its thermal and electrical properties produces a consistent but needlessly complicated model with respect to its main function of measuring temperature changes. Selecting an appropriate subset of component models, from a space of possible models, guarantees both properties.

Context-dependent behaviors

Component models in our system are encapsulations of component behavior and contextual modeling constraints. We call these models *context-dependent behaviors* (CDBs). Behavioral information is represented with qualitative or quantitative time-varying equations. Contextual modeling constraints are represented with structural and behavioral constraints.

CDBs are represented as classes that inherit properties to their instances. A component is modeled as a CDB by making it an instance of the corresponding class. CDBs are organized in three ways: (a) in a generalization hierarchy, representing the "subset-of" relation between CDBs; (b) in a "possible-models" hierarchy (e.g., Figure 2); and (c) into *assumption classes* [Addanki *et al.*, 1991; Falkenhainer and Forbus, 1991]. The possible models of a CDB are the set of additional CDBs that can be used to model instances of that CDB. The "subset-of" and "possible-models" relations between CDBs may overlap, but are not identical. For example, `thermal-thermistor` is a specialization, but not a possible model, of `thermal-object`: not all objects can be modeled as `thermal-thermistors`, only thermistors can.

An assumption class is a set of CDBs that can be viewed as different, mutually contradictory descriptions of the same phenomenon. CDBs within an assumption class can be related to each other by a primitive *approximation* relation which captures the relative accuracy with which the CDBs describe the phenomenon. We assume that each assumption class con-

```
(defcdb resistor (electrical-conductor)
  ((parameters
     (resistance
       :range resistance-parameter
       :doc "The resistor's resistance"))
   (equations
     (= (voltage-difference ?object)
        (* (resistance ?object)
           (current (elec-term-1 ?object))))
     (> (resistance ?object) 0))
   (possible-models constant-resistor
                    temp-dep-resistor
                    thermal-resistor)
   (possible-model-of electrical-conductor)
   (assumption-class elec-conductor-class)
   (approximations perfect-conductor)
   (req-assumption-classes resistance-class)
   (structural-constraints)
   (behavioral-constraints
     (implies
       (>= (* (voltage-difference ?object)
              (current (elec-term-1 ?object)))
           (elec-power-threshold ?object))
       (model-as ?object thermal-resistor)))))
```

Figure 3: The `resistor` CDB.

tains a single most accurate CDB.

Figure 3 shows the definition of the `resistor` CDB.[1] It is a specialization of the `electrical-conductor` CDB, and it defines the `resistance` parameter for its instances. The `equations` clause describes behavior with equations relating parameters defined for instances of `resistor`. The `possible-models` and `possible-model-of` clauses describe the `resistor`'s position in the "possible models" hierarchy. The `assumption-class` clause identifies the `resistor`'s assumption class, and the `approximations` clause identifies CDBs in that assumption class that are approximations of `resistor`. The `resistor` CDB provides a partial description of electrical resistance; the `req-assumption-classes` clause identifies assumption classes that must be included to complete the description.

The `structural-constraints` and `behavioral-constraints` clauses define the CDBs's contextual modeling constraints and are stated in a first-order constraint language. There are two types of constraints: `model-as` constraints and *preconditions*. `Model-as` constraints are implication constraints with a `model-as` literal (or a conjunction of `model-as` literals) in the consequent.[2] They are rules that specify conditions under which particular components must be modeled by particular CDBs. Preconditions are constraints that are not `model-as` constraints. They are necessary (but not sufficient) conditions for a component to be modeled by the CDB.

CDBs describing different aspects of a component's behavior can be combined to produce a component model. We use a set of rules to obtain the equations of such a component model from the equations of the individual CDBs.[3] For example, consider a hot wire under tension. To model the dependence of the wire's length on both tension and temperature, we would model the wire both as an `elastic-wire` and a `thermally-expanding-wire`.

The relationship between components and CDBs is a many-to-many mapping: a single component can be modeled by different CDBs, and a single CDB can model different components. For example, a wire can be modeled as an `electrical-conductor` or an `electromagnet`. The `electrical-conductor` CDB can be used to model a wire, or a metallic pipe, This many-to-many relation between components and CDBs gives modeling flexibility for different reasoning tasks. For example, device analysis consists of finding the appropriate CDBs for a given set of components. Device design consists of finding components for a given set of desired behaviors described as CDBs.

Structural context

The structural context of a component consists of its physical properties (e.g., its shape, mass, and material composition), the structural relations that it participates in, and the components to which it is structurally related. Structural relations describe the structure of a device and include relations such as `connected-to` (indicating that two component terminals are connected), `coiled-around` (indicating that a wire is coiled around a component), and `meshed` (indicating that a pair of gears mesh). The structural context constrains the space of possible device models: the physical properties of components constrain the space of possible component models, while the structural relations constrain the space of possible component interactions.

The structural preconditions of a CDB are constraints on the structural context of a component that must be satisfied if the component is to be modeled by that CDB. For example, the precondition:

```
(and (composition ?object ?material)
     (metal ?material))
```

in the `electrical-conductor` CDB indicates that a component must be metallic for it to be modeled as an `electrical-conductor`. Structural preconditions are similar to process preconditions in QP theory [Forbus, 1984]. However, unlike process preconditions, they are only necessary conditions. Hence, the above constraint does not require that every metallic object be modeled as an `electrical-conductor`.

`Model-as` structural constraints are heuristic constraints on the structural context of a component that

[1] Symbols starting with "?" are variables. "?object" is bound to the CDB instance under consideration.

[2] `(model-as ?x ?y)` says that `?x` should be modeled as an instance of `?y`.

[3] Similar to combining influences in [Forbus, 1984].

enforce the selection of compatible CDBs for structurally related components. Compatible CDBs allow structurally related components to interact with each other. For example, the **model-as** constraint:

```
(implies
  (and (terminals ?object ?term1)
       (voltage-terminal ?term1)
       (connected-to ?term1 ?term2))
  (model-as ?term2 voltage-terminal))
```

in the **electrical-component** CDB says that if a component is modeled as an **electrical-component**, then terminals connected to that component's voltage terminals must be modeled as voltage terminals. This allows the components corresponding to the connected terminals to interact by sharing voltages at those terminals.

Behavioral context

The behavioral context of a component consists of its behavior and the behavior of related components. The behavior of a component is the values, and variations over time of the values, of parameters used to model the component. A component's behavioral context can provide modeling information not explicitly available in the structural context. Behavior generation explicates information that is implicit in equations. Consider modeling an air gap: if the the voltage drop across it is large enough (as in a properly functioning spark plug), then it should be modeled as an electrical conductor; if the voltage drop across it is not large enough (as in a common electrical switch), it should be modeled as an electrical insulator. The value of the voltage drop across the air gap (a behavioral property) determines the appropriate model for it.

Behavioral preconditions in a CDB are constraints on the behavioral context of a component that must be satisfied if the component is to be modeled by that CDB. For example, the precondition:

```
(< (voltage-difference ?U)
   (voltage-diff-threshold ?U))
```

in the **perfect-conductor** CDB indicates that a component can be modeled as a **perfect-conductor** only if the voltage drop across it is less than some threshold. Behavioral preconditions are used to decide which CDBs in an assumption class can be used to model a component. This is in contrast to quantity conditions in processes [Forbus, 1984] which only control the activity of a process, but not the existence of the process.

Model-as behavioral constraints are constraints on the behavioral context of a component that enforce the selection of additional component CDBs. For example, the **model-as** constraint:

```
(implies
  (>= (* (voltage-difference ?object)
         (current (elec-term-1 ?object)))
      (elec-power-threshold ?object))
  (model-as ?object thermal-resistor))
```

in the **resistor** CDB states that if the dissipated power exceeds a threshold, then it must be explicitly modeled by modeling the component as a **thermal-resistor**.

Behavioral constraints can be viewed as enforcing the selection of significant CDBs. Significance is measured with appropriately set thresholds. Thresholds can be either preset or computed dynamically. A threshold of 2300 for Reynolds number, that distinguishes laminar flow from turbulent flow, is a widely used preset threshold. Other thresholds can be preset by an engineer from common practice. Thresholds can also be set dynamically based on the evolving device model and knowledge of acceptable tolerances on certain parameters (see [Shirley and Falkenhainer, 1990; Nayak, 1991] for some initial work).

Expected behavior

The expected behavior of a device is an abstract, possibly incomplete description of *what* the device does (but not *how* it does it). We use expected behaviors to capture, in part, what is commonly referred to as the *function* of a device. For example, stating that the device in Figure 1 is a temperature gauge indicates that the device model must explain how the temperature of the thermistor determines the angular position of the pointer. Expected behaviors can also provide abstract descriptions of device behaviors that would not normally be considered the device's primary function. For example, to assist a design engineer select dimensions for the temperature gauge wire, the device model must explain how the wire's length and cross-sectional area affect the angular position of the pointer. The most common expected behavior descriptions are input/output descriptions of device behavior.

Knowledge of the expected behavior is commonplace and almost always available either directly from the user, from the description of the problem to be solved, or from the context in which the device operates. For example, device names, such as light bulb, vacuum cleaner, and disk drive are widely used and all are associated with expected behaviors. Or suppose we want to know if a disk drive can be used as a door stop. The expected behavior—to stop the door from shutting—suggests that the disk drive model should focus on its kinematic and dynamic properties as an object, not its information retrieval properties! Expected behaviors are an essential component of a device description and play an important role in focusing the model selection process. Without it, all consistent device models are equally plausible: the disk drive as an information retrieval device, a heating device, or a door stop.

We specify expected behaviors with causal relations between parameters. Hence, expected behaviors specify which component parameters *must* appear in the device model, and the causal relations between them. For example, the temperature gauge's expected behavior, representing its primary function, is:

```
(causes (temperature thermistor)
        (angular-position pointer))
```

which says: the thermistor model must include a **temperature** parameter; the pointer model must include an **angular-position** parameter; and the model must explain how the thermistor's **temperature** determines the pointer's **angular-position**.

Modeling program

In this section we describe an implemented polynomial-time algorithm to find a simplest adequate model using structural, behavioral, and expected behavior constraints. A device model is said to be *adequate* when (a) it explains the expected behavior; (b) the structural and behavioral **model-as** constraints of each component CDB are satisfied;[4] and (c) each component model includes exactly one CDB from each required assumption class. A device model M_1 is said to be *simpler than* a device model M_2 if M_1 models fewer phenomena, more approximately: for each CDB C selected for component I in M_1, either C is selected for I in M_2 or C' is selected for I in M_2 and C is an approximation for C'.

Finding a simplest model that satisfies the expected behavior is in general intractable. However, Nayak [Nayak, 1992a] shows that this problem can be solved efficiently if all the approximation relations between CDBs are *causal approximations*. Briefly, when all the approximations are causal, the causal relations entailed by a simpler model are a subset of the causal relations entailed by a more complex model. Causal approximations are both natural and common in modeling the physical world. Our program assumes that all approximations are causal approximations.

The program's inputs are a device description, an expected behavior, and threshold values. The device description specifies the device's structure—its components and the structural relations between them—and any user-selected CDBs for each component. The program produces a simplest adequate model by first identifying an adequate model and then simplifying it. We describe these two phases next.

Identifying an adequate model

An adequate model is identified in four steps. The first step augments the initial device description to include all expected behavior parameters. The second and third steps augment the device model using the **model-as** structural and behavioral constraints, respectively. The fourth step checks the expected behavior. We now describe these steps and the flow of control between them.

[4]Structural **model-as** constraints are based on the heuristic that every component interaction that *can* take place *must* be modeled, even if the interaction is irrelevant. Hence, we do not require that adequate models satisfy structural **model-as** constraints.

In the first step, the program checks if the initial device model contains all expected behavior parameters. If a component parameter is missing, the program searches the **possible-models** of that component for the most general CDB that provides the required parameter.[5] Only CDBs whose structural and behavioral preconditions are satisfied are considered in this search. The resulting CDB is added to the component's model, together with the most accurate CDB of any **req-assumption-classes**.

For example, the expected behavior of the temperature gauge requires a **temperature** parameter for the thermistor and an **angular-position** parameter for the pointer. This can be achieved by adding the **thermal-thermistor** CDB to the thermistor model and the **rotating-object** CDB to the pointer model.

In the second step, the program checks the structural **model-as** constraints of each component model. If a constraint is not satisfied, then it means that the device model does not include a required CDB for some component. The program searches the **possible-models** of that component for the most general CDB that is a specialization of the required CDB.[5] The resulting CDB is added to the component's model, together with the most accurate CDBs of any **req-assumption-classes**. This can result in additional **model-as** structural constraints being violated. Hence, this step is repeated until all structural **model-as** constraints are satisfied.

For example, modeling the thermistor as a **thermal-thermistor** requires electrical models for the battery and the wire. Hence, the battery is modeled as a **voltage-source** and the wire as an **electrical-conductor**. The wire model is further augmented with the **resistor** and **temp-dep-resistor** CDBs to satisfy the **req-assumption-classes** constraints. Modeling the pointer as a **rotating-object** requires a kinematic interaction with the bimetallic strip, which can be satisfied by modeling the latter as a **thermal-bms**.

In the third step, the program uses the above device model to generate the behavior. It uses this behavior to check the behavioral **model-as** constraints of each component model. This check is exactly analogous to step two. If CDBs are added to any component model, the program repeats steps two and three until all structural and behavioral **model-as** constraints are satisfied.

For behavior generation the program computes the order of magnitude of each parameter using a novel technique described in [Nayak, 1992b]. Briefly, the order of magnitude, $om(q)$, of a quantity q is defined as: $om(q) = \lfloor \log_{10} |q| \rfloor$. A set of rules propagate exogenous orders of magnitudes to dependent parameters. For example, the rule:

$$om(a) + om(b) \leq om(a * b) \leq om(a) + om(b) + 1$$

propagates orders of magnitude to a product. The order of magnitude behavior is at the right level of detail:

[5]Ties are broken arbitrarily.

being qualitative, it does not require exact numerical values for exogenous parameters. In addition, it provides valuable numerical information not available from purely qualitative behaviors [Bobrow, 1984].

For example, assuming that the orders of magnitude of the `resistance` of the wire and of the thermistor is 2, and the `emf` of the battery is 1, the order of magnitude of the heat generated in the wire is predicted to be between -3 and 1. If the order of magnitude of the `elec-power-threshold` of the wire is set to be less than or equal to 1, the `model-as` behavioral constraint in the `resistor` CDB (Figure 3) requires that the wire be modeled as a `thermal-resistor`.

In the fourth step, the program determines if the expected behavior is satisfied by first computing the causal ordering [Iwasaki and Simon, 1986; Iwasaki, 1988] of the device model parameters, using the device model equations. The causal ordering is used to check if the expected behavior is satisfied. If it is satisfied, the model is adequate. Otherwise, the program augments the device model with an additional CDB for some component, and repeats steps two through four, until it finds an adequate model. The additional CDB is selected from the more special CDBs that could have been used to satisfy constraints in the above steps, but were not. If no additional CDB can be added, there is no adequate model, and the program reports failure. For example, the model generated above does satisfy the expected behavior and hence is adequate.

Simplifying the model

The adequate model identified above can be more complex than necessary for one of three reasons: (a) for each required assumption class, the program adds in the most accurate CDB, even though a more approximate (simpler) CDB might do; (b) since the structural `model-as` constraints are heuristic, the CDBs added to satisfy them may not be necessary; and (c) unnecessary CDBs may have been added in step four. A model can be simplified by applying one of the following two simplification operators: (a) replace a CDB by one of its immediate approximations; and (b) altogether remove a CDB. The program simplifies the adequate model found above by applying a sequence of simplification operators, while preserving adequacy. When all simplifications of a model are not adequate, the program terminates, and returns that model as a simplest adequate model. For example, `temp-dep-resistor` can be replaced by `constant-resistance` in the wire model.

The application of different sequences of simplification operators can result in different models. However, the different models differ in features deemed to be insignificant by the behavioral constraints and thresholds, and hence the program returns the first simplest adequate model it finds.

Implementation

We have constructed a library of 25 component types, including wires, bimetallic strips, springs, and permanent magnets. The CDB library consists of approximately 150 CDBs including descriptions of electricity, magnetism, heat, and the kinematics and dynamics of one-dimensional motion (including both rotation and translation). Each component has an average of 23 CDBs describing different aspects of its behavior.

The program is implemented in Common Lisp and has been tested on ten electromechanical devices drawn from [Artobolevsky, 1980; Macaulay, 1988; van Amerongen, 1967]. The devices range in complexity from 10 to 54 components per device, and include different types of temperature gauges, thermostats, relays, and workpiece inspection devices. In all cases the program constructs a model in 0.5 to 8 minutes on an Explorer II. The choice of devices and the scenarios of our experiments illustrate a number of points:

1. Device descriptions can include irrelevant information, e.g., in our representation of the temperature gauge, we include the atmosphere which can thermally interact with the wire and with the battery. The program correctly disregards the thermal interaction between the atmosphere and the battery.
2. Similar components in different devices are modeled differently. For example, in the temperature gauge the coil of wire is modeled as a resistor generating heat, while in a galvanometer the coil of wire is modeled as an electromagnet. In all cases our program correctly identifies the relevant models.
3. Models of differing complexity are built depending on the threshold settings. For example, if the `voltage-diff-threshold` of an `electrical-conductor` is low enough, the program models it as a `resistor`, even if modeling it as a `perfect-conductor` will explain the expected behavior.
4. The same device can have different expected behaviors, e.g., the effect of the wire's dimensions on the angular position of the pointer. The program correctly constructs different models to explain different expected behaviors.

Currently, the model selection program has two main limitations. First, behavior generation does not involve any integration over time. Instead, the order of magnitude values at the beginning of an interval are used as the behavior throughout the interval. A consequence is that expected behaviors have to be specified for each operating region. Of course, this means that different models can be built for each operating region, rather than requiring a single model for all operating regions. Second, expected behavior constraints are limited to causal relations between parameters.

Related work and conclusions

Falkenhainer and Forbus [Falkenhainer and Forbus, 1991] select models by *compositional modeling*. Each

model is conditioned on a set of simplifying and operating assumptions. A set of constraints govern the use of simplifying assumptions. These constraints are similar to our structural constraints (though the latter are only heuristics used to find an initial adequate model). In addition, we have identified a useful source of these constraints—the observation that components must be modeled in a compatible way. Operating assumptions are similar to behavioral constraints. An important difference is that while we generate an order of magnitude behavior, they generate either a purely qualitative or a purely quantitative behavior. As with our use of the expected behavior, they use a user query to generate an initial set of simplifying assumptions. However, in addition, the expected behavior provides feedback on the choice of models—an adequate model's equations must subsume the expected behavior. Finally, their model selection algorithm is based on *dynamic constraint satisfaction* [Mittal and Falkenhainer, 1990], while we exploit causal approximations to develop a polynomial time model selection algorithm.

Addanki *et al* [Addanki *et al.*, 1991] discuss techniques for selecting models of acceptable accuracy. The accuracy of a model is determined either by asking a user, or by using consistency rules. Consistency rules are similar to our behavioral preconditions. If a model's accuracy is unacceptable, a set of domain-dependent *parameter change* rules select a more accurate model. They start the analysis with the simplest model, and switch models until a model of acceptable accuracy is found. While we have not addressed model switching, our techniques can be used to make a more intelligent choice of an initial model, using contextual modeling constraints that are absent in their system.

In [Weld, 1990], Weld shows that, when models can be formalized as *fitting approximations* of one another, the parameter change rules used above can be replaced by a domain-independent technique for model switching. Most fitting approximations are also causal approximations, and hence his model switching techniques can be incorporated into our system.

In conclusion, having multiple models for individual components is necessary to account for the different assumptions, perspectives, and purposes that determine the adequate device model. This paper shows how the context in which the device and its components operate provide a powerful guide for the model selection process. We introduced *context-dependent behaviors* (CDBs), a component behavior model representation for encapsulating contextual modeling constraints. We showed how CDBs are used in the automated selection of device models by exploiting constraints from three sources: the structural and behavioral contexts, and the expected behavior of the device. We tested our ideas with an implementation.

We believe that our modeling paradigm will prove to be useful for a variety of tasks including analysis, design, and tutoring. The compositionality of CDBs and the many-to-many relationship between components and CDBs provides modeling flexibility, and expected behaviors allow teleological reasoning. CDBs provide a uniform mechanism to represent and reason about the structure, behavior, and function of a device. Future work will involve handling a wider range of expected behaviors, including behaviors over time, and the dynamic setting of thresholds.

Acknowledgements

We would like to thank Edward Feigenbaum, Richard Fikes, Brian Falkenhainer, Renate Fruchter, Andy Golding, Nita Goyal, Yumi Iwasaki, Rich Keller, Alon Levy, Rajan Ramaswamy, Rich Washington, Dan Weld, Michael Wolverton, and the anonymous reviewers for useful discussions and for comments on earlier drafts.

References

Addanki, S.; Cremonini, R.; and Penberthy, J. S. 1991. Graphs of models. *Artificial Intelligence* 51:145–177.

Artobolevsky, I. I. 1980. *Mechanisms in Modern Engineering Design*, volume 5. Mir Publishers, Moscow.

Bobrow, D., editor 1984. *Qualitative Reasoning About Physical Systems.* North-Holland.

Falkenhainer, B. and Forbus, K. D. 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

Forbus, K. D. 1984. Qualitative process theory. *Artificial Intelligence* 24:85–168.

Iwasaki, Y. and Simon, H. A. 1986. Causality in device behavior. *Artificial Intelligence* 29:3–32.

Iwasaki, Y. 1988. Causal ordering in a mixed structure. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* 313–318.

Macaulay, D. 1988. *The Way Things Work.* Houghton Mifflin Company, Boston.

Mittal, S. and Falkenhainer, B. 1990. Dynamic constraint satisfaction. In *Proceedings Eighth National Conference on Artificial Intelligence.* 25–32.

Nayak, P. P. 1991. Validating approximate equilibrium models. In *Proceedings of the 1991 Model-Based Reasoning Workshop.*

Nayak, P. P. 1992a. Causal approximations. In *Proceedings of the Tenth National Conference on Artificial Intelligence.*

Nayak, P. P. 1992b. Order of magnitude reasoning using logarithms. Technical Report KSL 92-29, Knowledge Systems Laboratory, Stanford University.

Shirley, M. and Falkenhainer, B. 1990. Explicit reasoning about accuracy for approximating physical systems. In *Working Notes of the Automatic Generation of Approximations and Abstractions Workshop.* 153–162.

van Amerongen, C. 1967. *The Way Things Work.* Simon and Schuster.

Weld, D. S. 1990. Approximation reformulations. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* 407–412.