Compositional Modeling of Physical Systems

Brian Falkenhainer
System Sciences Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto CA 94304

Kenneth D. Forbus
Qualitative Reasoning Group
Beckman Institute, University of Illinois
405 N. Mathews Street, Urbana IL 61801

Abstract

Automating analysis of physical systems requires techniques for managing complexity and finding an appropriate model for an analysis. *Compositional modeling* is a strategy for organizing multi-grain, multi-perspective models of physical phenomena which addresses these problems. In this paper, we identify several limitations with the approach presented in [7] and describe important extensions that address these limitations. First, we separate ontological commitments (e.g., energy flows, contained stuffs, etc) from granularity commitments. We present a new framework for explicitly reasoning about the appropriate ontology and the structural nesting of systems. Second, we provide a generalized modeling language which enables the compositional modeling approach to be applied to either quantitative or qualitative models. We show that model composition cannot *a priori* determine that a quantitative model will be sufficiently accurate. However, we show that by using explicit modeling assumptions, inaccurate models may be detected and these inconsistencies can be used to focus a new round of model composition, leading to using more accurate, but more costly, models only when they are needed. We describe an implemented algorithm for model composition, which can automatically ascertain the appropriate modeling assumptions to answer simple questions about physical and functional characteristics of its operation.

Submitted to QP-90

1 Introduction

This paper describes recent progress in our *compositional modeling* framework for organizing models of continuous physical systems. In previous work, we described how to organize large-scale qualitative models [7], so that one could automatically select an appropriate set of modeling assumptions to answer a given query. We organized models as *operating blocks*, which describe a system or subsystem at a uniform level of detail, and *functional blocks*, which hides internals and only has input-output behavior. Coherence was enforced by finding a single operating block which could serve as a focus of attention, and modeling all of its subsystems as functional blocks.

As we built more models, however, we discovered this decomposition is fundamentally flawed. It actually confounds several distinct changes in modeling assumptions, which this paper disentagles. First, modeling assumptions must control the amount of structure to be reasoned about. We show how *grain assumptions* can be used for this. Second, modeling assumptions must control the point of view taken on a system. *Perspective assumptions* control this. Whether or not a furnace is considered from a mass-flow or an energy-flow perspective, for example, is independent of the choice of whether or not it should be treated as a black box. Furthermore, we discovered that as the number of alternate models grew, it was important to inform the model composition algorithm about what choices were necessary to enforce coherence. To do this we adapted the notion of *assumption classes* from [1]. This paper describes our new taxonomy of modeling assumptions, describes a formal notion of system for controlling the granularity of an analysis, and extends our model composition algorithm to exploit these representational extensions.

This paper also describes our work towards a general-purpose modeling language which enables compositional modeling of quantitative as well as qualitative models. This extension required focusing on techniques for carefully verifying models. While our model composition algoirthm is sufficient to ensure that an equational model is relevant, we show that it cannot *a priori* determine that it will be sufficiently accurate. However, we show that by using explicit modeling assumptions, inaccurate models may be detected and these inconsistencies can be used to focus a new round of model composition, leading to using more accurate, but more costly, models only when they are needed.

2 Overview of the compositional modeling process

A general-purpose *domain model* is used to describe a class of related phenomena or systems. A domain model consists of a set of *elementary domain models*, each describing some fundamental piece of the domain's physics, such as processes (e.g., liquid flow), devices (e.g., transistor), and objects (e.g., container). We call the system or situation being modeled the *scenario*, and its model the *scenario model*. The scenario model is built by instantiating and composing elementary descriptions from the domain model.

Modularity is crucial in controlling complexity. By decomposing knowledge of a domain into small fragments, the task of generating and using a large domain model can be greatly simplified. This observation is the heart of the compositional modeling strategy. A domain model consists of elementary models which can be instantiated and assembled as needed to form a scenario model based on a combination of the properties of the physical scenario and the modeling assumptions appropriate to the task. This enables implicit representation of a vast space of diverse scenario models and allows each elementary model to be reused in a variety of settings. *Simplifying assumptions* are used to state the conditions under which elementary models are appropriate. These include their underlying approximations, perspectives, and granularity. *Operating assumptions* are used to

describe the kinds of behaviors relevant to a task (e.g., steady state behavior). Because the modeling assumptions underlying the elementary models are not always independent, domain-specific constraints are used to ensure that only coherent scenario models are formed.

In this approach the modeling process consists of four stages: (1) encoding the fundamental principles and phenomena of a domain, with each elementary domain model explicitly conditioned on a set of objects, assumptions, and operating regions; (2) composition of the simplest, coherent scenario model relevant to the needs of the task; (3) analysis using the model; and (4) examination of the model's underlying assumptions to see if an alternate model is required. This paper focuses primarily on methods for composing useful scenario models and the organizing the reasoning principles needed to support this process. We use standard qualitative and quantitative simulation techiques as our analysis method. Further, we only consider model switching due to modeling assumption violations uncovered during the analysis phase.

3 Model Organization

This section describes how to organize a domain model using the compositional modeling strategy. We focus on the use of simplifying assumptions for controlling a scenario model's granularity, ontology, and approximations. Operating assumptions are not discussed further (see [7, 8] for more details). Examples from a thermodynamics model of a steam propulsion plant are used throughout for illustration.

3.1 Simplifying assumptions

Predicating elementary domain models on simplifying assumptions provides the ability to ignore that which is irrelevant. For uniformity, we require that all simplifying assumptions take the form

`CONSIDER(`⟨*AsnType*⟩`(`⟨*system*⟩`))`

where ⟨*AsnType*⟩ is a predicate denoting the specific kind of assumption and ⟨*system*⟩ is the subject of the assumption. The collection of `CONSIDER` assumptions form the groundwork for any particular analysis. They are classified into three categories, as discussed below.

3.1.1 Grain assumptions

Crucial to analyzing large systems is the fact that not all objects in the system need be considered for every analysis task. First, objects outside the current area of concern are simply ignored. Second, abstractions are used to allow collections of objects to be considered as a single, aggregate entity. Control over which objects to explicitly consider in an analysis is organized around assumptions recognizing their existence, which we call *grain assumptions.*

We assume the objects in the scenario are organized into *systems.* A system is either a primitive object or a named collection of constituent systems. For example, a container is a primitive object, and the boiler assembly is not, since it consists of a furnace, boiler, superheater. The relation `Part-of` holds when one system is part of another. Thus

```
Part-of(boiler,boiler-assembly)
```

indicates that the boiler is part of the boiler assembly. The `Part-of` relation is limited to a system and its immediate subsystems (i.e., it is not transitive). We say that system s_1 *contains* system s_2 if either (a) s_2 is a part of s_1 or (b) s_2 is a part of some system s_3 which itself is contained in s_1. This relationship is indicated formally by the transitive relation `Sys-Contains`. For example,

`Sys-Contains(steam-plant, furnace)`

indicates that the `steam-plant` system contains the `furnace`.

We currently assume that systems always form a strict hierarchy. The root of this hierarchy, which contains all the objects in the scenario, is always a system called `:scenario`.

The following form is used to state grain assumptions about systems:

`CONSIDER(existence(`⟨*system*⟩`))`

If true, then the existence of ⟨*system*⟩ is considered and it is placed within the scope of the current analysis. This means some model for it must be included in any coherent scenario model. For example,

`CONSIDER(existence(boiler))`

forces a model of the boiler to be included in an analysis, rather than focusing on some subsystem of it (such as the steam tubes) or treating the boiler assembly as a black box.

The constraint governing grain assumptions is slightly subtle. The intuition is that one cannot arbitrarily choose a set of parts to model in isolation. Instead, given interest in some parts, one has to consider enough of the system so that all of the relationships involving the parts in focus are included. For example, thinking about how changes in an automobile transmission can affect the wheels of a car doesn't make much sense unless one takes the drive shaft and differential into account.

In the simplest case, if we consider two objects that are part of the same system, then all of that system's components must be considered:

$$\texttt{Consider(existence}(s_1)) \wedge \texttt{Consider(existence}(s_2)) \wedge \texttt{Part-of}(s_1, s_0) \wedge \texttt{Part-of}(s_2, s_0) \rightarrow \texttt{Consider(components}(s_0))$$

$$\forall s_i [\texttt{Consider(components}(s_0)) \wedge \texttt{Part-of}(s_i, s_0) \rightarrow \texttt{Consider(existence}(s_i))]$$

In general, a more sophisticated line of reasoning is required. A *covering system* is defined to be any system that contains all systems of interest.

$$\texttt{Covering}(s_c) \Leftrightarrow \forall s_i \, [\texttt{Consider(existence}(s_i)) \rightarrow \texttt{Sys-Contains}(s_c, s_i)]$$

Many covering systems are too large to be useful; `:scenario`, for example, is always a covering system. Therefore we define a *minimal covering system* to be a covering system that contains no smaller covering system. Specifically,

$$\texttt{Minimal-Covering}(s_c) \Leftrightarrow [\texttt{Covering}(s_c) \wedge \neg \exists s_i [\texttt{Covering}(s_i) \wedge \texttt{Sys-Contains}(s_c, s_i)]$$

During model composition (Section 4.2), the notion of a minimal covering system is used to help select a coherent set of grain assumptions given an initial set of query terms.

3.1.2 Ontology assumptions

Different tasks can demand carving the world up in different ways. Four ontologies are presently considered. The *contained stuff* ontology [15, 10] is used to qualitatively model static and dynamic fluids and their containers (similar to the fluid control volumes of classical thermodynamics). The following form is used to predicate a description on this ontology:

`CONSIDER(fluid-cs(`⟨*system*⟩`))`

The *energy-flow* ontology analyzes the flow of energy (both heat and work) through a system (c.f., energy flow diagrams). The *molecular collection* ontology [2] follows the movement of fluid particles during flow. Finally, *mechanics* is used for the dynamic analysis of mechanisms.

We assume all ontological assumptions are implemented with the following discipline. First, ontological assumptions are global and inherited by all systems being considered. Thus, any ontological assumptions that are made, and there must be at least one, must be made for the root system, `:scenario`. Second, when consistent, multiple ontological assumptions may hold. For example, an energy flow analysis often occurs with a mass flow analysis.

3.1.3 Approximations and Perspectives

Approximations are used to construct simplified and (typically) easier to use models at the cost of reducing accuracy. The approximations underlying each elementary model are explicitly stated to enable selection between possible alternatives based on the scenario's operating conditions and the accuracy needs of the analysis task. Approximations used in our models include incompressible fluids, inviscid flows, inelastic objects, and frictionless motion.

Perspective assumptions control a large variety of modeling choices. Some perspectives indicate how particular objects are modeled. For example, a fluid valve can be modeled either as a discrete, on/off switch or as resistance that can vary continuously. Other perspectives represent simplifying assumptions about the structure of the environment. For example, our fluid models typically assume level fluid paths.

The constraints on approximation and perspective assumptions are domain-specific. For example, in our models it does not make sense to consider the portals that connect a container to a fluid path unless one is willing to consider the geometric properties of the container.

3.1.4 Assumption classes

Some collections of assumptions represent mutually exclusive, alternative ways to model the same aspect of an object or phenomenon. To represent this important relationship, approximation and perspective assumptions are organized into sets called *assumption classes* [4, 1]. Assumption classes are declared with the form

`(defAssumptionClass` ⟨*class-form*⟩ ⟨*assumption-forms*⟩`)`

The order of the assumptions is important: models based on assumptions earlier in the list are less costly than models based on assumptions later in the list.[1] We say that the assumption class is *active* when ⟨*class-form*⟩ holds. When a class is active, one and only one of the assumptions associated with that class must hold in the scenario model. Additional information about the

[1] Having a context-independent cost estimate is something of an oversimplification, but has proved quite useful. The extension to a scheme where a task-specific cost procedure is provided to induce an ordering on assumptions is straightforward, and hence we have chosen not to complicate our presentation.

conditions under which each assumption class is most relevant or is inappropriate can then be specified independently.

For example, our models of liquid flow include the **fluid-viscosity** assumption class, which controls how the viscosity of a fluid flowing through a path is modeled. Its definition and the condition that it becomes relevant for each path through which liquid is considered to be flowing are stated as follows:

```
(defAssumption-class (fluid-viscosity ?path)
     ((CONSIDER (inviscid ?path))
      (CONSIDER (viscous ?path))))

(<== (fluid-viscosity ?path) ((process-instance fluid-flow ?pi)
                              (?pi PATH ?path)))
```

Additional constraints are then placed on the use of these assumptions. For example, questions concerning head loss should rule out the **inviscid** (frictionless) assumption.

3.2 A language for elementary domain models

We define elementary domain models using a generalization of the modeling language developed for QP theory (see [8]). The principle construct is **defModel**, which defines an elementary domain model. It has the following syntax:

(defModel ⟨*name-form*⟩
    Individuals ⟨*i-spec*⟩
    Assumptions ⟨*assumptions*⟩
    OperatingConditions ⟨*assertions*⟩
    Relations ⟨*assertions*⟩)

where ⟨*name-form*⟩ is an expression with variables which provides a term designating an instantiation of this model and ⟨*i-spec*⟩ is a set of variables and restrictions on their potential bindings. The **Individuals** field describes the physical settings to which the model applies. The **Assumptions** field contains the simplifying assumptions the model relies upon. Given a set of objects and constants that match the individuals specification, we say the model is *applicable* for that collection. If furthermore the simplifying assumptions hold, then we say the model is *applied* for that collection, and becomes part of the scenario model. For example, a model of a string under tension is not applicable to analyzing a steam propulsion plant, while a model of boiling is applicable but need not be applied.

A model can apply to a scenario yet not be imposing constraints. For example, a model of the process of liquid flow can be applied even when the flow is not occuring (say if a valve in the fluid path becomes closed). Prerequisite behavioral conditions are contained in the **OperatingConditions** field. We say a model is *active* when it is both applied and its operating conditions hold. When the model is active, the statements in the **Relations** field hold. The model's relations include the constraints imposed by the model, both qualitative and quantitative, and any other consequences which directly follow as a consequence of the model being active.

While any expression may appear in the relations field, expressions declaring quantities are of particular interest. The predicate **Quantity** is used to state that an object(s) has a quantity of a particular type:

(Quantity (⟨*quantity-type*⟩ . ⟨*objects*⟩))

Because it is generally easy to extract from a task specification or query a set of quantities which must exist for the task to even make sense, quantity declarations are the major link for matching models to task requirements.

4 Model Composition

Given a question (e.g., analysis goal, tutorial query, etc.), our task is to construct a scenario model that (a) suffices to answer to question and (b) minimizes extraneous details and problem solving effort. Sufficiency in turn has two aspects. The first is concerned with *aboutness*, that is, the scenario model includes all the aspects of the system required to perform the analysis. The second is concerned with *accuracy*, that is, the scenario model must contain enough information to provide a suitable answer. In this paper we focus on the aboutness criterion (see [9] for more on the accuracy criterion). Clearly, sufficiency and minimality are somewhat in conflict. This is resolved by treating minimality as the selection criterion for models satisfying the absolute sufficiency requirement.

Our approach to model composition operates under two restrictions. First, we require that the information needed to derive an appropriate scenario model can be gleaned solely from the query and the contents of the models, without recourse to other domain-specific conventions or default patterns of communication. Second, we only address the problem of selecting appropriate simplifying assumptions; while some operating assumptions are forced as a by-product of selecting approximations, the general problem of finding appropriate operating assumptions is beyond the scope of this paper.

The basic idea of our algorithm is this: While our goal is to produce a scenario model, explicit reasoning about combinations of elementary domain models to ascertain which of them are consistent and sufficient is expensive and unnecessary. Rather, we find which elementary models are applicable and reason instead about their underlying modeling assumptions, since the number of modeling assumptions is much smaller than the number of elementary domain models. The final output of the algorithm is a consistent set of ground modeling assumptions that, together with the scenario description, entail a minimal, sufficient scenario model.

Our algorithm assumes an underlying assumption-based truth maintenance system (ATMS) [4].[2] Each statement in the problem solver's database has a corresponding ATMS *node*. Some nodes are specifially designated as *assumptions*, which we assign to each instantiated modeling assumption. An *environment* is a set of assumptions, logically equivalent to a conjunction of assumptions. The consistent, complete, and minimal disjunction of environments under which each node holds is called its *label*. If some environment in a node's label is a subset of a given environment, then the corresponding statement is believed under the assumptions which comprise the given environment.

The labels and dependencies maintained by the ATMS are used to compute a *modeling environment*, the conjunction of modeling assumptions needed to produce an appropriate scenario model. After using the scenario description to fully instantiate the domain model (i.e., find all applicable models), model composition consists of four steps: *query analysis*, *object expansion*, *candidate completion*, and *candidate evaluation and selection*. The remainder of this section describes each step in more detail. Steps 2 and 3 are performed for each seed environment and are described from the perspective of a single candidate seed.

[2] We use the ATMS vocabulary to describe our query analysis procedure. The actual implementation could be modified, at possibly reduced efficiency, to work with other kinds of TMS'.

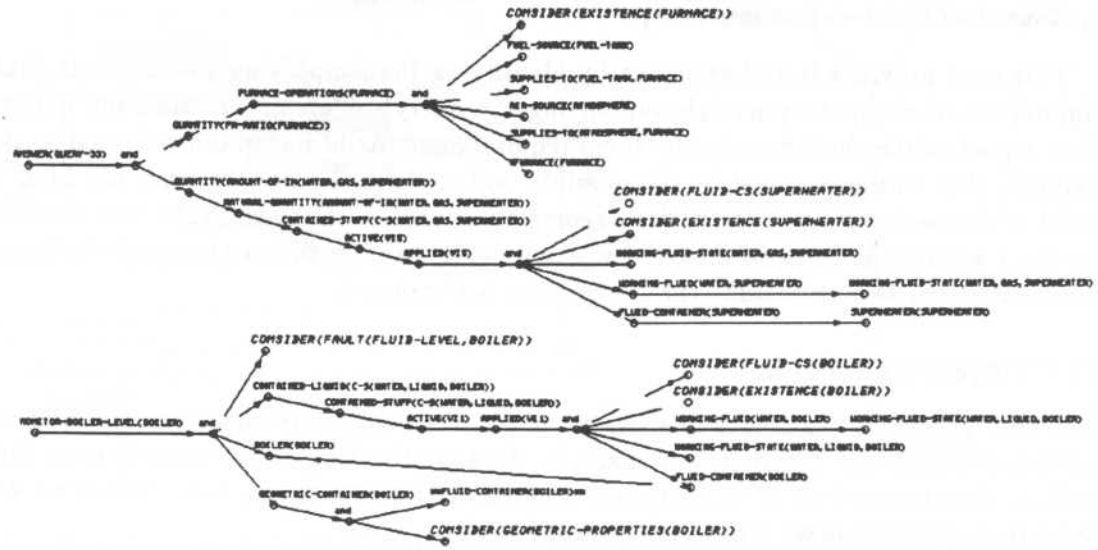

Figure 1: TMS dependency structure showing the relationship between terms in the models and the modeling assumptions that enable their use. `Answer(query-33)` represents the conjunction of the query expressions. `Monitor-boiler-level(boiler)` shows what would have been needed to also include the boiler's fault model and why it was deemed unnecessary for the query.

4.1 Query analysis

We assume that whatever the initial form of the query, it can be decomposed by a *query elaboration procedure* into a set of ground expressions $\mathcal{Q}$, each having referents in the fully instantiated domain model. These ground expressions provide the input for this step of model composition and contain the conjunction of objects, quantities, and relations of interest.

Given query expressions $\mathcal{Q} = \{e_1, \ldots, e_n\}$, we construct *seed candidate modeling environments* (hereafter "seeds") as follows. Let `QUERY` be a new ATMS node. Justify `QUERY` by the conjunction of the expressions in $\mathcal{Q}$ and compute its label $\mathcal{L}_\mathcal{Q}$. Since the only assumptions in the ATMS database are modeling assumptions, every environment in `QUERY`'s label is a seed. Each seed in $\mathcal{L}_\mathcal{Q}$ provides an alternate set of necessary simplifying assumptions. That is, since we have included all potential simplifying assumptions in the instantiated domain model, every minimal consistent combination of these assumptions that together entail the objects and properties mentioned in the query will appear as an environment in the label of the node `QUERY`. However, these seeds may not by themselves entail a coherent scenario model. The next two steps extend these seeds into coherent candidate scenario models.

As an example, suppose we are interested in the question "How does the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?". This query is transformed into the set of expressions representing the quantities of interest:

$\mathcal{Q}$ = {`Quantity(amount-of-in(water,gas,superheater))`,
`Quantity(FA-ratio(furnace))`}

The dependencies for the corresponding `QUERY` node are shown in Figure 1. Inspection of these dependencies produces the single candidate seed environment:

```
{Consider(existence(furnace)), Consider(existence(superheater)),
 Consider(fluid-cs(:scenario))}
```

This seed provides initial guidance by identifying the simplifying assumptions that entail the minimal set of elementary models supporting the query's expressions. Had the query mentioned other aspects of the boiler assembly, these requirements would have been reflected in the seed. For example, also stating interest in the possible boiler faults (i.e., water level too high or too low) would add consideration of the boiler's geometry as part of the seed. Note that this seed does not represent a coherent scenario model. For example, it assumes the existence of the furnace and the superheater, yet fails to include the intervening boiler object.

4.2 Object expansion

Each seed entails the existence of some set of objects. To ensure coherence we must determine if additional objects are required. For example, if we are thinking about steam flow in the boiler and turbine, then we need to think about the condenser and feed pump, too, so that we will correctly recognize that these flows are part of a closed cycle.

These decisions are made by finding the minimal covering system for the objects entailed by the seed. If the seed only requires a single object, then the minimal covering system will be that object and nothing needs to be added. Otherwise, the seed is extended according to the grain assumption constraints defined in section 3.1.

Resuming our example, recall that the candidate seed explicitly stated a need to consider the furnace and superheater. To identify the complete set of objects required to form a coherent scenario model from this seed, we first determine the seed's minimal covering system. For the `furnace` and `superheater` systems, this is the `boiler-assembly` system. From the grain assumption constraints, the following grain assumptions are then required (Figure 2):

```
{Consider(existence(furnace)), Consider(existence(superheater)),
 Consider(existence(boiler))}
```

The net effect is that the boiler has been added to the scope of consideration. Note that the steam plant's other primary components, such as the turbine and condenser assemblies, are ignored for this query. Further, additional detail within the selected systems is ignored, such as the furnace's fuel pump and exhaust manifold.

4.3 Candidate completion

At this stage our seeds have grown to include every object that must be included in a coherent domain model. However, this does not necessarily mean that we have chosen exactly how each object should be modeled. For example, modeling fluid flows requires decisions about paths, resistances, and flow regimes. The extra information needed to both determine what kinds of modeling assumptions need to be made and what the alternatives are is provided by assumption classes. Recall that an assumption class is considered *active* when its class form holds, and thus requires the problem solver to include one of its members in a scenario model. For each assumption class whose activity is entailed by the candidate seed environment, one of its assumptions is selected and added to the seed. Choices made for one assumption class may lead to inconsistencies or the activation of other assumption classes. The process of making and retracting choices, and satisfying constraints over a dynamically changing set of assumption classes is called *dynamic constraint satisfaction* [16].

The candidate completion procedure produces a set of candidate modeling environments. The scenario models entailed by these candidates are coherent, since they each entail all the objects

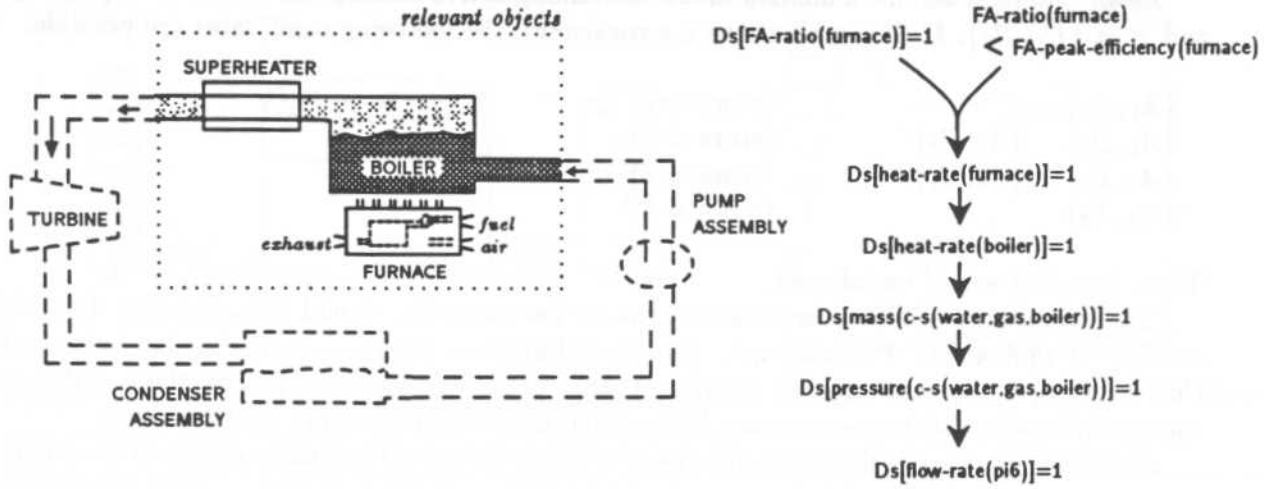

Figure 2: How does the furnace's fuel/air ratio affect the steam flowing in the superheater?

necessary to reason about the queried objects, and include a choice for all relevant modeling decisions.

In our continuing example, the developing seed is currently comprised of:

```
{Consider(existence(furnace)), Consider(existence(superheater)),
 Consider(existence(boiler)), Consider(fluid-cs(:scenario))}
```

This seed entails nine assumption class instances, corresponding to three general assumption class types. The first type requires a decision about whether or not to model the geometric properties of each fluid container whose existence is considered. Because there are three "fluid containers" (furnace, boiler, superheater), there are three **geometric-properties** assumption class instances, each of the form `Geometry(`⟨*can*⟩`)`. The second type of activated assumption class concerns the thermal properties of each object whose existence is considered. The third concerns the fluid resistance of each flow's path. In the next section, we describe our metric for evaluating the relative desirability of each choice. This metric is consulted by the candidate completion procedure so that all possibilities need not be generated.

4.4 Candidate evaluation and selection

Given a set of candidate modeling environments, the final step is to apply an evaluation metric to the candidates and select the "best" one to use in answering the query. What is "best" often depends on the details of the domain and task. We currently use a simple scheme that is based on crude estimates of model costs and has worked surprisingly well in our experiments.

The first stage prunes the set of candidates by retaining only those with the smallest number of objects. For example, if two candidates entail four objects and three candidates entail five objects, those entailing five objects would be deleted.

The second stage uses the ordering information in assumption classes to estimate overall simplicity. Recall that the choices in each assumption class are ordered in increasing complexity. Consequently, we assign a number to each choice corresponding to its place in its ordering. For each candidate these values are summed, and the candidate whose sum is the smallest is selected. In case of a tie, one of the minimal environments is selected at random.

As an example, assume a domain model containing active assumption classes $A = \{A_1, A_2, A_3\}$ and $B = \{B_1, B_2\}$. If all combinations are consistent, the following candidates are possible:

$\{A_1, B_1\}$		(score = 2)
$\{A_1, B_2\}$	$\{A_2, B_1\}$	(score = 3)
$\{A_2, B_2\}$	$\{A_3, B_1\}$	(score = 4)
$\{A_3, B_2\}$		(score = 5)

Thus, $\{A_1, B_1\}$ would be selected.

The limitations of this scheme are obvious. For example, should $\{A_2, B_2\}$ and $\{A_3, B_1\}$ be considered equivalent? Probably not. But the information required to distinguish between them (for instance, by ascertaining the relative expense of A_3 and B_2) depends on the specifics of the task and domain, and hence we must defer attempting to provide more guidance here.

Returning to our running example, the selected modeling environment describes the fluids and processes associated with the three components being considered, as well as their interactions, in isolation from the rest of the steam plant.

5 Model use, verification, and change

Simulation, both qualitative and quantitative, has been the principle use of the models we develop. Our qualitative simulations are carried out using `QPE` [11], an envisioner for QP theory. Once the model composition algorithm has generated an acceptable set of modeling assumptions, the scenario model they entail is built by `QPE`. `QPE` then uses this scenario model to produce an envisionment under the current operating assumptions. For example, the scenario model derived in the previous section produces an envisionment describing the various ways in which the furnace's fuel/air ratio can affect the amount of steam flowing in the superheater. For the case in which the furnace has been operating in a suboptimal, low F/A region, Figure 2 shows the resulting perturbation when the fuel/air ratio is increased. Briefly, an increase in F/A results in increased heat production, which results in increased steam production and increased boiler steam pressure, leading to an increased flow of steam through the superheater.

Our quantitative simulations are carried out by a simple simulator we have built around a fourth-order Runge-Kutta integration algorithm with adaptive step-size control (from [18, ch. 15]). The equations for the simulator are gathered from the scenario model, with some minor processing to get them into the form it expects. Currently we restrict an analysis to concern a single operating region – that is, all equations hold all of the time.

Because not all of the model's parameters are known beforehand, the initial scenario model may make invalid assumptions about the system's behavior. There are of course a variety of ways to verify a model, including comparing its results to observations or to other models. We focus here on internal consistency tests, since these are usually cheaper than external tests. Identifying an internally inconsistent qualitative model is simple: If the envisionment is empty, then the model is inconsistent.

One aspect of verifying a quantitative simulation involves checking to see if the behavior predicted by the scenario model violates its assumptions. To do this, we gather the set of *critical inequalities*. These are the set of inequalities that represent the conditions required by the model's simplifying assumptions. For example, the incompressible flow assumption requires that the flow's Mach number (ratio of velocity to speed of sound) is less than 0.3. If at any time during a simulation using the incompressible flow assumption its Mach number exceeds 0.3, the modeling environment is deemed inconsistent.

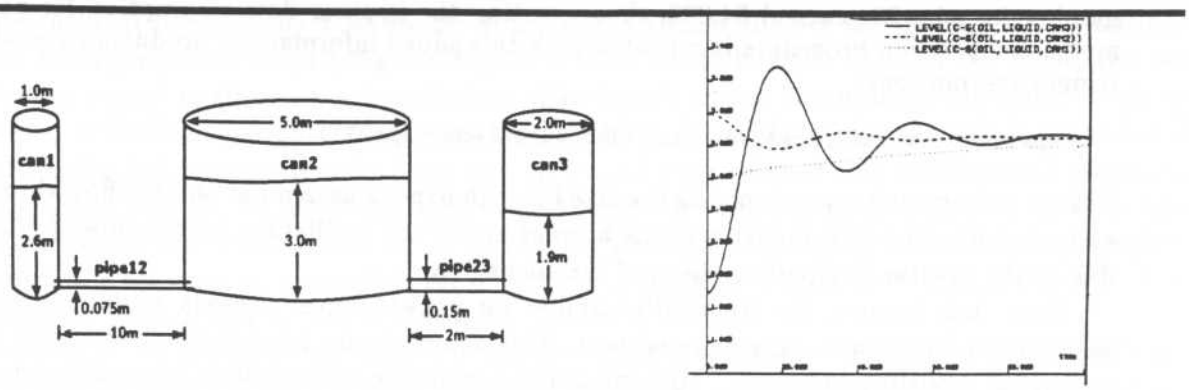

Figure 3: Two oil supply drums connected to a central reservoir. Find the level of the three containers as a function of time when the system is released from the given initial conditions.

The only recourse when an internal inconsistency is discovered is to look for a new modeling environment. The model composition process is repeated with new information about inconsistent modeling assumptions. We demonstrate this process in the next section.

6 Identifying an appropriate flow model

The compositional modeling strategy has been used to answer a number of qualitative and quantitative questions about a hypothetical steam propulsion plant (see [7, 9]). Here, we demonstrate its use on a quantitative analysis problem concerning a hypothetical set of lubrication tanks and pipes. Given two oil supply drums connected to a central reservoir (Figure 3), the task is to determine the behavior of the oil levels when the system is released from the initial condition. Our current fluid flow models represent various simplifications of the unsteady Bernoulli equation, which describes incompressible flow along a streamline:[3]

$$\frac{p_1}{\rho} + \frac{\overline{V}_1^2}{2} + gz_1 = \frac{p_2}{\rho} + \frac{\overline{V}_2^2}{2} + gz_2 + h_l + \int_1^2 \frac{\partial V_s}{\partial t} ds$$

where ρ is the density of the fluid, z_i is the height at point i, V_i is the fluid velocity at point i, and h_l is the head loss due to frictional effects. The formula used to compute h_l is dependent on the flow regime (laminar or turbulent), which is normally determined by Reynold's number ($Re = \rho \overline{V} D/\mu$). For low Reynold's numbers (low flow rates), the flow is laminar; for high Reynold's numbers (high flow rates), the flow is turbulent. Although the transition occurs over an interval, flow in pipes is generally taken to be laminar for Re<2,300.

Given only the initial conditions, our model composition algorithm lacks information about the Reynold's number for either pipe. Thus, it selects a simple modeling environment that includes assumptions of laminar flow:

```
Consider(laminar(pipe12)), Consider(laminar(pipe23))
Consider(incompressible-flow(pipe12)), Consider(incompressible-flow(pipe23))
```

The predicted level of each container as a function of time is shown in Figure 3. At this point, the system inspects the predicted behavior and finds one modeling environment violation: the Reynold's

[3] At the present time in our model development, we use the unsteady Bernoulli equation, which subsumes the steady and unsteady flow cases.

number for pipe23 reached 54,000, thus violating the laminar flow assumption for pipe23. The model composition procedure is repeated with this added information, producing a new set of flow regime assumptions:

```
Consider(laminar(pipe12)), Consider(turbulent(pipe23))
```

This modeling environment models the flow through pipe12 as laminar and the flow through pipe23 as turbulent. The new model predicts a lower amplitude oscillation for the flow through pipe23, due to the greater dissipative effects of turbulent flow.

Note that because the Reynold's number for pipe12 never exceeds 1,100, the laminar flow assumption for pipe12 remains consistent. This demonstrates an important attribute of the compositional modeling approach. By representing modeling assumptions as predications over individual objects and phenomena, a scenario model is able to represent the same type of object or phenomenon in different ways, depending on their individual conditions. This ability is crucial for analyzing large systems. For example, some wires in complex circuits must be modeled as transmission lines. The computational cost of considering all wires and connections as transmission lines is prohibitive, and modeling as few wires as possible that way is a necessity.

7 Related Work

The work closest to our own is the *graph of models* (GoM) approach [1], in which the space of possible models is represented explicitly as a graph. Each node represents a model of the system being analyzed, and each edge indicates which assumptions differ between the models it connects. What GoM refers to as a model is what in the compositional modeling framework is a complete and consistent set of modeling assumptions (i.e., a scenario model). To the extent that relevant, complete scenario models can be pre-enumerated, the GoM approach is faster, but incurs a potentially exponential increase in storage. The guiding principle behind the compositional modeling approach is the belief that there are too many useful scenario models to enumerate. This comes in part from our concern for grain, ontology, and operating assumptions, which significantly increases the space of possible models. Additionally, the number of influences acting on a system can significantly differ from one setting to the next (e.g., sum of forces, sum of fluid flows into and out of a container, etc). The compositional approach allows these effects to be folded into the scenario model as needed.

One very interesting aspect is Graph of Model's ability to reason about how changing assumptions affects model/observation discrepancies. Weld [19] presents a domain-independent approach that we believe could easily be adapted for compositional models. In particular, the set of modeling assumptions that specify a scenario model could be perturbed individually, generating "adjacent" alternatives by repeating our candidate completion procedure.

Davis' system for model-based diagnosis of digital circuits used multiple levels of structural descriptions to control search [3]. In many respects our use of grain assumptions and system distinctions is similar. However, we are able to focus in on individual objects in the system more easily, and do not require a fixed hierarchy of finer-grained models.

8 Discussion

We believe the compositional modeling strategy is an important step towards understanding how to build large-scale, multi-grain, multi-perspective models of physical domains that can be flexibly and efficiently applied to whatever task is at hand. We believe the insulation of the analyst from the details of the domain model our model composition algorithm provides is very important. For

tutoring tasks, it is obviously a necessity: If a student's knowledge of the domain were sufficient to select the appropriate simplifying assumptions, there would be little need for the tutor. But we believe even expert engineers will benefit from allowing the model composition algorithm to share the burden of finding the right foundation for an analysis. First, the engineer's task is simplified if she can specify just enough to make her intent clear, and leave the rest to a (mechanical) assistant. Second, in practice, all too often one finds models where the underlying assumptions made in one part of a model conflict with those made by another part (c.f., the negative water level example). The automatic selection of a complete set of explicit simplifying assumptions can help ensure that they are used consistently across an analysis.

However, much research remains. We are currently investigating two important extensions. First, in most domains there is a large gap between engineering drawings and the abstractions an engineer uses to decompose the system for analysis. For many tasks, the mapping from structural description to structual abstraction is *the* crucial step; doing it incorrectly can prevent consideration of important phenomena (such as ignoring resonance phenomena in the design of structures). The discipline of explicit modeling assumptions must be extended to this part of the modeling process, so that we can build engineering problem solvers whose analyses are trustworthy. Second, the current approach requires a query consisting of terms in the instantiated domain model. This provides an important framework for studying the problem of query analysis: given some analytic question, how can its form be used to suggest an appropriate model beyond simply looking for terms in the model. For example, asking when an oscillator will stop a-priori rules out a frictionless model.

9 Acknowledgements

The authors wish to thank Mark Shirley, Sanjay Mittal, and Johan deKleer for productive discussions about this work. John Collins provided valuable commentary and technical assistance. Significant portions of our thermodynamics model were developed in collaboration with John. We thank Mark Shirley for providing the Runge-Kutta code, who in turn obtained it from Elisha Sacks.

This research was supported in part by the National Aeronautics and Space Administration, Contract No. NASA NAG-9137, by the Office of Naval Research, Contract No. N00014-85-K-0225, and by an NSF Presidential Young Investigator Award.

References

[1] Addanki, S, Cremonini, R, and Penberthy, J. S. Reasoning about assumptions in graphs of models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989. Morgan Kaufmann.

[2] Collins, J and Forbus, K. Reasoning about fluids via molecular collections. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 590–594, Seattle, WA, July 1987. Morgan Kaufmann.

[3] Davis, R. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24, 1984.

[4] deKleer, J. An assumption-based TMS. *Artificial Intelligence*, 28(2), March 1986.

[5] deKleer, J and Bobrow, D. G. Qualitative reasoning with higher-order derivatives. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 86–91, August 1984.

[6] deKleer, J and Brown, J. S. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[7] Falkenhainer, B and Forbus, K. D. Setting up large-scale qualitative models. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 301–306, St. Paul, MN, August 1988. Morgan Kaufmann.

[8] Falkenhainer, B. and Forbus, K. D. Compositional Modeling: Finding the right model for the job. *Submitted to Artificial Intelligence, February, 1990.*

[9] Falkenhainer, B and Shirley, M. Explicit reasoning about accuracy for approximating physical systems. (*submitted for publication*), 1990.

[10] Forbus, K. D. Qualitative process theory. *Artificial Intelligence*, 24, 1984.

[11] Forbus, k. d. The Qualitative Process Engine in *Readings in Qualitative Reasoning about Physical Systems*, Weld, D. and de Kleer, J. (Eds.), Morgan Kaufmann, 1989.

[12] Forbus, K. D and Falkenhainer, B. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. (*submitted for publication*), 1990.

[13] Fox, R. W and McDonald, A. T. *Introduction to Fluid Mechanics.* John Wiley & Sons, New York, NY, third edition, 1985.

[14] Granet, I. *Fluid Mechanics for Engineering Technology.* Prentice Hall, 1989.

[15] Hayes, P. J. Naive physics 1: Ontology for liquids. In Hobbs, J and Moore, R, editors, *Formal Theories of the Commonsense World.* Ablex, 1985.

[16] Mittal, S and Falkenhainer, B. Dynamic constraint satisfaction problems. (*submitted for publication*), 1990.

[17] Murthy, S and Addanki, S. PROMPT: An innovative design tool. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 637–642, Seattle, WA, July 1987. Morgan Kaufmann.

[18] Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. *Numerical Recipes.* Cambridge University Press, 1986.

[19] Weld, D. Automated model switching: Discrepancy driven selection of approximation reformulations. Technical Report 89-08-01, Department of Computer Science and Engineering, University of Washington, 1989.

[20] Welty, J. R, Wicks, C. E, and Wilson, R. E. *Fundamentals of Momentum, Heat, and Mass Transfer (second edition).* John Wiley & Sons, New York, NY, 1984.