XEROX

Composition Modeling of Physical Systems

Brian Falkenhainer
Ken Forbus

SSL-91-95 [P91-00018]

System Sciences Laboratory
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Compositional Modeling of Physical Systems

Brian Falkenhainer and Kenneth D. Forbus

1.1 Introduction

This paper describes recent progress in our *compositional modeling* framework for organizing models of continuous physical systems. Previously we described how to organize large-scale qualitative models [FAFO88] to allow automatically composing domain model fragments into an appropriate task-specific model. We organized model fragments as *operating blocks*, which describe a system or subsystem at a uniform level of detail, and *functional blocks*, which hide internals and only have 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 was fundamentally flawed. It confounded several roles of modeling assumptions, which this paper disentagles with a new taxonomy. *Grain assumptions* control the amount of structure to be reasoned about. We introduce a simple notion of system for controlling the granularity of an analysis. *Perspective assumptions* control the point of view taken on a system. These include the choice of ontology, approximation, and abstraction. The relationships between these assumptions can be complex, so we adapt the notion of *assumption classes* from [ACP89] to allow domain-specific coherence constraints to inform model composition. We describe a new model composition algorithm which uses these representational extensions.

We also demonstrate that these ideas can be applied to quantitative as well as qualitative models. While our algorithm can always provide a relevant model, it cannot guarentee sufficient accuracy *a priori*. We show how the use of explicit modeling assumptions can sometimes allow the detection of inaccurate models and suggest appropriate revisions.

1.2 Overview of compositional modeling

A *domain model* describes a class of related phenomena or systems. It consists of a set of *fragments*, each describing some fundamental piece of the domain's physics, such as processes (e.g., liquid flow), devices (e.g., transistor), or 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 fragments from the domain model. This modularity is the heart of compositional modeling: implicit in the domain model is a vast set of consistent scenario models, which can be assembled as needed rather than explicitly enumerated in advance. Automatic model composition requires explicit representation of modeling assumptions. Each fragment must include sufficient conditions for its applicability. The language of modeling assumptions provides the "connective tissue" for organizing large-scale domain models.

We divide the process of modeling a specific scenario into three stages: (1) composing the simplest coherent model sufficient for the task, (2) performing the task using the model, and (3) evaluating the results to ensure they are reasonable. This paper focuses on (1), with a foray into (3) for the special case of modeling assumption violations uncovered during the analysis phase.

1.3 Domain model organization

We begin by outlining how to organize domain models using the compositional modeling strategy. We focus on *simplifying assumptions*, which provide the bulk of control over the applicability of model fragments[1]. We draw on examples from an implemented domain model of the thermodynamic phenomena in steam propulsion plants.

1.3.1 Simplifying assumptions

The groundwork of any particular analysis is a set of simplifying assumptions specifying which aspects of a domain are relevant. For uniformity, we stipluate that all simplifying assumptions take the form

CONSIDER((*AsnType*)((*system*)))

[1] *Operating assumptions*, which constrain potential behaviors (e.g., steady-state) are also important, but are not discussed further here. See [FAFO88, FAFO90].

where ⟨*AsnType*⟩ is a predicate denoting the specific kind of assumption and ⟨*system*⟩ is what the assumption is about. We distinguish three kinds of simplifying assumptions, described below.

Grain assumptions Tractable analysis of large systems requires tightly focusing on what is relevant to answer specific questions. A rich model of a ship's laundry, for instance, provides no direct insight into boiler efficiency. Often collections of objects can be considered as a single, aggregate entity, such as ignoring the internal structure of the furnace when analyzing the global behavior of a propulsion plant. *Grain assumptions* control what objects are considered in an analysis.

We require all objects in scenarios to be organized into *systems*. A system is either a primitive object or a named collection of 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. Currently we require systems to form a strict hierarchy. The root of this hierarchy, which contains all the objects in the scenario, is always a system called `:scenario`.

Grain assumptions are stated using the `existence` predicate. When

```
CONSIDER(existence((system)))
```

holds, it indicates that a model for ⟨*system*⟩ must be included in the current analysis. Thus including

```
CONSIDER(existence(boiler))
```

in the framework of an analysis forces the boiler itself to be modeled, rather than treating the boiler assembly as a black box or focusing on the boiler's subsystem (e.g., steam tubes, economizer, etc.)

The notion of system provides critical constraint on grain assumptions. Intuitively, one cannot simply pick an arbitrary subset of a system to model. Enough parts must be included to ensure that all relevant relationships involving the objects of interest are included. Considering an automobile transmission and wheels in isolation, for instance, will miss important interactions between them unless the drive shaft and differential are also taken into account.

We define a *covering system* to be any system that contains all systems of interest. (Clearly `:scenario` is always a covering system.) A *minimal covering system* is the lowest common ancestor in the part-of hierarchy of the systems being considered. The idea of minimal covering system provides the means to enforce the intuition above. Suppose we have two objects of interest, and both are part of the same system. Then all of that system's components must be considered. (We presume that if some parts of the system could be further isolated, this fact would be reflected in the system hierarchy.) If the two objects are at different levels of the system hierarchy, the minimal covering system will be the smallest system which has components which include both objects, and hence instantiating its parts will ensure that the relevant structural connections between them will be included. We return to this in Section 1.4.2.

Ontology assumptions Different tasks demand carving the world up differently. For instance, fluids can be modeled as *contained stuffs* [FORB84], an Eulerian perspective, or as *molecular collections* [COFO87], a Lagrangian perspective. Other ontological assumptions include focusing on energy or on mechanics. Ontological assumptions are specified by domain-specific predicates, e.g., the following indicates the relevance of contained-stuffs:

```
CONSIDER(fluid-cs(:scenario))
```

Ontology assumptions work as follows. (1) All ontological assumptions are global. That is, they always apply to `:scenario` and are inherited by all subsystems. (2) At least one ontological assumption must be included in every analysis. (3) Multiple ontological assumptions are allowed when consistent. For instance, some questions require combining results from energy flow and mass flow analyses.

Approximations and Abstractions *Approximations* provide simpler models at the cost of reduced accuracy. Examples include incompressible fluids, inviscid flows, inelastic objects, and frictionless motion. Approximations are stated via explicit predicates, such as `viscous`. *Abstractions* reduce the complexity of a model without reducing accuracy, but at the cost of diminished detail and increased ambiguity. Examples include modeling a fluid valve as a discrete switch versus a continually varying conductance. Both approximations and abstractions differ from ontological assumptions in that they typically do not represent sufficient

viewpoints by themselves.

Constraints on approximation and perspective assumptions are obviously domain-specific. For example, in our models it does not make sense to consider the space that connects a fluid path to a container as an explicit portal unless the geometric properties of the container are included in the analysis.

Assumption classes As seen above, some assumptions represent mutually exclusive alternatives for modelling some aspect of an object or phenomena. We use *assumption classes* to represent this important relationship. Assumption classes are declared as

```
(defAssumptionClass ⟨class-form⟩ (a-forms))
```

where ⟨*class-form*⟩ indicates when the set of choices is relevant and (*a-forms*) is an ordered list of alternatives. We call an assumption class *active* when ⟨*class-form*⟩ holds, and in this case exactly one of ⟨*a-forms*⟩ must be included in any scenario model. For example, our domain includes two models of viscosity:

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

We use the ordering of ⟨*a-forms*⟩ to provide a simple model of cost: Models specified by assumptions earlier in the list are presumed to be cheaper, in some sense, than later models. (For instance, the `viscous` model occurs second in the specification above because including fluid resistance is often an unnecessary complication.) While an oversimplification, this simple model of costs is surprisingly useful (see Section 1.4.4).

Conditions of applicability and constraints on using different models can be stated independently, thus enhancing modularity. For instance, in our domain the need to model viscosity is declared by

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

which indicates that it is relevant when fluid flow is being considered. Some of these constraints are subtle: For instance, questions concerning head loss should rule out the `inviscid` (frictionless) assumption.

1.3.2 Specifying model fragments

We use a generalization of the modeling language developed for QP theory [FORB84]. The basic form is `defModel`, whose syntax is:

```
(defModel ⟨name-form⟩
    Individuals ⟨i-spec⟩
    Assumptions ⟨asns⟩
    OperatingConditions ⟨opcon⟩
    Relations ⟨rels⟩ )
```

where ⟨*name-form*⟩ provides a pattern for specifying instantiations of the model. The `Individuals` field describes the physical settings to which the model applies, and ⟨*i-spec*⟩ defines all the model's (logical) variables. ⟨*asns*⟩ contains its simplifying assumptions, and ⟨*opcon*⟩ are the operating assumptions under which it holds. ⟨*rels*⟩ describe the consequences of the model, including qualitative and quantitative equations[2]

A model is *applicable* for a set of objects and constants if they satisfy ⟨*i-spec*⟩. If the simplifying assumptions also 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 plant, while a model of boiling is applicable but need not be applied. Notice that a model can apply to a scenario yet not constrain it. For example, a model of liquid flow is applicable to a path even if it is currently blocked. An instance of a model fragment is *active* exactly when it is both applicable and its `OperatingConditions` hold.

1.4 Model Composition

Our task is to construct a scenario model that is sufficient to answer a given question with minimal effort. Sufficiency has two aspects. The first is *aboutness*, i.e., ensuring the scenario model includes all the aspects of the system required to perform the analysis. The second is *accuracy*, i.e., ensuring the scenario model contains enough detail to provide a suitable answer. In this paper we focus on aboutness criterion[3] Accuracy and minimal effort often conflict. We resolve this dilemma by using minimality as a filter on sufficient models.

Our model composition algorithm operates under two restrictions. First, we require that the information needed to derive an appropriate

[2] QP fans will note that this syntax can be easily transformed into standard QP notation. `Preconditions` and `QuantityConditions` become `OperatingConditions`, `Assumptions` are pulled from the information in the standard `Individuals` field, and a model is considered a view or a process according to whether or not ⟨*rels*⟩ includes direct influences.

[3] [SHFA90] focuses on accuracy.

scenario model can be gleaned solely from the query and the contents of domain model. Any domain-specific conventions or default patterns of communication must be handled via preprocessing of the query. Second, we only address the problem of selecting appropriate simplifying assumptions. Although some operating assumptions are entailed by choices of simplifying assumptions, the general problem of choosing operating assumptions is beyond the scope of this paper.

The idea underlying our algorithm is this. While a scenario model consists of a collection of instantiated model fragments, directly searching the space of model fragments is too expensive. Instead, we find what model fragments are applicable, and organize our search for scenario models in the space of simplifying assumptions they suggest. This is more efficient because there are generally fewer modeling assumptions than model fragments. Thus we produce as output a consistent set of ground modeling assumptions which, together with the given scenario description, entail a minimal, sufficient scenario model.

For simplicity, we describe our algorithm in terms of assumption-based truth maintenance systems (ATMS) [DEKL86]. Each statement in the database has a corresponding ATMS *node*. Some nodes are marked as *assumptions*. An *environment* is a set 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 $\mathcal{E}$, then the corresponding statement is believed under the assumptions which comprise $\mathcal{E}$.

All ground simplifying assumptions are marked as ATMS assumptions. We use the labels and dependencies maintained by the ATMS to compute ***modeling environments***, consisting of conjunctions of modeling assumptions which entail scenario models. Given a scenario description, we begin by finding instantiations of the domain model's fragments. Once this is done, model composition consists of four steps: *query analysis*, *object expansion*, *candidate completion*, and *candidate evaluation and selection*. We describe each in turn.

1.4.1 Query analysis

We presume a *query elaboration procedure* which can decompose a query into a set of ground expressions $Q = \{e_1, \ldots, e_n\}$, where each e_i has a referent in the applicable model fragments. This procedure must be domain and task specific. However, one useful general technique is to

focus on extracting quantities from a query, which is typically easy and highly indicative of what objects must exist for the analysis to make sense.

Given $\mathcal{Q}$, 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 $\mathcal{L}_{\mathcal{Q}}$ is a seed, since at least one of these environments must be included in the eventual scenario model in order for the statements in $\mathcal{Q}$ to hold.

Suppose for instance our query is "How does the furnace's fuel/air ratio affect the amount of steam flowing in the superheater?". By translating it into the quantities of interest, we get

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

Figure 1.1 shows the dependencies for the corresponding `QUERY` node, and in this case the ATMS computes the single seed:

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

Note that this seed does not entail a coherent scenario model. For example, it assumes the existence of the furnace and the superheater, yet fails to include the boiler which connects them. The next two steps grow seeds into full modeling environments.

1.4.2 Object expansion

The first step in ensuring coherence is to include any relevant objects beyond those directly entailed by the seed. In tracing the properties of steam flowing through the boiler and turbine, for instance, we need to consider the condenser and feed pump as well to correctly recognize that its flow is part of a closed cycle. New objects to include are selected based on the minimal covering system for the objects entailed by the seed. If the seed entails the existence of only a single object, that object is the system and no new assumptions are added. Otherwise, the seed is extended to include grain assumptions about any part of the minimal covering system not already entailed by it. In the example above, the minimal covering system for the `furnace` and `superheater` is the `boiler-assembly`, which forces

```
Consider(existence(boiler)
```

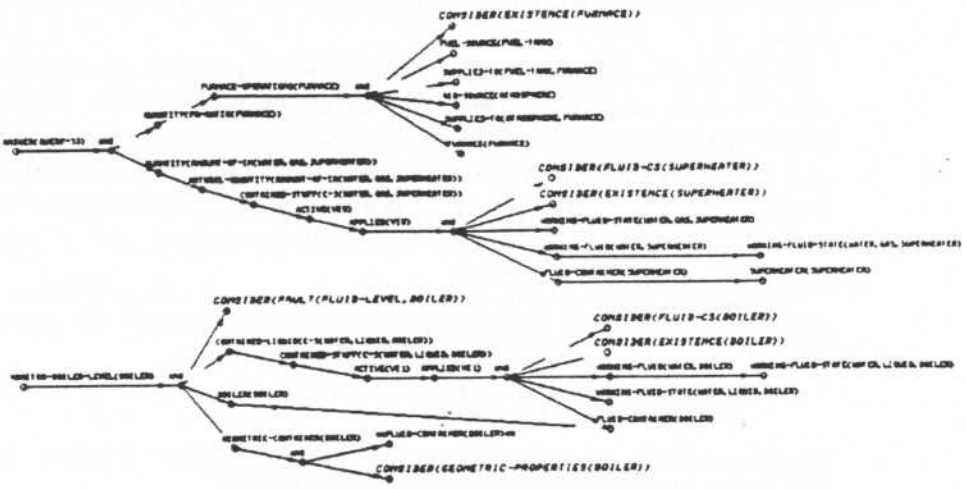

Figure 1.1
This illustrates the relationship between model terms and the modeling assumptions that support them. `Answer(query-33)` represents the conjunction of the query expressions. `Monitor-boiler-level(boiler)` shows what would have been needed to also include the boiler's fault model and why it was deemed unnecessary for the query.

to be added to our seed (see Figure 1.2). We continue to ignore other components (e.g., the turbine and condenser assemblies) and subsystem details (e.g.,the furnace's fuel pump and exhaust manifold).

1.4.3 Candidate completion

Although our seeds now entail every relevant object, they do not necessarily prescribe exactly how each object should be modelled. This step uses assumption classes to add the necessary approximation and perspective models to complete the seeds. Roughly, it works like this. Each seed entails some set of assumption classes. Extended seeds are generated by finding all consistent combinations of choices from these active assumption classes. The actual procedure is slightly complicated because (a) choices from different assumption classes can interact, by one either implying or contradicting another, and (b) some choices make other assumption classes active in turn. We use the *dynamic constraint*

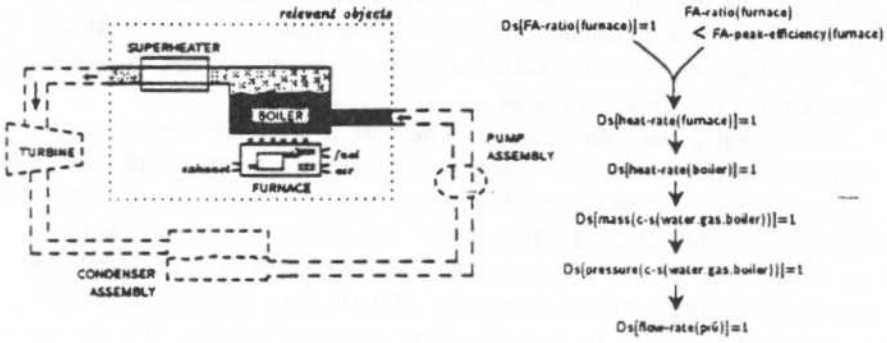

Figure 1.2
How does the furnace's fuel/air ratio affect the steam flowing in the superheater?

satisfaction procedure described in [MIFA90]. Assuming the domain model is correctly formed, i.e., that it includes assumption classes and constraints between them to completely describe the relevant aspects which need to be included in modeling each kind of object, then the result of this step is a set of full modeling environments.

In the example above, the seed is now

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

There are nine instances of assumption classes active for this seed, drawn from three distinct types. The first concerns whether or not a container's geometric properties are modeled (i.e., the furnace, boiler, and superheater). The second concerns whether thermal properties need to be considered for fluids in these containers. The third type concerns the appropriate model for fluid resistance in each flow path.

For efficiency, we do not exhaustively enumerate all full modeling environments. Instead, we do a best-first search of the set of full modeling environments, using the evaluation metric described next.

1.4.4 Candidate evaluation and selection

The final step is to select "the best" modeling environment. This of course depends on the details of the domain and task, but we have found the following procedure useful. Essentially, we use several general constraints to induce a preference ordering on modeling environments,

and begin by selecting the best. (Given equally prefered models, one is chosen at random.) Should this model prove unsuitable, we backtrack to select the next best alternative, subject to the constraints uncovered in analyzing the previous model.

We currently use two criteria. The first is to prefer environments which entail fewer objects. After all, at this stage each candidate is presumed sufficient to answer the query, and minimizing objects tends to minimize the total size of a model and hence the inferential work involved in using it. The second criteria uses the ordering in assumption classes to estimate overall simplicity. Recall that the choices in each class are presumed to be ordered in increasing complexity. We use the position of each choice in the list as a score, and compute the score for an environment as the sum of the choices made from each assumption class it entails. Lower scores are prefered over higher ones, since they involve simpler choices (or at least, fewer perspectives).

Suppose a set of model fragments contained 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.

This scheme has obvious limitations. For example, it is unlikely that $\{A_2, B_2\}$ and $\{A_3, B_1\}$ should be considered equivalent. But we leave more sophisticated schemes for future work.

1.5 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 [FORB89], an envisioner for QP theory. For example, the envisionment generated for the furnace/superheater question describes how the furnace's fuel/air ratio can affect the amount of steam flowing in the superheater. Figure 1.2 shows the result of a perturbation when the fuel/air ratio is increased if the furnace has been operating suboptimally, with a low fuel/air ratio. Briefly, an increase in fuel/air ratio 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 fourth-order Runge-Kutta integration algorithm with adaptive step-size control (from [PFTV86, ch. 15]). The equations are gathered from the scenario model, with some minor processing to get them into the form it expects. Currently we restrict numerical simulations to a single operating region – that is, all equations hold all of the time.

In both qualitative and quantitative problems, routines from QPE are used to find the applicable model fragments, and to generate a scenario model once an acceptable set of modeling assumptions has been found. Then the appropriate simulation method is invoked, based on the preprocessing of the query.

Verifying the results of an analysis is always a crucial problem in modeling. Sometimes *a priori* guarentees can be made for specific cases, but since generally some parameters are unknown before analysis, the initial scenario model may rest on invalid assumptions about the system's behavior. Models can be verified in many ways, including observation or comparison with other models. Here we focus on internal consistency tests, since these tend to be cheaper. For instance, identifying an internally inconsistent qualitative model is easy: If the envisionment is empty, the model must be inconsistent because there must be some state for the physical system to be in. See [FAFQ88] for examples of qualitative analyses using compositional modeling.

Determining if predicted behavior violates initial assumptions is an important aspect of verifying numerical simulations. We do this by gathering *critical inequalities*, those required by the model's simplifying assumptions. For example, to assume incompressible flow requires that its Mach number (The ratio of its velocity to the speed of sound) be less than 0.3. If the Mach number ever exceeds 0.3 in a simulation based on this assumption, the modeling environment is deemed inconsistent. When this happens we backtrack and repeat model composition, informed by the inconsistency. The next section illustrates.

1.6 Example: Identifying appropriate flow models

Consider two oil supply drums connected to a central reservoir (Figure 1.3). The problem is to determine the behavior of the oil levels when

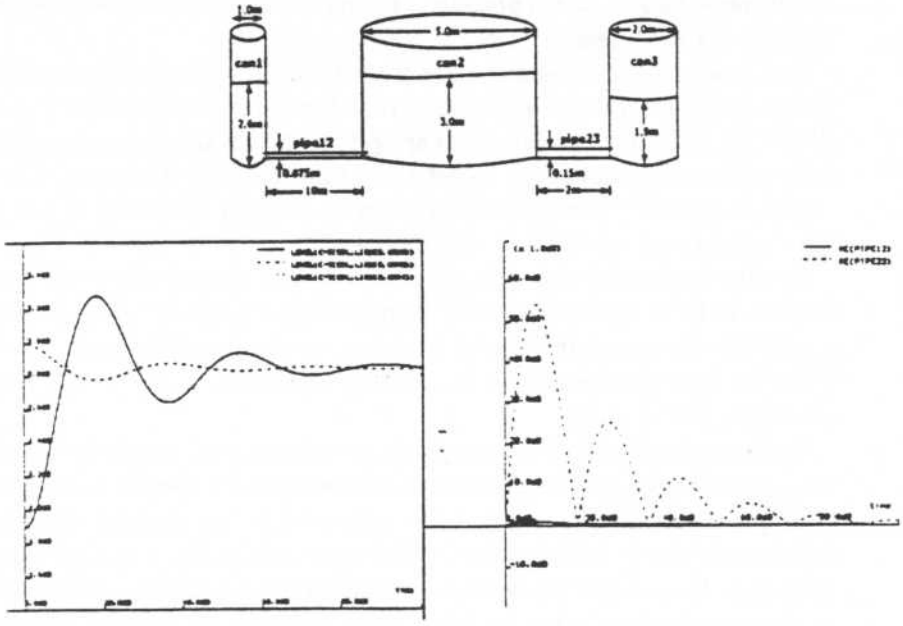

Figure 1.3
Two oil supply drums connected to a central reservoir. Find the level of the three containers as a function of time when the system is released from the given initial conditions.

the system is released from the illustrated initial condition. Our domain model includes the unsteady Bernoulli equation, e.g.,

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

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 depends 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 algorithm cannot determine the Reynold's number for either pipe. Thus, it selects the simplest modeling

environment, which 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 1.3. The system inspects the predicted behavior and finds a modeling violation: the Reynold's 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.

Notice that the laminar flow assumption for pipe12 remains consistent, since its Reynold's number never exceeded 1,100. This shows an important feature of compositional modeling. By representing modeling assumptions as predications over individuals, a scenario model can use different models for the same type of object or phenomena as appropriate. This flexibility is crucial for analyzing large systems. Sometimes in analyzing a circuit, for example, certain wires must be considered as transmission lines – even though it would be computationally disasterous to consider all wires thusly.

1.7 Related Work

The closest work is the *graph of models* (GoM) approach [ACP89], in which the space of possible scenario models is represented explicitly as a graph. Each node represents a scenario model, and each edge indicates which assumptions differ between the models it connects. To the extent scenario models can be pre-enumerated GoM will be faster, but at a worst-case exponential increase in storage. We believe the compositional model approach, with its emphasis on automatic scenario modeling, has more potential for organizing large-scale knowledge bases.

An interesting aspect of GoM is the ability to reason about how changing assumptions affects model/observation discrepancies. Weld

[WELD89] describes a perturbation technique which might be applied to compositional models, generating "adjacent" alternatives by repeating our candidate completion procedure.

Some aspects of the use of grain assumptions to control search were captured by Davis [DAVI84]. The equivalent of his strategy in compositional modeling would be to automate movement through a set of grain assumptions during an analysis.

1.8 Discussion

Automatic modeling of physical systems is one of the long-term goals of qualitative physics. We believe that compositional modeling provides an important step towards that goal, by providing ways to organize large-scale, multi-grain, multi-perspective models of physical domains. We view our model composition algorithm's ability to insulate the analyst from the details of the domain model as particularly important. For tutoring tasks it is a necessity: If the student knew enough to build the appropriate model, the tutor would be unnecessary. But we believe even expert engineers will benefit from allowing the model composition algorithm to share the burden of finding (and verifying) the right model. Efficiency will be gained if an engineer can specify just enough to make her intent clear, leaving the obvious to a (mechanical) assistant. Consistency will be gained if routine and rigorous tests are made to ensure simplying assumptions made by engineers working on the same artifact in different places and times do not conflict.

1.9 Acknowledgements

We thank John Collins, Mark Shirley, Sanjay Mittal, and Johan deKleer for productive discussions. 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.

Bibliography

[ACP89] 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.

[COFO87] 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.

[DAVI84] Davis, R. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24, 1984.

[DEKL86] deKleer, J. An assumption-based TMS. *Artificial Intelligence*, 28(2), March 1986.

[FAFO88] 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.

[FAFO90] Falkenhainer, B. and Forbus, K. D. Compositional Modeling: Finding the right model for the job. To appear, *Artificial Intelligence*, Fall, 1991.

[FORB84] Forbus, K. D. Qualitative process theory. *Artificial Intelligence*, 24, 1984.

[FORB89] 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.

[MIFA90] Mittal, S and Falkenhainer, B. Dynamic constraint satisfaction problems. Proceedings of AAAI-90, Boston, August, 1990.

[PFTV86] Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. *Numerical Recipes*. Cambridge University Press, 1986.

[SHFA90] Shirley, M. and Falkenhainer, B. Explicit Reasoning About Accuracy for Approximating Physical Systems, *Working Notes of the Automatic Generation of Approximations and Abstractions Workshop*, July, 1990.

[WELD89] 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.