Qualitative Models for Space System Engineering Final report June, 1990 Contract NASA NAG 9137

P-75

Kenneth D. Forbus
Qualitative Reasoning Group
Department of Computer Science
University of Illinois
405 N. Mathews Avenue
Urbana, Illinois, 61801

Internet: Forbus@cs.uiuc.edu
Voice: (217) 333-0193

1 Introduction

The objectives of this project were:

1. To investigate the implications of qualitative modeling techniques for problems arising in the monitoring, diagnosis, and design of Space Station subsystems and procedures.

2. To identify the issues involved in using qualitative models to enhance and automate engineering functions. These issues include representing operational criteria, fault models, alternate ontologies, and modeling continuous signals at a functional level of description.

3. To develop a prototype collection of qualitative models for fluid and thermal systems commonly found in Space Station subsystems.

We have accomplished most of these objectives. This report summarizes the research carried out under this project, pointing off to technical reports and publications produced by the project to supply details. This report also includes several attachements which will be produced as technical reports after appropriate review by NASA personnel.

Secton 2 begins by summarizing potential applications of qualitative modeling to space-systems engineering, including the notion of *intelligent computer-aided engineering*, and focusing on which systems of the proposed Space Station provide the most leverage for study, given the current state of the art. Section 3 summarizes our progress on developing a new methodology, *compositional modeling*, for organizing large-scale qualitative domain models, and summarizes our prototype domain model for engineering thermodynamics. Section 4 summarizes our progress on using qualitative models, including our development of the *molecular collection* ontology for reasoning about fluids, the interaction of qualitative and quantitative knowledge in analyzing thermodynamic cycles, and an experiment on building a natural language interface to qualitative reasoniner. Finally, Section 5 makes some recommendations for future research.

2 Potential space-systems engineering applications

As engineering projects grow more complex, automation becomes more important. In design, choices by individuals and teams can interact in subtle ways, leading to expensive iterations as constraint violations are detected, or

in-service failures if problems remain undetected. In monitoring, as systems grow more complex operators need more help in maintaining their understanding of what the system is doing, what kinds of events might happen, and how the system can be re-configured to continue safe operations in the event of failures.

Space-systems engineering shares these problems with other branches of engineering, but includes several other problems as well. First, less is known about the environment space systems operate in. For instance, the behavior of two-phase fluid systems in zero-g conditions is still a topic of investigation, and these investigations will impact the design of thermal control systems. Second, supporting human beings in space is very expensive, hence the need for automation is more pressing. And third, variability in funding climate can lead to long project delays and high personnel turnover, leading in turn to a loss of design and operational expertise.

Explicit representation of knowledge and computational accounts of how knowledge is used by engineers seems crucial to improving the degree of automation and support for complex engineering projects. Many areas of Artificial Intelligence are important for this. Particularly important is *qualitative physics*, which focuses on representing and reasoning about continuous physical systems. This project has explored how qualitative physics might be used to support space-system engineering, and pushed the state of the art closer to being useful in this context.

2.1 Intelligent Computer-Aided Engineering

Expert systems are only the beginning of how AI could be useful for engineering. Today's expert systems tend to be narrow, in that they contain knowledge about a specific physical system, specialized for a specific reasoning task. It is hard to characterize what fraction of their task they actually can perform, since both knowledge acquisition and testing tends to be based on a library of specific cases, rather than a systematic theory of the task and the domain. It is easy to claim that if we "just added more knowledge" the systems would be better. But what kinds of knowledge, and how much of it, do we need to achieve the flexibility of human engineers?

While human engineers often specialize on particular kinds of systems and/or tasks, they know far more than just their specialty. Furthermore, their knowledge appears to be anchored on a groundwork of commonsense knowledge of the physical world. Encoding such knowledge in a way usable by computers, therefore, seems to be a key step in making programs which can provide a new level of assistance to human engineers and operators. The

idea of *intelligent computer-aided engineering* is that a program's knowledge should include the same breadth and types of knowledge that a human engineer has. For instance, it should include *domain models* which encode knowledge about a wide variety of fundamental phenomena independent of specific physical systems and specific reasoning tasks. Some speculations on the nature of intelligent computer-aided engineering are contained in [7].

2.2 High-leverage space-systems engineering applications

There are two aspects to an application: What is being reasoned about, and what kind of reasoning is being performed. To select something to reason about, we examined design documents for several subsystems of the proposed Space Station. Since our research focus is Qualitative Process theory [6], we focused on designs for the Thermal Control System (TCS). The basic task we have used is prediction, that is, deriving the set of possible behaviors the system can undergo. Specifically, we compute *envisionments*, which contain every possible dynamical behavior of the system, for each logically consistent initial state. Envisionments are interesting in themselves, but can also be used for planning, measurement interpretation, and a variety of other tasks. Section 4 describes our investigations of specific reasoning tasks. [7] outlines applications which should be possible when the state of the art has advanced farther.

3 Progress in qualitative modeling

Much of our energy has been focused on developing new, richer qualitative models of engineering thermodynamics. In doing so, we have been forced to develop a new methodology for developing domain models. We summarize each in turn.

3.1 Compositional Modeling strategy

No engineer applies everything she knows in an analysis. In figuring out the maximum thermal load of a cooling system, for instance, one's knowledge of quantum mechanics is irrelevant. Furthermore, one even ignores the potential ways the cooling system can fail, by assuming it is behaving normally. (To be sure, a good designer may later perform similar analyses for various potential failure conditions.) Making appropriate *modeling assumptions* is,

we claim, crucial to engineering reasoning. Our *compositional modeling* strategy for organizing domain models provides a formal way of making and using simplifying assumptions and operating assumptions to control the creation and use of qualitative and quantitative models.

The first description of the compositional modeling strategy appeared in [4]. A more complete description, which includes a better account of model composition and demonstrates that the same techniques can be applied to quantitative models, is included in [5].

3.2 The FSThermo domain model

We have completed the first version of a large-scale model of fundamental phenomena in engineering thermodynamics. While the model has many limitations, we believe it covers the domain better than any previous model. Furthermore, it makes heavy use of the compositional modeling strategy to provide several levels of detail in modeling objects and systems. The model itself and the design decisions underlying it are described in [2].

4 Progress in qualitative reasoning

In addition to developing new domain models, we have explored several new ways to use qualitative models.

4.1 Reasoning about fluids using molecular collections

Most qualitative models have described fluids in terms of the containers they are housed in. This *contained stuff* ontology is useful for many purposes, including figuring out what kinds of physical processes are occurring in a system. However, this ontology is useless for other kinds of questions. It does not make sense, for instance, to talk about fluid moving through a system from this perspective – the "liquid ammonia in an evaporator", for instance, is always in the evaporator, by definition. An alternate ontology is needed to explain and analyze thermodynamic cycles. We developed the *molecular collection* ontology for this purpose. The idea is to think about a little piece of stuff ("MC") roaming through a system defined so that (a) it is large enough to have macroscopic properties such as temperature and pressure and (b) so small that it never splits up. The processes computed using the contained-stuff ontology suffice to figure out both how MC can move through a system and how its properties will change as it travels from

place to place. This description can then be used for other tasks, such as recognizing that a system is operating as a heat engine.

The first publication describing the molecular collection ontology was [1]. An expanded description of this ontology, including a set of formal definitions and laws and detailed algorithms, is included in [3].

4.2 Solving textbook thermodynamics problems

One of the purposes of qualitative knowledge is to provide a framework using other kinds of engineering knowledge. Many engineering tasks require using quantitative models, often using symbolic algebra or numerical simulation to come up with numerical estimates or analytic approximations. We have begun exploring how qualitative and quantitative knowledge should interact in engineering thermodynamics, using textbook problems as examples. This work has been described in [8].

4.3 Providing natural language interfaces

One of the motivations of qualitative physics is that human mental models of systems seem to include qualitative aspects. Hence a system which used qualitative physics should provide a better "impeadance match" for human engineers. While this project did not explore interface issues much, we did experiment with a natural language interface to our envisioner [9].

5 Discussion

We believe qualitative modeling is an extremely promising technology for space-systems engineering. Furthermore, we believe that this project (1) has made substantial progress towards understanding exactly how qualitative modeling could be applied to space-systems engineering and (2) has pushed the state of the art closer to being applicable. Even so, much basic research remains to be performed before qualitative modeling will be a technology ready for widespread application.

First, the state of the art in qualitative modeling is still fairly primitive. Some open problems include:

1. *Scaling in structure:* We cannot envision extremely large structural descriptions (such as the blueprints of the Boeing TCS test article used in the TEXSYS project). Techniques for representing and performing the mapping from structural descriptions, such as blueprints or pictures

of systems, to structural abstractions, which isolate only the aspects of the object critical to a particular set of tasks, need to be developed.

2. *Integration:* Our current attempts to integrate quantitative models have been fragmentary and fairly crude. Integrating multiple analytic approximations of the same phenomena, and controlling when each is used, is an extremely important problem. Integration of information across multiple ontologies (e.g., contained stuffs and molecular collections) also needs work.

3. *Ontological investigations:* Ways of individuating more complex spatial entities are needed in order to produce qualitative analyses of systems currently analyzed through finite-element analysis. Multiple-substance mixtures and chemical effects need to be modeled, to capture phenomena involved in corrosion and reasoning about process plants. Even in the sub-domain of analyzing thermodynamic cycles, it seems that an ontology "between" the molecular collection and contained stuff ontolgies is needed to capture changes which occur over a single spatial dimension.

Second, qualitative reasoning itself involves many open problems:

1. *Incremental techniques:* Envisioning is exponential, and hence intractable for large-scale systems. We need to develop techniques for exploring only relevant subsets of behaviors.

2. *Analytic qualitative reasoning:* There is more to quantitative analyses than numerical simulation: When possible, traditional, symbolic algebra analyses can yield far more insight. We need to explore similar options for qualitative analysis.

3. *Integrated analyses:* Our current textbook problem solver carries out its qualitative reasoning first, and then performs the quantitative analysis. While the quantitative analysis uses the framework of the qualitative analysis and consults it constantly, the mathematical results never force a re-analysis of the system in qualitative terms. Yet this is exactly what happens when an engineer discovers that an incorrect approximation was used, or that the system in fact cannot work.

Progress needs to be made on these issues, and others, before qualitative modeling will achieve its full potential. Today qualitative modeling is being tested in a variety of applications, and feedback from these attempts should also prove useful in refining the research agenda of qualitative physics.

References

[1] Collins, J. and Forbus, K. "Reasoning about fluids via molecular collections", Proceedings of AAAI-87, July, 1987.

[2] Collins, J. and Forbus, K. "Building qualitative models of thermodynamic processes", Technical Report in preperation. (See attachment # 1).

[3] Collins, J. and Forbus, K. "Molecular Collections: An ontology for reasoning about fluids", submitted for publication, February, 1990. (See attachment # 3).

[4] Falkenhainer, B. and Forbus, K. "Setting up large-scale qualitative models" Proceedings of AAAI-88, August, 1988.

[5] Falkenhainer, B. and Forbus, K., "Compositional Modeling: Finding the right model for the job", submitted for publication, February, 1990. (See attachment # 2)

[6] Forbus, K. "Qualitative process theory" *Artificial Intelligence*, **24**, 1984

[7] Forbus, K. "Intelligent Computer-aided Engineering", AI Magazine, Fall, 1988.

[8] Skorstad, G. and Forbus, K. "Qualitative and quantitative reasoning about thermodynamics", Proceedings of the Cognitive Science Society, Ann Arbor, Michigan, August, 1989.

[9] White, D. "QATT: A natural-language interface for QPE. University of Illinois Department of Computer Science Technical Report No. UIUCDCS-R-89-1535, August, 1989.

6 Attachment # 1

Building Qualitative Models of Thermodynamic Processes

John W. Collins
Kenneth D. Forbus
Qualitative Reasoning Group
Beckman Institute, University of Illinois
405 North Mathews St.
Urbana, IL 61801

Abstract

This paper describes a qualitative domain theory for core phenomena in engineering thermodynamics, expressed in Qualitative Process theory. It represents many of the best features of domain models developed by our group over the past five years. It focuses on supporting system-level qualitative analyses of typical fluid and thermal systems, such as refrigerators and power plants. We use explicit modeling assumptions [3] to control the level of detail used in building models of specific scenarios. We begin by outlining the primitives of the specific QP modeling language. The bulk of the paper describes the domain model itself, highlighting our design choices, simplifications, and use of modeling assumptions. Next we demonstrate how this domain model can be used to build models of a variety of specific scenarios, including simplified versions of a refrigerator, a steam plant, and a thermal control system. Finally, we describe some planned extensions to the model.

Contents

List of Figures

1 Introduction

This paper develops a qualitative domain model for thermodynamic and fluid systems, based on Qualitative Process theory. This model incorporates many of the best features of the domain models our group has been developing over the last five years. Several domain models for such systems have been described previously [5,3], but have various limitations.

The FSThermo domain model exhibits three important features:

- *Broad Coverage:* Previous models (e.g., [5]) covered only a small subset of relevant phenomena. The FSThermo model captures a broader spectrum of phenomena. For example, it defines richer models for a variety of physical processes, including fluid flows (liquid or gas, forced or free), heat flows, and phase transitions between the liquid and gaseous phases.

- *Fine Grain:* The FSThermo model provides more detailed perspectives of several phenomena, such as the role of portals in fluid systems and latent heat in boiling, than previous qualitative models.

- *Modeling Assumptions:* The domain model of [3] demonstrated that modeling assumptions could be used to organize abstract, system-specific models. Here we use the same methodology to control a fine-grained model, showing that by varying the granularity appropriately, a quite intricate qualitative domain theory can still be efficiently used to answer questions.

Furthermore, this is the first detailed description of the design choices underlying a substantial domain model. We have tried to be explicit about our reasons for various design choices, and where our simplifying assumptions impact the model, for good or ill. While this is not a tutorial for QP modeling, we hope it will be useful to other qualitative modelers. We also show how the FSThermo model can be used to model a variety of systems, including a steam plant, refrigerator, and Thermal Control System.

Section 2 begins by outlining some issues involved in building domain models. Next, Section 3 describes the modeling language we use. Specifically, the domain model is written in the language of QPE[8], an envisioner for Qualitative Process theory. We assume a reading knowledge of QP theory: This section only describes some of the implementation-specific properties of this modeling language that are important in understanding how the domain model is used. Section 4 describes the FSThermo model itself. We begin with basic object and structural descriptions and examine how flows are modeled. We describe phase changes and pumps next. Finally, we examine the interrelationships between the various modeling assumptions and the encoding and importance of the steady-state assumption. Section 5 shows a variety of systems modeled using FSThermo. We show how the same structural description can lead to a variety of models, according to what simplifying assumptions are in force, and analyze the consequences for the complexity of qualitative simulation. We also see how models of larger (although still abstract) systems can be successfully simulated in minutes yielding a handful of states, rather than days and thousands of states, if performing reasonable analyses. Finally, Section 6 outlines what we have learned

by building this model, and makes suggestions concerning future domain models, modeling languages, and qualitative reasoners.

2 Modeling Issues

An important feature of Qualitative Process theory is that it makes more of the modeling process explicit. That is, knowledge of the physical world is organized as a *domain model*, which describes the basic conceptual entities and phenomena. Given a particular physical situation, constructs of the domain model are combined to form a *scenario model* of the specific situation.

Component-centered ontologies [2,16] are also organized in this way, but subject to the following restrictions. First, it is assumed that all primitive phenomena can always be associated with a single, explicit component. Second, the interconnections between components are in terms of shared quantities only, rather than introducing new objects. Finally, the process of mapping from a structural description to elements of the component library is assumed to be straightforward (or at least left outside the scope of existing theories). While these restrictions work reasonably well for electronics, they do not work very well for most engineering domains (e.g., thermodynamics), and quite poorly for many important domains (e.g., motion).

A process-centered ontology is more apt for thermodynamics and fluid systems. Many thermodynamic phenomena are typically conceptualized as processes. Furthermore, fluid systems have non-trivial node capacities, so the approximation represented by Kirchoff's Current Law is often inappropriate. The mapping from a structural description to conceptual entities is also more complex in fluid and thermal problems. For example, in some problems the geometry of containers is important, and in others it is not. (This is actually true for electronics as well, ouside the usual (implicit) assumptions of low-frequency signals.) The need for multiple levels of granularity cannot be ignored in engineering thermodynamics problems.

QP theory also provides an additional source of leverage, beyond its ability to express process-centered models. It provides ways to encode explicit *modeling assumptions*, so that the problem of building a model for a specific scenario from a domain model becomes a subject for explicit reasoning by the QP interpreter. Developing a domain model that is capable of covering a wide variety of fluid and thermodynamic phenomena requires careful consideration of several issues:

Composability Anticipating every potential scenario is impossible. Instead, the constructs of the domain model are *composable.* That is, complex systems and behaviors can be described by applying and combining the results of many simple, local descriptions. Furthermore, we attempt to minimize the number of primitive constructs. It would be a mistake, for example, to encode the activity in the normal, steady-state operating mode of a steam plant as a single process. This model would apply in very few situations, and common phenomena with similar systems would remain implicit. Instead, we limit our-

selves to describing only fundamental physical processes in the domain model. Of course, an engineer's stock of knowledge includes detailed information about specific scenarios and classes of systems (e.g., two–phase refrigeration systems). We exclude such entries from this domain model. By covering the basic physical phenomena, we hope to provide the constructs needed to ground these more specific models.

Level of Detail A primary modeling decision concerns choosing the appropriate level of detail. An early step in developing a model for some domain is to partition the domain up into discrete objects. The coarseness of the partitioning determines the coarseness (and efficiency) of the reasoning. For example, reasoning at the level of contained-liquids would be too coarse if our goal were to understand sloshing.

The appropriate level of detail depends on the goals of the modeler. For instance, the desired level of performance—whether expert or novice—greatly influences modeling choices. Likewise, a model for only examining nominal operations will look very different from a model designed to anticipate possible failure modes.

Modeling Idealizations Every finite model only considers those aspects of objects and their behaviors deemed relevant by the model–builder. *Modeling idealizations* ignore aspects of the model which are either (a) insignificantly small in magnitude, duration, or likelihood; (b) outside of the intended functionality for some component; or (c) qualitatively uninteresting.

Often we are interested in modeling the long–term or steady–state aspects of a thermodynamic system, and so choose to ignore transient behaviors. For example, our model for fluid flow ignores the acceleration of the fluid in the path, in favor of an equilibrium model which relates flow rate and pressures directly.

A quantity which never changes might be viewed as qualitatively uninteresting. For example, the *conductance* (or *resistance*) of a fluid path is generally constant, and can be excluded from the model by defining the flow rate as the qualitative difference in pressures across the path. However, having conductance provides a hook for adding a continuous model for valves, and avoids the direct comparison of quantities of different units (eg. flow-rate and pressure).

Modeling Assumptions As models develop, many choices must be made between distinct perspectives on phenomena and different levels of detail. When multiple alternatives look useful, one might split the model into seperate pieces. But as the number of options grows, the number of distinct models can rise exponentially. By organizing domain models around modeling assumptions, conflicting models can peacefully co-exist.

Here modeling assumptions typically take the form `(Consider ?X)`, where `?X` represents some aspect or dimension which is or is not being included in the scenario model under construction.

Modularity To manage complexity, the domain model is partitioned into a set of relatively independent modules. For example, *heat flow* is sufficiently different from other processes in thermodynamics to be considered a separate module. No module is totally independent from the others; heat flows involve physical objects, as do all other processes. In general each module depends on a set of lower modules, and may be used by still higher modules.

As a matter of pragmatics, each module is stored in a separate file. This allows an evolving model to be compiled incrementally; in addition, only those modules required for a particular scenario need be loaded.

Not surprisingly, hierarchical representation is useful in qualitative physics. Hierarchies are used extensively in representing physical entities; for example, a contained–liquid is a contained–stuff, which is a physob. Quantities and other properties are inherited from the general class to the specific instance. Hierarchical representations have also been applied to processes, though to a lesser extent. For example, there is much in common between liquid flow and gas flow. Consequently, we have defined a common, abstract `fluid-flow` process to contain their intersection, and ancillary perspectives which represent phase-specific details. No new, special syntax is introduced to handle hierarchies – we simply use logical implication and the binding abilities of normal QP descriptions.

3 An Overview of the QPE Modeling Language

Our representations are encoded in QP theory. The syntax is that used by the modeling language associated with a particular program which implements QP theory, called QPE. Given a domain model, a structural description, and a collection (possibly empty) of modeling assumptions, QPE constructs a model of that scenario based on the constructs of the domain model, and produces a total envisionment of it. The details of how QPE works are described in [8]. This modeling language is quite close to the syntax used in the original QP papers, but has the advantage that it is executable. Almost no special properties of this modeling language are essential to understanding the domain model, but we point out any interactions below.

3.1 Defining objects, properties, and relationships

The form `Defquantity-type` introduces a new kind of quantity. The first argument is the name of the type of quantity. Each quantity type is considered a function, and the rest of the arguments are the arguments of that function. Each argument is declared as either an individual or a constant. This information is used in computing whether or not a quantity exists in a particular situation. That is, if any of the individuals it is about do not exist, then that quantity does not exist. For example, if we were describing the temperature of the arsenic in a cup of coffee, and there was no arsenic, then it would not even make sense to talk about its temperature.

An example of `DefQuantity-Type` is

```
(defQuantity-Type distance individual individual)
```

which allows us to describe distances between two entities, such as

```
(greater-than (A (distance Urbana Chicago)) (A (distance Evanston Chicago)))
```

QPE's vocabulary now includes `defPredicate`, which may be used to specify consequences of a single antecedent predicate. The first argument to `defpredicate` is the predicate whose consequences are being defined. The rest is the body, which constitutes a set of consequences which should be believed when the predicate is believed. When the predicate is a single symbol, then it is implicitly a unary predicate, with the variable `?self` bound to the object of the predicate. `defEntity` is similar, but also implies existence of its object.

For example, we might define some of the economic aspects of a person by

```
(defEntity Person
 (Quantity (income ?self))
 (Quantity (net-worth ?self)))
```

which indicates that when a person exists, they have some income and net worth. (To be less dismal we might constrain these quantities to be non-negative.) Then to define someone as solvent for some purpose, we might say

```
(defPredicate (Solvent-For ?person ?purpose)
  (greater-than (A (net-worth ?person)) (A (cost-of ?purpose))))
```

that is, their net worth is more than the cost of the thing they want to do.

3.2 Qualitative Mathematics

The standard modeling primitives of QP theory are available, albeit in a lisp-style syntax. That is, where in theoretical papers one might see

$$Q_1 \propto_{Q+} Q_2$$
$$Q_1 \propto_{Q-} Q_3$$

we will write

```
(Qprop+ Q1 Q2)
(Qprop- Q1 Q3)
```

Other primitives are translated to lisp-style syntax in the obvious fashion. Several new primitives are special versions of existing ones which exploit computational savings available for special cases. For instance, an `Ordered-Correspondence` is a form of `Correspondence` which assumes a positive qualitative proportionality; this permits QPE to use a simpler set of internal justifications to enforce its semantics. Similarly, `*0+` and `/0+` are special versions of multiplication and division which assume that their arguments are non-negative.

There are two other important things to note about the algebraic primitives used in QPE. First, qualitative proportionalities and direct influences have a causal interpretation as well as a mathematical one. That is,

```
(Qprop+ (temperature ?obj) (heat ?obj))
```

indicates that a change in heat (i.e., internal energy) causes a change in temperature, as well as indicating that when the heat rises the temperature will, all else being equal. In the case of direct influences, the process in which the I+ or I- appears is causing a change in the first parameter, at a rate specified by the second parameter. Thus if the only direct influence on (heat ?obj) was (flow-rate ?heat-flow) and imposed by an instance of heat flow, that is,

```
(I+ (heat ?obj) (flow-rate ?heat-flow))
```

it would indicate both that the instance of the heat flow process was the cause of any change in (heat ?obj)[1], and that

```
(D (heat ?obj)) = (A (flow-rate ?heat-flow))
```

The second point is that the semantics of +, -, *, and / are defined in terms of qualitative proportionalities and correspondences.[2] Thus they inherit the causal interpretation of the qualitative proportionalities they expand into. Thus the expression

```
(Q= (Temperature ?self) (/0+ (heat ?self) (mass ?self)))
```

indicates that temperature causally depends on heat and mass, as well as indicating the mathematical nature of the relationship.

There is an additional subtlety concerning direct influences. If the quantity being influenced does not exist, the direct influence has no effect. This stipulation greatly simplifies defining processes which behave properly when their effects cause objects to come into existence. Otherwise, one often needs to double the number of constructs for certain phenomena, to handle the instant in which a process acts before the stuff it produces appears.

3.3 Defining views and processes

The basic syntax of views and processes is a lispified version of the normal QP syntax. For example, we might define a budget with a surplus as

```
(defview (surplus ?gov)
 Individuals ((?gov :type government))
 QuantityConditions ((greater-than (A (resources ?gov)) zero))
 Relations ((Probability (during-election-year) High)))
```

That is, the relationship surplus happens to things which are governments, when their resources are greater than zero, and the direct consequence of a surplus is that it is probably an election year.

Each entry in the individuals field contains a variable and some restrictions on what it can be bound to. The syntax and meaning of the restrictions are explained below. By convention, each entry is thought of as defining a *role* for each instance of that view (or process), hence one can speak of the gov of an instance of surplus as a function mapping from view instances to the individual filling that role.

Processes are specified similarly:

[1] We haven't specified the sign of (flow-rate ?heat-flow) here, remember, so we don't know for a fact that there is a change.

[2] For products and ratios, these must be conditioned on the signs of the appropriate multipliers/divisors.

```
(defprocess (Taxation ?sap ?gov)
 Individuals ((?gov :type government)
              (?sap :type person
                    :conditions (honest ?sap)))
 Relations ((quantity taxes)
            (greater-than (A taxes) zero)
            (Qprop- taxes (resources ?gov)))
 Influences ((I+ (resources ?gov) (A taxes))
             (I- (net-worth ?sap) (A taxes))))
```

Notice that some additional syntax has been added to the `individuals` field to facilitate more expressive pattern-matching. In particular, it is stipulated that entries in the `individuals` field are matched sequentially, in order of appearance. The following keywords are supported:

`:type` Indicates that the next token is a unary predicate which must hold for an instance to exist.

`:form` Indicates that the variable can only be bound to expressions which match the pattern which follows.

`:bind` Indicates that the variable is to be bound to the form which follows. The form must contain variables, all of which are bound by earlier entries in the `individuals` field.

`:test` Indicates that the next form is a lisp expression which must be non-nil for an instance to be created with the bindings so far.

`:conditions` Indicates that all the remaining forms in the entry are additional statements which must hold for an instance to be created. Obviously, `:conditions` must be the last keyword in any entry.

One should think of `:form`, `:bind`, and `:test` as extra controls on the instantiation of views and processes, while `:type` and `:conditions` provide the antecedents which justify creation of an instance. That is, the instance of a view or process exists exactly when the union of any statements generated by the bindings of the `:type` and `:conditions` modifiers hold. Notice that if any of these statements is known to be false such an instance can never exist, let alone be active. The implementation is guaranteed to respect this constraint by never creating instances of views or processes if one of these antecedents is known to be false at creation time[3]. This stipulation is what allows us to control the level of detail when instantiating scenario models.

3.4 Defining perspectives

Sometimes it is useful to exploit the pattern-matching machinery introduced above to define new predicates which do not have quantity conditions (and hence do not contribute

[3] A common bug in domain models is that sometimes the falseness of some antecedent isn't discovered until after the instance is created. The record of instances of processes and views are never erased, even though their existence is carefully predicated on the appropriate antecedents.

new constituents of state, see [8]). In particular, these relationships are often predicated on modeling assumptions, using the `:conditions` keyword. Owing to their role in defining domain models, we call such rules *perspectives*, and define them via `defPerspective`. A `DefPerspective` is interpreted the same was a `defView` is, except that it is forbidden to have quantity conditions.

4 A Tour of the Core Thermodynamics Model

This section examines the FSThermo domain model in detail. We begin by outlining the class of problems which motivated it, to make the underlying simplifications clearer. Then we start with various kinds of physical objects, move on to processes, and end by describing the simplifying assumptions and operating assumptions used to structure the domain and analyses thereof.

We need to distinguish concepts in engineering thermodynamics from our formal renderings of them. Concepts in engineering thermodynamics will be described in normal type face, using English or mathematical formulae as appropriate. For example, the pressure of water in some can c is typically written as P_c in thermodynamics texts. Our formal renderings of them will be put in typewriter font:

```
(Pressure (C-S water liquid can))
```

4.1 The Organization of the Model

How does one build a domain model for a set of physical phenomena? The first thing to think about is the kind of phenomena you are trying model. In thermodynamics, this consists of various flows and energy transformations. It requires models of fluid flow, heat flow, work flow, phase changes, and if one is describing the outputs of certain systems, motion. These physical processes are of course modeled as processes in QP theory. When we know what kinds of processes are involved, we next have to think about the sorts of objects they occur between, and what properties of those objects and their interrelationships allow those physical processes to occur. This gives us the framework upon which to hang the constructs of our model.

The degree to which one wants to decompose objects depends on what phenomena you need to be able to reason about independently. For example, if you discover you want to think about heat flow independently from mass flow, it becomes important to decompose a physical object into its thermal and non-thermal aspects. In fact, deriving the complete set of processes in advance can be difficult, and we find ourselves alternating between thinking about processes and thinking about objects many times in constructing a domain model.

By formalizing the objects and the conditions under which processes can occur, we have made our ontological commitment. In this ontological framework, we can then figure out the qualitative proportionalities and direct influences which capture the corresponding equations governing them. In this way, the compositionality of QP primitives allows the construction of the appropriate set of qualitative equations for any specific scenario, given that one identifies the appropriate physical objects with their formal equivalents.

Any model highlights some aspects of reality and ignores others. It is crucial when developing a domain theory to be clear about what phenomena one does not intend to capture. We have tried to make our simplifying assumptions reflect those found in normal engineering thermodynamic analyses. For instance, we obviously ignore quantum effects and the possibility of relativistic motion and other exotic physics.

Engineering thermodyanmics is concerned with the understanding of systems such as power plants, engines, refrigerators, and other energy conversion devices. Our goal is to provide the qualitative and ontological framework for the sorts of analyses found in a first year engineering thermodynamics course. Roughly, this means analyzing systems made of abstract fluid components, rather than detailed analyses of the properties of specific components. Thus we restrict ourselves to circumstances where we can ignore details of geometry. This restriction is implicit in many engineering thermodynamics textbooks. However, it does rule out some phenenomena which engineers learn in their schooling. For instance, the FSThermo model is not concerned with how fluid properties change through nozzles or across blades in turbines. It does not capture the effects of scaling on heat transfer across surfaces. It also ignores the detailed dynamics of fluids. In particular, it ignores any inertial effects of fluid flow, the distinction between turbulent and laminar flow, and any effects of water hammer. We suspect that at least some of these phenomena could be added with few changes to this model.

For further simplification, the FSThermo model ignores the effects of chemical interactions. In fact, we limit it to single-substance systems, although we make no particular assumptions about what the working substance is. We believe that adding chemical interactions will require some, but not substantial, modifications to the existing model, in addition to defining new processes associated with such interactions.

4.2 Types of Objects

Our model includes six basic kinds of concrete objects: *physobs*, *containers*, *contained stuffs*, *paths*, *pumps*, and *compressors*. We describe each in turn.

4.2.1 Physical Objects

It is useful to extract a common core of physical properties that most concrete objects must have. This common core notion is called *physob*. There are several kinds of physobs, each corresponding to a different coherent bundle of object properties, to control granularity and perspective. For example, in modeling a pure hydraulics system one typically ignores thermal properties of the working fluid. Similarly, if we are considering an abstract heat flow problem, we can ignore any hydraulic aspects of a part.

We use physob to refer to the most basic description. Various specializations of physob are defined to represent specific combinations of properties. Figure 1 introduces the continuous properties used with different types of physobs. `Mass`, `Volume`, `Pressure` and `Temperature` represent their usual thermodynamic properties. We use `heat` for internal

Figure 1: Defining quantities associated with physob

```
;; Extensive properties
(defQuantity-Type Mass Individual)
(defQuantity-Type Heat Individual)
(defQuantity-Type Volume Individual)
;; Intensive properties
(defQuantity-Type Pressure Individual Individual)
(defQuantity-Type Temperature Individual Individual)
(defpredicate non-negative-quantity
  (quantity ?self)
  (not (less-than (A ?self) ZERO)))
(defpredicate positive-quantity
  (quantity ?self)
  (greater-than (A ?self) ZERO))
```

energy out of respect for the intuitive language often still found in modern thermodynamic textbooks.

The extensive properties (i.e., `Mass`, `Heat`, and `Volume`) belong to specific individuals. The intensive properties (i.e., `Pressure` and `Temperature`) are point properties, and hence involve a comparison with respect to some frame of reference. We have chosen to make this comparison explicit in this model. We thus avoid introducing new types of quantities to represent ΔP's and ΔT's, at the cost of always naming an explicit comparison point. The token `:ABSOLUTE` is considered to be an abstract individual which always exists and indicates that the comparison is with the appropriate ground or absolute zero value for that type of quantity[4].

Figure 1 also defines predicates for sign constraints. Such constraints abound in thermodynamics texts, and they are just as crucial in qualitative reasoning. Two specializations of `Quantity` are defined using `defPredicate`: `Positive-Quantity` ensures that it is always larger than zero, and `Non-Negative-Quantity` ensures that it is never less than zero.

The actual definitions of physobs is contained in Figure 2. The first `defentity` is the basic notion of physob, which serves as a uniform basis for matching. Thermal properties are captured via `Simple-Thermal-Physob` and `Thermal-Physob`. A `Simple-Thermal-Physob` has `Temperature`, and a `Thermal-Physob` is a `Simple-Thermal-Physob` with `Heat`. (The reason for the distinction will become clear in Section 4.3.1.) The temperature of a `Thermal-Physob` is qualitatively proportional to its heat. Thus if the heat of a `Thermal-Physob` is influenced (up or down), then its temperature is *indirectly influenced* in the same direction.

A `Volumetric-Physob` has mass and volume. While we know it has these properties, until we know its phase we cannot say anything about how they are related. A `Complex-Physob` is both a `Thermal-Physob` and a `Volumetric-Physob`. Having both

[4]Notice that zero pressure here is zero in absolute pressure rather than gauge pressure.

Figure 2: Defining quantities associated with physob

```
;;; Basic types
(defentity Physob) ;; root type -- good for matching
(defentity Simple-Thermal-Physob
  (Physob ?self)   ;; Presume Kelvin scale, and forbid absolute zero.
  (Positive-Quantity (Temperature ?self :ABSOLUTE)))
(defentity Thermal-Physob
  (Simple-Thermal-Physob ?self)
  (Non-Negative-Quantity (Heat ?self))
  (Qprop+ (temperature ?self :ABSOLUTE) (heat ?self))
  (Consider (Thermal-Properties ?self)))
(defPerspective (Thermal-Physob ?phob)
  Individuals ((?phob :type physob
                      :conditions (Consider (Thermal-Properties ?phob)))))
(defentity Volumetric-Physob
  (Physob ?self)   ;; Forbid negative masses, pressures, and volumes
  (Non-Negative-Quantity (Mass ?self))
  (Non-Negative-Quantity (Volume ?self))
  (Non-Negative-Quantity (Pressure ?self :ABSOLUTE))
  (Consider (Volumetric-Properties ?self)))
(defPerspective (Volumetric-Physob ?phob)
  Individuals ((?phob :type physob
                      :conditions (Consider (Volumetric-Properties ?phob)))))
(defentity Complex-Physob
  (Thermal-Physob ?self)
  (Volumetric-Physob ?self)
  ;; With thermal and volumetric properties temperature can
  ;; be defined in the usual way:
  (Q= (Temperature ?self :ABSOLUTE) (/Q+ (heat ?self) (mass ?self))))
(defPerspective (Complex-Physob ?phob)
  Individuals ((?phob :type physob
                      :conditions (Consider (Volumetric-Properties ?phob))
                                  (Consider (Thermal-Properties ?phob)))))
```

aspects allows us to define the relationship between temperature, heat and mass. In particular, the temperature is the quotient of the heat and the mass.

The rest of the statements in Figure 2 enforce the semantics of modeling assumptions. Notice the `Consider` statements in the consequences of the `Thermal-Physob` and `Volumetric-Physob` definitions. These statements enforce the consistent use of modeling assumptions. That is, if it is assumed that `FOO` is a `Thermal-Physob`, then it must be the case that one is considering the thermal properties of `FOO`. Attempting to also assume that one should ignore thermal properties globally, or just of that specific object, is thus inconsistent, and any self-respecting QP interpreter should detect this contradiction.

The two `defPerspective` definitions associated with `Thermal-Physob`, `Volumetric-Physob`, and `Complex-Physob` play a similar role. For maximum flexibility, an object can be described as the most minimal physob consistent with its nature (in most cases Physob, but for contained stuffs, `Volumetric-Physob` is minimal) and modeling assumptions used to control which additional aspects of its nature should be included in some analysis.

Figure 3: Definition for Container

```
(defentity Container
  (Physob ?self)
  (Positive-Quantity (volume ?self))
  (Non-Negative-Quantity (volume (liquid-in ?self)))
  (Non-Negative-Quantity (pressure (gas-in ?self) :ABSOLUTE))
  (Non-Negative-Quantity (pressure (bottom ?self) :ABSOLUTE))
  (Qprop+ (pressure (bottom ?self) :ABSOLUTE) (pressure (gas-in ?self) :ABSOLUTE)))
```

These perspectives provide this service by supporting the appropriate predication if the corresponding antecedents hold.

This combination of perspectives and consider assumptions appears repeatedly in the domain model, so we will not dwell on it when it appears again. At first glance it might appear that the use of `Consider` assumptions in `defEntity` descriptions is a violation of modularity. After all, we are placing what is essentially control information into a description of a physical object. But this is actually an important feature. The whole purpose of developing a qualitative language for physical modeling is to be able to encode information in ways that allows it to be used in reasoning. A language which did not capture modeling assumptions must perforce leave them implicit, and thus will fail to take on some of the burden that a qualitative physics must.

4.2.2 Containers

Most thermodynamic systems involve fluids existing inside some kind of container. Examples of objects modeled by containers are evaporators, boilers, and tanks. We are using the *contained stuff* ontology for fluids [10,5], so containers play a central role in defining stuffs.

Containers are defined as specializations of `physob`. Since volume is a key property of containers, it is tempting to model containers as `volumetric-physobs`. However, for the problems we are considering containers remain in fixed positions. This means we can ignore their mass, and hence the `volumetric-physob` description contains excess committments. Instead, we declare the container to have `volume` explicitly.

It is worth dwelling on this choice a bit further, since it illustrates an important principle n building domain models. We are not assuming that genuine physical containers per se lo not have `mass`. Instead, when we view an object as a container, we are only interested n those aspects which are relevant to its capacity to contain fluids. If we wish to reason bout moving a pot of water to the stove, we must view the pot both as a container and a moveable object, which makes its `mass` relevant. Similarly, if we wanted to model ntainers melting, we could describe the container as a `thermal-physob` in addition to scribing it as a container. This composability is one of the powerful aspects of the ysics.

There are two other crucial choices to be made when modeling containers. One is whether or not containers are open or closed. This intuitive distinction rests on whether or not a container is exposed to the atmosphere. Since we can always model an open container by including an explicit fluid path to an entity representing the atmosphere, we assume all containers are closed.

The other choice is how detailed should container geometry be modeled. The detailed three-dimensional shape of containers is irrelevant for the level analyses we are considering. Essentially, the most detail we need are heights and volumes. However, often we don't even require this much detail. The cost of including container geometry is the introduction of additional quantities representing geometric properties and additional comparisons between them to express geometric relationships. For some kinds of systems, such as siphons or devices where gravity head is used to produce flow, this cost is unavoidable. But geometric considerations can be ignored for many systems, including most pump-driven ones. Consequently we include the modeling assumptions `Geometric-Properties` to control whether or not such details are introduced.

Figure 3 shows the basic definition of `Container`. We assume volumes are always positive: zero-volume "nodes" are not allowed. The main property to represent is pressure, which is important because it determines when material flows are possible. Physically, the pressure in a container depends both on what is in it and where it is measured. If it is filled with a gas the pressure will be uniform throughout, for example, and if it has both liquid and gas in it, the pressure at a point will vary with the depth of the liquid covering it. Expressing these relationships can be quite complex, since they depend on exactly what exists in a container. When something doesn't exist, neither do its properties[5]. When the amount of a contained stuff shrinks to zero it vanishes, and hence its properties vanish as well. Maintaining physically correct relationships over such changes in existence can be a daunting task.

Our solution to this problem is to introduce two new abstract individuals: the *liquid in the container* and the *gas in the container*, denoted by the functions `liquid-in` and `gas-in`, respectively. These abstract individuals always exist, whether or not there is any liquid or any gas in the container. When stuff of the appropriate phase exists, these abstract individuals take on their properties. Otherwise, their properties are constrained to produce physically reasonable results. In particular, we define the volume of the `liquid-in`, since it determines the volume available for any contained gas. Similarly, we define the pressure of the `gas-in`, because it contributes to the pressure of a liquid.

If portals are used, each portal can have a pressure. If portals are too detailed, we need some standard measuring point to talk about the pressure of a liquid. We choose the bottom of the container, allowing us to presume that no matter how little liquid there is, it will always be in contact with the bottom. (How the connectivity is inferred when portals are explicit is described in Section 4.2.4.) We assume the function `bottom` maps a container to the lowest point of the container's inside. We note the dependence of the bottom pressure on the pressure of the `gas-in` explicitly with a qualitative proportionality.

[5]Can one speak seriously of the temperature of the arsenic in the coffee one is drinking and continue drinking it?

Figure 4: Definition for Geometric-Container

```
(defQuantity-Type Height Individual)
(defQuantity-Type Level Individual)
(defentity Geometric-Container
  (Container ?self)
  (Consider (Geometric-Properties ?self))
  (Quantity (height (bottom ?self)))
  (Quantity (height (top ?self)))
  (greater-than (A (height (top ?self))) (A (height (bottom ?self))))
  (Quantity (level (liquid-in ?self))) ; Portals use this
  (Qprop+ (level (liquid-in ?self)) (volume (liquid-in ?self)))
  (Ordered-Correspondence ((A (level (liquid-in ?self))) (A (height (bottom ?self))))
                          ((A (volume (liquid-in ?self))) ZERO))
  (Ordered-Correspondence ((A (level (liquid-in ?self))) (A (height (top ?self))))
                          ((A (volume (liquid-in ?self))) (A (volume ?self))))
  (Qprop+ (pressure (bottom ?self) :ABSOLUTE) (level (liquid-in ?self)))
  (Ordered-Correspondence ((A (pressure (bottom ?self) :ABSOLUTE)) (A (pressure (gas-in ?self) :ABSOLUTE)))
                          ((A (level (liquid-in ?self))) (A (height (bottom ?self))))))
(defPerspective (Geometric-Container ?can)
  Individuals ((?can :type container
                     :conditions (Consider (Geometric-Properties ?can)))))
```

Any further information about the relationship between these two parameters depends on additional information about exactly what stuffs are in the container.

Container geometry Figure 4 shows the `Geometric-Container` extension of the basic container model. Two new geometric properties, `height` and `level`, are introduced. `height` corresponds to vertical distance along some presumed global reference frame. `level` corresponds to the vertical position of liquid within a container, again within this same global frame. Since we are assuming containers have fixed positions, we leave this reference frame implicit rather than including a second argument, as we did with `temperature` and `pressure`.

We introduced the `bottom` of a container previously. The function `top` maps from a container to the lowest point of the container's top opening, if it has one; otherwise it represents the highest point inside the container. We continue to ignore the thickness of container walls. Both the `top` and `bottom` of a container have an associated `height`. We assume in this model that containers are sitting in their "normal" position, i.e., the `height` of the `top` is `greater than` the `height` of the `bottom`.

The level of liquid (should it exist) determines what touches a portal (see Section 4.2.4). Hence we introduce `level` of the `liquid-in`, and constrain it to be a function of the volume of the `liquid-in`. Furthermore, we make the pressure at the bottom a function of the level of the `liquid-in`. We associate two limit points with the level, the `height`s of the `bottom` and `top` of the container, to represent three important facts (via the `Ordered-Correspondence` statements). First, the level is at the bottom when the volume of the `liquid-in` is zero. This covers the case of no liquid in the container. Second, the

level is at the top when the volume of the `liquid-in` is the same as the volume of the container. This helps define fullness and sets the conditions for overflows. Finally, we constrain the pressure at the bottom to be the pressure of the `gas-in` when the level is at the bottom, e.g., when no liquid is present.

4.2.3 Contained Stuffs

A contained stuff is defined by the substance it is, the phase it is in, and the container which holds it. A contained stuff is denoted by the function `C-S`:

$$\texttt{C-S} : \textit{substance} \times \textit{phase} \times \textit{container} \longrightarrow \textit{contained stuffs}$$

For example, `C-S(water,gas,boiler)` refers to the contained stuff which is made of water in the gaseous phase inside the boiler, or more simply, "the steam in the boiler".

The amount of stuff of a particular substance in a particular phase within a particular container can vary over time. When there is a non-zero amount of it we say the corresponding contained stuff exists, and when the amount is zero the contained stuff does not exist. Clearly, negative amounts of stuff are impossible.

In addition to representing these basic intuitions, we must also represent – and decompose – our knowledge about particular kinds of stuffs. Often analyses only concern a single phase: gasses are ignored when analyzing a hydraulic system, for instance, and liquids are ignored when analyzing an air-cycle refrigerator. We may choose to ignore many kinds of substances: We all know about plutonium, but rarely do we think much about "the lump of plutonium in the bottom of my coffee cup". We may wish to consider material sources and sinks, and hence ignore the possibility that containers can become empty or overflow. As usual, we begin with the basic intuitions of contained stuffs, and add layers of models to represent the ramifications of different modeling assumptions.

Figure 5 defines the basic notions of contained stuffs. Formally, we treat substances and phases as constants. The model does not include quantitative data or other properties which distinguish one substance from another, so water, ammonia, and alcohol are all alike. (This is sensible under our assumption that only a single substance is under consideration at any time.) Phase can be either `liquid` or `gas`. The choice of phase, of course, has important consequences.

Intuitively, `amount-of-in` should be thought of as the number of molecules of a given substance and phase in that particular container. Two things should be noticed here. First, we cannot make this a property of the contained stuff itself, since the property must exist even when the object doesn't in order to be that which defines the object's existence. Second, notice that containers are treated as full-fledged individuals, and hence potentially have finite temporal extent. While nothing in the current model provides for the creation or destruction of containers, it is easy to imagine augmenting the vocabulary with actions which do so. Such changes will be required for detailed modeling of melt-downs and explosions, for instance, as well as a more detailed model of the surroundings.

The `Stuff-In-Container` perspective sets up `amount-of-in` for each combination of substance, phase, and container and constrains it to be non-negative. It also helps enforce

Figure 5: Definition for Contained-Stuff

```
(defQuantity-Type amount-of-in Constant Constant Individual)
(defperspective (stuff-in-container ?s ?st ?c)
   Individuals ((?s  :type Substance)
                (?st :type Phase
                     :conditions (Consider ?st))
                (?c  :type Container))
   Relations ((Non-Negative-Quantity (Amount-of-in ?s ?st ?c))
              (when (not (Can-Contain-Substance ?c ?s ?st))
                (equal-to (A (Amount-of-in ?s ?st ?c)) ZERO))
              (when (Can-Contain-Substance ?c ?s ?st)
                (when (not (Consider (Empty-Container ?c)))
                  (greater-than (A (Amount-of-in ?s ?st ?c)) ZERO)))
              (when (Consider Capable-Containers)
                (Can-Contain-Substance ?c ?s ?st))))
(defview (Contained-Stuff ?cs)
  Individuals ((?can :type container)
               (?sub :type substance)
               (?st :type phase
                    :conditions (Consider ?st)(Consider Changing-Existence))
               (?cs :bind (C-S ?sub ?st ?can)))
  Preconditions ((Can-Contain-Substance ?can ?sub ?st))
  QuantityConditions ((greater-than (A (amount-of-in ?sub ?st ?can)) ZERO))
  Relations ((there-is-unique ?cs)))
(defperspective (Contained-Stuff ?cs)
  Individuals ((?can :type container)
               (?sub :type substance)
               (?st :type phase
                    :conditions (Consider ?st) (Can-Contain-Substance ?can ?sub ?st)
                    (not (Consider Changing-Existence)))
               (?cs :bind (C-S ?sub ?st ?can))))
(defentity (contained-stuff (C-S ?sub ?st ?can))
  (Volumetric-Physob (C-S ?sub ?st ?can))
  (Q= (mass (C-S ?sub ?st ?can)) (amount-of-in ?sub ?st ?can)))
(defentity (Contained-Stuff (C-S ?sub liquid ?can))
  (Contained-Liquid (C-S ?sub liquid ?can))
  (Q= (volume (liquid-in ?can)) (volume (C-S ?sub liquid ?can))))
(defentity (Contained-Stuff (C-S ?sub gas ?can))
  (Contained-Gas (C-S ?sub gas ?can)))
```

various properties and modeling assumptions about stuffs. First, we may know that a container cannot contain certain kinds of stuffs (e.g., nitric acid in a copper beaker or sulphuric acid in a paper cup). Such facts are indicated by the appropriate instance of `Can-Contain-Substance` being false, and this perspective pins the `amount-of-in` in these cases to be zero. Second, if we want to assume that a container is never empty, then we constrain the `amount-of-in` to be positive. Finally, the assumption of `Capable-Containers` is tantamount to assuming that every container can contain every substance in any phase, which is enforced by justifying `Can-Contain-Substance` for each combination. (This assumption is used to simplify the specification of inital conditions in scenario models. If it is false, the scenario modeler must have some external theory which introduces the appropriate instances of `Can-Contain-Substance`, or do so by hand.)

The `Contained-Stuff` view defines existence if we are allowing contained stuffs to have finite temporal extent (as evidenced by the dependence on the `Changing-Existence`

Figure 6: Definition of Contained-Liquid

```
(defentity Contained-Liquid
  (Qprop+ (volume ?self) (mass ?self))
  (Ordered-Correspondence ((A (volume ?self)) ZERO)
                          ((A (mass ?self)) ZERO)))
(defPerspective (Contained-Liquid-Geometry ?cl)
  Individuals ((?can :type Geometric-Container
                     :conditions (Consider Gravity) (Consider (Geometric-Properties ?can)))
               (?cl  :type Contained-Liquid
                     :form (C-S ?sub liquid ?can)))
  Relations ((Quantity (level ?cl))
             (not (less-than (A (level ?cl)) (A (height (bottom ?can)))))
             (Qprop+ (level ?cl) (volume (liquid-in ?can)))
             (Q= (level (liquid-in ?can)) (level ?cl))))  ;;; Portals use this
(defperspective (Aspatial-Contained-Liquid ?cl)
  Individuals ((?can :type Container
                     :conditions (Consider Gravity)
                     (not (Consider (Geometric-Properties ?can))))
               (?cl :type Contained-Liquid
                    :form (C-S ?sub liquid ?can)))
  Relations ((Qprop+ (pressure (bottom ?can) :ABSOLUTE) (volume (liquid-in ?can)))
             (Ordered-Correspondence ((A (pressure (bottom ?can) :ABSOLUTE))
                                      (A (pressure (gas-in ?can) :ABSOLUTE)))
                                     ((A (volume ?cl)) ZERO))))
```

modeling assumption). Recall that the `there-is-unique` predicate ensures that when the containing form is false, its argument is also false, thus enforcing the biconditional nature of the existence conditions under these assumptions. Importantly, recall that if `Changing-Existence` is false, no instances of this view will ever be created, hence this restriction will not be in force. In that case, the next `defPerspective` ensures that all possible stuffs exist, subject to container capabilities.

The core of contained stuffs is expressed in the next three `defentity` forms. We require all contained stuffs to be `volumetric-physobs`, regardless of phase, to ensure that they have mass, volume, and pressure. Furthermore, we constrain the mass to be the value of the `amount-of-in`, to reflect the fact that the mass will vary as the amount of stuff does. In essense, this `Q=` links the underlying molecular conception to the macroscopic construct of mass.

The second `defentity` specializes contained stuffs to be contained liquids (note the constant `liquid` in the second argument position for the `C-S` in the pattern). It also pins the volume of the `liquid-in` to in fact be the volume of the contained liquid. (In a multi-substance model, the volume of the `liquid-in` would have to be the sum of the volumes of the set of contained liquids in the container.) The third `defentity` plays a similar role for contained gasses.

Contained liquids Figure 6 illustrates the model of contained liquids. The `defentity` provides the geometry-independent properties, namely that the volume is qualitatively proportional to the mass, and is zero when the mass is. (In a more detailed model –

Figure 7: Novel environmental conditions can be handled compositionally

```
(defperspective (Zero-Gravity-Contained-Liquid ?cl)
  Individuals ((?can :type Container
                     :conditions (not (Consider Gravity)))
               (?cl  :type Contained-Liquid
                     :form (C-S ?sub liquid ?can)))
  Relations ((Q= (pressure ?cl :ABSOLUTE) (pressure (gas-in ?can) :ABSOLUTE))))
```

especially if multiple substances are included – the additional dependence on density should be noted as well.) The first perspective defines the additional properties which hold when geometry is considered. In particular, the contained liquid has a level, which is never lower than the bottom of the can and depends on the volume of liquid. (We have made level depend on the volume of the `liquid-in` rather than directly on the volume of the contained liquid for upward compatibility with future, multiple-substance models.) Furthermore, the level of the `liquid-in` is exactly this level. The second perspective ties the pressure at the can's bottom to the volume of the `liquid-in`, to provide an appropriate constraint when geometry is being ignored.

Figure 7 shows how compositional modeling can be used to deal with a wide range of special conditions. To model fluid and thermal systems for space systems engineering, one must be able to control whether or not gravity is considered as a factor. At the level of detail of our current model, this assumption has two impacts. First, even if geometry is considered, it becomes meaningless to talk about levels. Second, the pressure of a contained liquid no longer depends directly on the amount of liquid present. Instead, it is determined by the pressure of any gas present (which depends in part on the volume available, and hence on the volume of the liquid, and therefore indirectly on the amount of liquid present). The `Zero-Gravity-Contained-Liquid` perspective encodes this model.

Contained gasses Many thermodynamic analyses involve gasses. Modeling gasses introduces several new factors. Unlike liquids, which we can assume are incompressible, gasses expand to fill their container. In the process of expanding or compressing, gasses are subject to doing work or being worked upon. These processes affect the internal energy of the gas, which in turn affects its temperature and pressure. Our model captures these effects.

Since a contained gas expands to fill its container we must always represent its volume. This means that we do not have to provide distinct perspectives according to combinations of `Geometric-Properties` and `Gravity`. However, the relationship between the pressure and volume of a gas depends signficantly on temperature[6]. Hence we must introduce different perspectives according to whether or not thermal properties are considered.

[6] In reality it does for liquids, too, but this effect is so small that typically it is ignored.

Figure 8: Definition of Contained-Gas

The version of the ideal gas law used depends on whether or not thermal properties are under consideration.

```
(defentity (Contained-Gas (C-S ?sub gas ?can))
  (Q= (pressure (gas-in ?can) :ABSOLUTE)
      (pressure (C-S ?sub gas ?can) :ABSOLUTE))
  (Q= (volume (C-S ?sub gas ?can))
      (- (volume ?can) (volume (liquid-in ?can)))))

(defperspective (thermal-gas ?cg)
  Individuals ((?cg :type Contained-Gas
                    :form (C-S ?sub gas ?can)
                    :conditions (Consider (thermal-properties ?cg))))
  Relations ((Q= (pressure ?cg :ABSOLUTE)
                 (/0+ (heat ?cg) (volume ?cg))))) ;; Ideal gas law

(defperspective (non-thermal-gas ?cg)
  Individuals ((?cg :type Contained-Gas
                    :conditions (not (Consider (thermal-properties ?cg)))))
  Relations ((Q= (pressure ?cg :ABSOLUTE)
                 (/0+ (mass ?cg) (volume ?cg))))) ;; Non-thermal approximation
```

The defentity in Figure 8 links the properties of the contained gas to the properties of the container and any contained liquid in it. In particular, the pressure of the gas-in is the pressure of the contained gas (again, assuming a single substance), and the volume of the contained gas is determined by the difference between the volume of the container and the volume of the liquid-in.

Physically, what constrains the pressure of a gas? When a gas is sufficiently above its boiling point, its behavior is approximated by the ideal gas law:

$$PV = mRT = U$$

where P, V, m and T represent pressure, volume, mass and (absolute) temperature, respectively. R is the gas constant for the substance in question; U is the internal energy of the gas, which for simplicity will be referred to as heat.

Because QP theory requires a causal model, we must represent the ideal gas law as a set of directed influences. The first step is to identify the independent parameters, which form the inputs to the causal chains. These are always the quantities which can be directly influenced by some process. As with liquids, it is reasonable to choose mass and heat as independent parameters, since there are clearly-identifiable processes which directly influence them. In addition, volume is viewed as independent, since the volume of a contained-gas is determined by the volume of its container[7].

With heat, mass, and volume identified as independent parameters, we can solve for the remaining dependent ones:

$$P = U/V; \quad T = U/m;$$

[7]Expansion and compression processes have been developed (in other models) which directly influence a container's volume.

The constant R is dropped since it does not affect the qualitative behavior of a gas. The equation for temperature is the same constraint already imposed by `Complex-Physob`. Since contained stuffs are already `Volumetric-Physobs` (see Figure 5) and considering thermal properties makes them `Complex-Physobs` (see Figure 2), temperature is already appropriately constrained.

The expression for pressure may seem unintuitive, since it involves neither temperature nor mass. Intuitively, when gas is added to a closed container, or when a contained gas is heated, the pressure of the gas increases. But in both cases heat is being added to the gas while its volume remains constant. The model predicts that if the amount of the gas could be increased while its heat is held constant (say by adding gas at absolute zero temperature), then the pressure would remain unchanged. This result does not conflict with an intuitive view based on a product of mass and temperature, since the temperature in the this case would be decreasing, and the net influence on pressure would be ambiguous.

Figure 8 also encodes this analysis using two perspectives. The `Thermal-Gas` perspective defines the pressure of the gas as the ratio of heat and volume (through the `Q=//0+` combination).[8] Thus if the volume of the contained gas is decreased and/or its heat increased, the pressure will increase. This corresponds with the result derived from the ideal gas law. The `Non-Thermal-Gas` perspective is similar, but defines the pressure of the gas as the quotient of mass and volume. This is the most reasonable approximation available when thermal properties are not being considered.

Possible phase combinations Recall that we may independently decide whether or not to consider liquids and/or gases. Realistically, we are either considering liquids only, gases only, or situations where both may coexist. Each combination changes how the possible contents of the container are viewed. Here we describe the consequences of these different phase combinations.

There are three special cases which may be independently treated or not when liquids are considered, described in Figure 9. First, we can model a container as `Empty` when it has no liquid. Second, we can model a container as `Full` when the liquid completely takes up the volume of the can. Third, we can define `Overflowed` as occurring when the volume of the contained liquid is greater than that of the can. Certainly the latter is unintuitive, since the liquid is individuated by being in the container, rather than being "of" the container in some sense. However, it is useful to mark the existence of such conditions as potential hazards. The predicate `Unsafe-Condition` signals such violations. When used properly by external reasoning systems, this convention allows unsafe aspects of states to be identified.

If it were necessary, an overflow process could easily be added to gauge the severity of the problem. This process would remove liquid at a rate depending on the level of the liquid above the top of the container. The destination of the liquid removed would remain implicit, thus avoiding the necessity of specifying the details of the container's surroundings. Should such information be available, a cleaner technique would be to use

[8] The use of `/0+` is a signal to QPE that the parameters involved in the quotient are never negative, which allows it to use simpler internal justifications.

Figure 9: Definition of single-substance phase mixtures

```
(defview (Empty ?can ?sub)
  Individuals ((?can :type container
                     :conditions (Consider liquid)
                     (Consider (Empty-Container ?can)))
               (?sub :type substance))
  QuantityConditions ((equal-to (A (amount-of-in ?sub liquid ?can)) ZERO)))
(defview (Full ?can ?sub)
  Individuals ((?can :type container)
               (?sub :type substance
                     :conditions (Consider liquid) (Consider (Full-Container ?can))))
  QuantityConditions ((equal-to (A (volume (C-S ?sub liquid ?can))) (A (volume ?can)))))
(defview (Overflowed ?cl)
  Individuals ((?cl :type Contained-Liquid
                    :form (C-S ?s LIQUID ?c)
                    :conditions (Consider (Overflow ?c))))
  QuantityCondition ((greater-than (A (volume ?cl)) (A (volume ?c))))
  Relations ((Unsafe-Condition ?c)))
```

the overflow to infer the existence of a fluid path to the surroundings, and capture the dependence on level by making the conductance of the path depend on it (see Section 4.2.4).

The Evacuated view (Figure 10) does for contained gasses what Empty does for contained liquids. The pressure of the gas-in when there doesn't happen to be any gas in the container is of course zero. The qualitative proportionality linking the pressure of gas-in to the amount-of-in provides a smooth transition to the normal laws of contained gases. The Liquid-Substance-in-Container perspective relates the volume of the liquid-in to the amount-of-in. The relationship with volume is slightly redundant with that imposed by the contained-stuff definition, but this one imposes the correct constraint when there actually isn't any liquid in the container.

The last two perspectives in Figure 10 pin the relevant values of abstract container-dependent individuals when ignoring phases. In the Never-Liquid perspective the volume of the liquid-in is set to zero, thus freeing the entire volume of the container to be filled by gas. In the Never-Gas perspective, the pressure of the gas-in is set to zero, thus removing any contribution to the pressure of the liquid (if any) from potential gases.

4.2.4 Paths, Portals and Connectivity

So far we have described objects in isolation (e.g., physobs) or objects that are related by definition (e.g., a contained stuff and its container). Here we describe a vocabulary for representing connections found in typical structural descriptions. First we investigate some design choices, and then explain models for fluid paths, thermal paths, and portals.

Design choices for connectivity One extreme strategy for representing connectivity is to make connections as abstract as possible. This is the strategy used by most qualitative

Figure 10: Definition of single-substance mixtures, continued

```
(defview (Evacuated ?can ?sub)
  Individuals ((?can :type container)
               (?sub :type substance
                     :conditions (Consider gas)
                     (Quantity (Amount-of-in ?sub gas ?can))))
  QuantityConditions ((equal-to (A (amount-of-in ?sub gas ?can)) ZERO))
  Relations ((equal-to (A (pressure (gas-in ?can) :ABSOLUTE)) ZERO)
             (Qprop+ (pressure (gas-in ?can) :ABSOLUTE)
                     (amount-of-in ?sub gas ?can))))
(defperspective (Liquid-Substance-in-Container ?can ?sub)
  Individuals ((?can :type container
                     :conditions (Consider liquid))
               (?sub :type substance))
  Relations ((Qprop+ (volume (liquid-in ?can)) (amount-of-in ?sub liquid ?can))
             (Ordered-Correspondence ((A (volume (liquid-in ?can))) ZERO) ;; Single-Substance Ass
                                     ((A (amount-of-in ?sub liquid ?can)) ZERO))))
(defperspective (never-liquid ?can)
  Individuals ((?can :type container
                     :conditions (not (consider liquid))))
  Relations ((equal-to (A (volume (liquid-in ?can))) ZERO)))
(defperspective (never-gas ?can)
 Individuals ((?can :type container
                    :conditions (not (consider gas))))
 Relations ((equal-to (A (pressure (gas-in ?can) :ABSOLUTE)) ZERO)))
```

models, including non-QP models. However, this strategy has several limitations. First, it does not explicitly represent the fact that there can be different kinds of stuff inside a path at distinct times. This is not a problem if real fluids can be accurately modeled as abstract stuffs, as system-dynamics models do [2]. Anyone who has tried debugging plumbing systems, however, knows that this is often not always a realistic approximation! Second, the purely abstract path representation does not allow the geometry of the container and the arrangements of stuffs inside to be taken into account. A hole drilled in the middle of a water tank, for example, will not drain it completely, while a hole drilled on the bottom will. For some problems, the ability to reason about the geometry of the piping system is essential.

Our model abstracts all structural objects into two kinds: containers and paths which connect them. Every fluid path connects exactly two distinct containers. Abstract nodes, commonly used in modeling electrical circuits, are not allowed. The reason is that they are inconsistent with our view of causality as unidirectional and loop–free. To see this, imagine glueing together three pipes in series. The resulting assembly should behave as a single pipe. The problem is that there is no consistent rendering of causal directedness which can account for the pressures at the internal nodes. For example, if one end of the pipe sees an increasing pressure while the other end sees a decreasing pressure, the pressures at the internal nodes will be ambiguous. This could be explained by having each node determine its pressure by looking at its two adjacent nodes. But this requires causality to run in both directions through the center pipe, which is unintuitive.

It is therefore necessary to model nodes in a piping system as containers, whose pressures vary with the amount of fluid present. This choice has the disadvantage that one must deal with extra contained stuffs. More significantly, a node modeled as an accumulator does not obey Kirchoff's Current Law—in general, the flow out will not equal the flow in. New (and often unwanted) behaviors emerge as node pressures rise and fall. One solution is to "pre-assemble" multiple pipes into a single path, and model the system accordingly. This is part of a larger problem of mapping structural descriptions to structural abstractions. At present, this is done manually. A second alternative, common in engineering analyses, is to only consider steady-state behaviors (see Section 4.5.1).

We introduce the idea of a *portal* to reason about the geometry of stuffs inside a container. Many problems do not require the level of detail represented by portals. Consequently, we use modeling assumptions to control whether or not portals are introduced for any particular analysis. If the assumption `(Consider Portals)` is false, the QP interpreter uses a more abstract model of path.

Another design choice concerns the representation of conductance. In physics, conductance refers to how easily stuff can flow through a path. In a qualitative physics, conductance shows up as a factor affecting rates associated with flow processes. Conductance can be modeled in two ways. The first is not to represent it at all. Many qualitative analyses are concerned with making broad predictions about systems having only fixed conductances, so the particular value is irrelevant. The second choice is to introduce an explicit quantity for a path's conductance. This provides more accurate credit assignment if one is performing a comparative analysis. Our model provides both options, controlled by the modeling assumption `(Consider (fluid-conductance ?path))`. The assumption `(Consider (thermal-conductance ?path))` plays a similar role for heat paths.

Finally, it is often convenient to place restrictions on what kinds of stuff can flow through particular paths and in what directions. For instance, some piping systems have check valves which prevent liquid from flowing in one direction. An open trough leading from one container to another works perfectly well as a path for liquids, but will not successfully convey air between them. Our vocabulary for connections includes restrictions which can be used to model situations like these.

A purist might insist that scenario modelers always resort to a CAD-style encoding of a structural description, and derive restrictions on the kinds of flows which can occur through paths based on a "first principles" analysis. We lean towards this view ourselves, but also recognize that (a) scenario modelers have a hard enough job as it is without us making it harder for them; and (b) such a first principles analysis will need a set of distinctions like ours to express the results of their derivations anyway.

We assume that consistency tests on structural descriptions, such as ensuring that each path only connects to two components, are carried out by a preprocessor. It would be easy to install such checks in the domain model, but separating them makes more sense pragmatically because their encoding depends on interface issues as well as inferential ones.

Fluid paths Figure 11 provides the starting point for the definition of fluid paths. All fluid paths are physobs, as enforced by the first `defentity`. A fluid path is a `gas-path` if it

Figure 11: Definition for Fluid–Paths

```
(defentity Fluid-Path (Physob ?self))
(defperspective (General-Fluid-Path ?path)
  Individuals ((?path :type fluid-path
                      :conditions (Consider capable-fluid-paths))))
(defperspective (Liquid-Path ?path)
  Individuals ((?path :type general-fluid-path
                      :conditions (Consider liquid))))
(defentity Liquid-Path (Possible-Path-Phase ?self liquid))
(defperspective (Gas-Path ?path)
  Individuals ((?path :type general-fluid-path
                      :conditions (Consider gas))))
(defentity Gas-Path (Possible-Path-Phase ?self gas))
(defpredicate (Possible-Path-Phase ?path ?st)
  (Fluid-Path ?path)
  (Consider ?st))
```

Figure 12: Defining connections

```
(defpredicate (Fluid-Connection ?path ?from ?to)
  (Connects-To ?path ?from ?to)  (Connects-To ?path ?to ?from))
(defpredicate (Connects-To ?path ?from ?to)
  (Path-Container ?path ?from)   (Path-Container ?path ?to))
```

allows gasses to flow, a `liquid-path` if it allows liquids to flow, and a `General-Fluid-Path` if it allows both liquids and gasses to flow.

The representation of single-substance paths might seem overly complicated, but is necessary to provide flexibility for scenario modelers. The first perspective allows the modeler to declare all fluid paths to be general fluid paths, by assuming (`Consider capable-fluid-paths`).

Recall that a modeler may choose independently whether or not to consider gasses or liquids in a particular analysis. If one is considering liquids and not gasses, say, then a `general-fluid-path` should only act as a liquid path and not as a gas path. The next two perspectives in Figure 11 provide this ability. Finally, the predicate `Possible-Path-Phase` provides a functional encoding of the phase(s) which a particular path is allowed to carry. This is essential for the general-purpose fluid-flow process, described in Section 4.3.2.

Figure 12 shows the relationships which link a fluid path to its containers. The predicate `Connects-To` is used by flow processes to establish whether or not fluid can flow in a particular direction. Thus the modeler can declare a unidirectional path by asserting a single instance of `Connects-To`. Since `Fluid-Connection` implies `Connects-To` in both

Figure 13: Establishing possible path contents without portals

```
(defperspective (Fluid-Wireup ?path ?can)
  Individuals ((?path :type Fluid-Path
                      :conditions (not (Consider Portals))
                      (Connects-to ?path ?can ?dest) (Possible-Path-Phase ?path ?st))
               (?c-s  :type Contained-Stuff
                      :form (C-S ?sub ?st ?can)))
  Relations ((Filled ?path ?c-s)))
```

Figure 14: Direct implications of connectivity

```
(defperspective (Thermal-Wireup ?path ?can)
  Individuals ((?path :type Fluid-Path
                      :conditions (Path-Container ?path ?can)
                      (Consider (thermal-properties ?can))))
  Relations ((Consider (thermal-properties ?path))))
(defQuantity-Type fluid-conductance individual)
(defperspective (Conductive-Path ?path)
  Individuals ((?path :type Fluid-Path
                      :conditions (Consider (Fluid-Conductance ?path))))
  Relations ((Quantity (fluid-conductance ?path))
             (not (less-than (A (fluid-conductance ?path)) ZERO))))
```

directions, asserting it declares a path to be bi-directional. The predicate `Path-Container` expresses the fact that the given path and container are joined; this information is used below to establish several consequences of connectivity.

The possible interaction of fluids inside a container and the fluid path are expressed by the predicate `Filled`. `(Filled ?path ?stuff)` says that `?stuff` is touching `?path` at on end, and thus could be involved in a flow[9]. When portals are under consideration, `Filled` is inferred from the existence and heights of liquids relative to portals (see Section 4.2.4.) When portals are ignored, we presume that every stuff in a container can potentially flow through every path involving it. Figure 13 shows how this is done.

Making fluid connections has other implications aside from enabling flows. For example, if thermal properties are being considered, fluid flows will affect them as well as volumetric properties. The `Thermal-Wireup` perspective in Figure 14 ensures that such thermal properties are considered when appropriate. The `Conductive-Path` perspective of Figure 14 introduces the quantity `fluid-conductance` when it should be considered.

Flows, as Section 4.3.2 details, require a pressure difference to occur. But given con-

[9]The name `Filled` is something of a misnomer, since a path can be `Filled` with up to four things (assuming two phases and a single substance), while it would really only be filled with one. A more descriptive name might have been `Included-in-path-contents-or-at-least-touching`.

Figure 15: Selecting which pressure to use in inferring flow

```
(defperspective (Pressure-Definer ?path ?can (bottom ?can))
  Individuals ((?path :type Fluid-Path
                      :conditions (Path-Container ?path ?can) (not (Consider Portals)))))
```

tainers which can hold more than one contained stuff, how do we know which pressure to use?

Recall that the abstract individuals `liquid-in` and `gas-in` gave us a more modular way to represent the properties of mixtures in a container. Similarly, we introduce the notion of a `Pressure-Definer` as a source of information about pressures to insulate us from whether or not we are using portals. This insulation greatly simplifies the description of material flows. `(pressure-definer ?path ?can ?obj)` means that `?obj` should be the entity whose pressure is used as the pressure of `?can` for `?path`. Since every path has exactly two distinct containers, there will be two distinct `pressure-definers` for each path. Figure 15 shows the simplest approximation for pressure definers: When no other information is available, use the pressure at the bottom of the container. Physically, this is tantamount to restricting all fluid paths to connect to bottoms of containers.

Heat paths Heat paths (see Figure 16) connect two distinct `simple-thermal-physobs`—that is, physical objects which have a temperature. Heat paths are simpler than fluid paths because (a) internal energy doesn't come in phases and (b) geometry (at least in this level of modeling) is irrelevant[10]. `Thermally-Connects-To` indicates One-way connections, and `Heat-Connection` indicates bidirectional thermal paths. `thermal-conductance` represents a path's ability to transmit heat. As with `fluid-conductance`, the introduction of thermal conductance is controlled by a modeling assumption.

Valves Valves are employed to regulate or restrict flows through paths. The simplest model of a valve is binary, providing an on/off switch for fluid flow. This level of model can easily be achieved by introducing `Blocked` (see below) as an explicit assumption on a fluid path and using actions to correspond to changing its state [7], so we do not discuss it further. A slightly more complex model has valves affecting the conductance of a path. A path can of course have multiple valves. If any valve is closed, the path is blocked and

[10]In a more detailed model heat paths would be inferred from the geometry of the system and the existence of stuffs, and the conductance would depend on the nature and geometry of the stuff providing a physical connection. However, we include so little information about materials and container geometry that this additional level of detail would be useless. Whole textbooks are written on heat transfer, which analyse special cases analytically and describe how to use finite element methods to derive numerical solutions for more realistic shapes. We suspect that there may be one or two useful levels of detail between this model and a quantitative geometry, but that the extra leverage they provide is not very high.

Figure 16: Definition of heat paths

```
(defQuantity-Type thermal-conductance individual)
(defentity Heat-Path
  (Physob ?self))
(defperspective (Variable-Thermal-Conductance ?path)
  Individuals ((?path :type Heat-Path
                      :conditions (Consider (thermal-conductance ?path))))
  Relations ((Quantity (thermal-conductance ?path))
             (greater-than (A (thermal-conductance ?path)) ZERO)))
(defpredicate (Heat-Connection ?path ?from ?to)
  (Heat-Path ?path)
  (Thermally-Connects-To ?path ?from ?to)
  (Thermally-Connects-To ?path ?to ?from))
```

Figure 17: Valve definition

```
(defQuantity-Type open-area individual)
(defQuantity-Type change-rate individual)
(defentity valve
  (Physob ?self)
  (Non-Negative-Quantity (open-area ?self)))
(defperspective (Valve-in-Path ?valve ?path)
  Individuals ((?path :type fluid-path
                      :conditions (Consider (Valves ?path)))
               (?valve :bind (Valve-in ?path))))
(defpredicate (Valve-in-Path ?valve ?path)
  (Valve ?valve))
```

its conductance equals zero. This implicit disjunction makes the representation of valves a bit tricky.

Figure 17 provides the basic definition for valves. A valve is a physob whose open-area is never negative. If we are considering valves, we assume that each path has at least one. The generic valve `valve-in`, introduced by the `Valve-In-Path` perspective, provides a minimum of one valve per path. (The scenario modeler, of course, is free to define as many as necessary by using `Valve-In-Path`.)

Figure 18 defines the possible status of a valve using the two views: `Open-Valve` and `Closed-Valve`. A valve is open whenever its open-area is positive, and is closed otherwise. A single closed valve along a path is sufficient to cause the path to be *Blocked*, which forces the path's conductance to zero. Only if all valves are open is the path considered aligned. This is enforced by the fact that `Blocked` is a closed predicate, which means that it will be assumed to be false for all conditions in which it is not known to be true. That is, unless one knows of a closed valve, one assumes that the path is aligned.

Figure 18: Valve status

```
(defview (Open-Valve ?valve)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Valves ?path))))
  QuantityConditions ((greater-than (A (open-area ?valve)) ZERO)))
(defview (Closed-Valve ?valve)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Valves ?path))))
  QuantityConditions ((equal-to (A (open-area ?valve)) ZERO))
  Relations ((Blocked ?path)
             (equal-to (A (conductance ?path)) ZERO)))
(defClosed-Predicate Blocked)
(defperspective (Aligned ?path)
  Individuals ((?path :type Fluid-Path
      :conditions (not (Blocked ?path))))
  Relations ((only-during (greater-than (A (conductance ?path)) ZERO))))
```

Figure 19: Valve dynamics

```
(defprocess (Changing-Valve ?valve)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Changing-Valves ?path))))
  Relations ((Quantity (change-rate ?valve)))
  Influences ((I+ (open-area ?valve) (A (change-rate ?valve)))))
(defview (Opening-Valve ?valve)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Changing-Valves ?path))))
  QuantityConditions ((greater-than (A (change-rate ?valve)) ZERO)))
(defview (Closing-Valve ?valve)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Changing-Valves ?path))))
  QuantityConditions ((less-than (A (change-rate ?valve)) ZERO))
  Relations ((greater-than (A (open-area ?valve)) ZERO)))
(defperspective (Changing-Conductance ?path)
  Individuals ((?valve :type Valve
                       :conditions (Valve-In-Path ?valve ?path)
                       (Consider (Changing-Valves ?path))
                       (Aligned ?path)))
  Relations ((Qprop+ (conductance ?path) (open-area ?valve))))
```

Figure 20: Portals detail how paths connect to containers

```
(defperspective (Portal (PT ?can ?path))
  Individuals ((?can :type Container
                     :conditions (Consider Portals)
                     (Consider (Geometric-Properties ?can))
                     (Path-Container ?path ?can))))
(defperspective (Portal (PT ?can ?pump))
  Individuals ((?can :type Container
                     :conditions (Consider Portals)
                     (Consider (Geometric-Properties ?can))
                     (Pump-Container ?pump ?can))))
```

Figure 19 defines a process for changing a valve's status. When considering `Changing-Valves`, a valve has a change-rate quantity which directly influences its open-area. The two views—`Opening-Valve` and `Closing-Valve`—are used to distinguish the possible directions of change. The model provides no constraint on the `change-rate`, so the scenario may constrain it as desired. The `Changing-Conductance` perspective relates a valve's open-area to the conductance of its fluid-path, as long as the path is aligned. Modeling many control systems requires modeling valves whose state is linked to system parameters. This model can be modified to suit this purpose by (a) adding a precondition to the `Changing-Valves` process, controlled by other processes or views, which determines when it is acting and (b) by imposing the appropriate sign constraints on the `change-rate`.

Portals The abstract model of containers and paths suffices for many problems. However, sometimes it is important to represent the geometry of the interface between paths and containers. If there are two holes in a water tank at different heights, for example, we know the higher one will run dry before the lower one. If we are trying to siphon water out of a tub, it is important to keep the inlet of the siphon below the water line. Following the terminology used by Hayes [10], we call these interfaces *portals*.

We consider portals to be distinct entities, whose existence depends on the connection between some form of fluid path and a container. The function `PT` maps from containers and paths to portals. That is, `(PT ?can ?path)` refers to the portal formed by connecting `?path` to `?can`. Clearly this encoding is unable to distinguish the portals of a path connected to the same container at both ends; fortunately this situation rarely arises in engineered fluid and thermal systems.

Figure 20 shows the perspectives which introduce portals. Notice that in addition to requiring the consideration of geometric properties of the container, we also require a global assumption that portals are relevant. The reason for the extra assumption is that portals are expensive to reason about, hence we offer the option of modeling geometric properties partially (i.e., `Geometric-Properties` assumed and `Portals` false) or not at all (both `Geometric-Properties` and `Portals` assumptions false), as well as in full detail.

Figure 21: Properties of portals

```
(defentity (Portal (PT ?can ?path))
  (Quantity (height (PT ?can ?path) ?pt))
  (not (less-than (A (height (PT ?can ?path))) (A (height (bottom ?can)))))
  (not (greater-than (A (height (PT ?can ?path))) (A (height (top ?can)))))
  (Quantity (pressure (PT ?can "path)))
  (Quantity (pressure (PT ?can "path) (gas-in ?can)))
  (Q= (pressure (PT ?can ?path) :ABSOLUTE)
      (+ (pressure (gas-in ?can) :ABSOLUTE)
         (pressure (PT ?can ?path) (gas-in ?can)))))
(defperspective (pressure-definer ?path ?can ?pt)
  Individuals ((?pt :type portal
                    :form (PT ?can ?path)
                    :conditions (Consider Portals))))
```

The reason for having two perspectives is that our model treats pumps as a special kind of path (See Section 4.3.4).

The basic properties of portals are defined in Figure 21. A portal has a `height`, which is constrained to lie between the container's top and bottom. It has a `pressure`, which is defined as the pressure of the container's `gas-in` plus the pressure contributed by the weight of any liquid above the portal. The latter is represented as `(pressure ?pt (gas-in ?can))`, e.g., the difference between the portal pressure and the pressure of the `gas-in`. This simple definition of pressure puts the complexity elsewhere, namely in the definition of the constituent pressures.

Given that we are considering only single-substance systems, a portal is in contact with either a contained gas, a contained liquid, or neither. Since we are approximating portals by only a single height, we ignore the fact that in real portals there are times when both the liquid and gas would be in contact, as the interface between them moves between the heights of the top and bottom of the portal.

The view `Submerged-In` describes the case where the portal is in contact with liquid. This occurs when the portal's height is lower than the liquid's level. When the portal is submerged, we stipulate that the path is `Filled` with the liquid (See Section 4.2.4). (Notice that this model ignores the possibility of complicated geometry in the fluid paths, which would allow part of a piping system to remain empty while another part is full. We have not delved into this level of detail because the contained-stuff ontology is not suitable for representing finite-sized "chunks" of stuff (i.e., bubbles) inside a fluid path.) When a portal is submerged, the pressure contributed by the liquid above it is positive, and is an increasing function of the level.

The view `Dry-Portal` describes the case where the portal is not submerged. The fact that there is no liquid above the portal is reflected by the constraint that the pressure of the portal relative to the gas pressure is equal to zero. Notice that being dry does not necessarily imply that the portal is in contact with a gas, since there might not be any gas in the container. The consequence of dryness when gas is present is represented by

Figure 22: Describing what touches a portal

```
(defview (Submerged-in ?pt ?cl)
  Individuals ((?cl :type contained-liquid
                    :form (C-S ?sub liquid ?can))
               (?pt :type portal
                    :form (PT ?can ?path)))
  QuantityConditions ((greater-than (A (level ?cl)) (A (height ?pt))))
  Relations ((only-during (Filled ?path ?cl))
             (only-during (Exposed-to ?pt ?cl))
             (Qprop+ (pressure ?pt (gas-in ?can)) (level ?cl))
             (greater-than (A (pressure ?pt (gas-in ?can))) ZERO)))

(defview (Dry-Portal ?pt)
  Individuals ((?pt :type portal
                    :form (PT ?can ?path)))
  QuantityConditions ((not (greater-than (A (level (liquid-in ?can))) (A (height ?pt)))))
  Relations ((equal-to (A (pressure ?pt (gas-in ?can))) ZERO)))

(defperspective (Exposed-to ?pt ?cg)
  Individuals ((?cg :type contained-gas
                    :form (C-S ?sub gas ?can))
               (?pt :type Dry-Portal
                    :form (PT ?can ?path)))
  Relations ((only-during (Filled ?path ?cg))))
```

the Exposed-To perspective, namely that the path is then Filled with the gas. A subtle point: Notice that Dry-Portal is predicated on the level of Liquid-In. This means that it will hold even when liquids are not considered (recall the Empty and Never-Liquid perspectives), and so in that case every portal will touch the gas of its container, if any.

To weed out any violations in transitivity, it is important to ensure that as many

Figure 23: Relating pressures of portals in the same container

```
(defPerspective (Common-Portals ?pt1 ?pt2)
  Individuals ((?pt1 :type Portal
                     :form (PT ?can ?path1))
               (?pt2 :type Portal
                     :form (PT ?can ?path2)
                     :test (alphalessp ?path1 ?path2)))
  Relations ((Ordered-Correspondence
              ((A (pressure ?pt1 :ABSOLUTE))      (A (pressure ?pt2 :ABSOLUTE)))
              ((A (pressure ?pt1 (gas-in ?can))) (A (pressure ?pt2 (gas-in ?can)))))))

(defPerspective (Common-Submerged-Portals ?pt1 ?pt2)
  Individuals ((?pt1 :type Submerged-Portal
                     :form (PT ?can ?path1)
                     :conditions (Common-Portals ?pt1 (PT ?can ?path2)))
               (?pt2 :type Submerged-Portal
                     :form (PT ?can ?path2)))
  Relations ((Ordered-Correspondence
              ((A (pressure ?pt1 (gas-in ?can))) (A (pressure ?pt2 (gas-in ?can))))
              ((A (height ?pt2))                 (A (height ?pt1)))))))
```

Figure 24: Relating pressures of portals which share a common path

```
(defPerspective (Same-Path-Portals ?pt1 ?pt2)
  Individuals ((?pt1 :type Portal
                     :form (PT ?can1 ?path))
               (?pt2 :type Portal
                     :form (PT ?can2 ?path)
                     :test (alphalessp ?can1 ?can2)))
  Relations ((equal-to (A (height ?pt1)) (A (height ?pt2)))  ;;***These are = for now
             (Ordered-Correspondence
               ((A (pressure ?pt1 :ABSOLUTE))          (A (pressure ?pt2 :ABSOLUTE)))
               ((A (pressure ?pt1 (gas-in ?can1)))     (A (pressure ?pt2 (gas-in ?can2))))
               ((A (pressure (gas-in ?can1) :ABSOLUTE)) (A (pressure (gas-in ?can2) :ABSOLUTE))))))
(defPerspective (Same-Path-Submerged-Portals ?pt1 ?pt2)
  Individuals ((?pt1 :type Submerged-Portal
                     :form (PT ?can1 ?path)
                     :conditions (Same-Path-Portals ?pt1 (PT ?can2 ?path)))
               (?pt2 :type Submerged-Portal
                     :form (PT ?can2 ?path)))
  Relations ((Ordered-Correspondence
               ((A (pressure ?pt1 (gas-in ?can1))) (A (pressure ?pt2 (gas-in ?can2))))
               ((A (level (liquid-in ?can1)))      (A (level (liquid-in ?can2)))))))
```

inequality relations of interest are derivable as possible. Figure 23 shows how this is done for two portals sharing a common container. The contribution to each portal's pressure made by the gas in the container will be identical, so any difference in their pressures must be due to a difference in the relative heights of any liquid above the portal. If both portals are submerged, we know that their pressures are equal exactly when their heights are equal, and that if one portal is lower than another, then its pressure will be higher. These facts are encoded by the correspondences in the `Common-Portals` and `Common-Submerged-Portals` perspectives.

The `Same-Path-Portals` and `Same-Path-Submerged-Portals` perspectives in Figure 24 reflect the fact that the same laws apply to portals at each end of a fluid path. Note that paths are currently constrained to be level—that is, the portals at either end have the same height. This constraint could be relaxed by introducing a new quantity head to represent pressure at a fixed height. This is discussed further in Section 6.

It should be clear by now that our representation for portals is fundamentally different from the notion of port or terminal used in system dynamics or bond graphs. Like ports in these formalisms, portals provide an interface between components and connectors. But there the resemblance ends. Portals, in this model, are distinct entities, with a number of properties and possible states. This extra complexity is a necessary consequence of explicitly representing working fluids. However, it is important to remember that portals only need to be considered if one is worrying about geometric details. If this level of detail is undesirable, portals can be eliminated by the "flick of an assumption". This provides a dramatic simplification when reasoning about large-scale engineered systems at the level of system diagrams (see Section 5).

Figure 25: Process Definition for Heat Flow

```
(defQuantity-Type heat-flow-rate individual)
(defprocess (Heat-Flow ?src ?dst ?path)
  Individuals ((?path :type heat-path
                      :conditions (thermally-connects-to ?path ?src ?dst))
               (?src :type simple-thermal-physob)
               (?dst :type simple-thermal-physob))
  Preconditions ((heat-aligned ?path))
  QuantityConditions ((greater-than (A (temperature ?src :ABSOLUTE))
                                    (A (temperature ?dst :ABSOLUTE))))
  Relations ((quantity heat-flow-rate))
  Influences ((I+ (Heat ?dst) (A heat-flow-rate))
              (I- (Heat ?src) (A heat-flow-rate))))
```

4.3 Flow Processes

Several thermodynamic processes involve the transfer of material or energy from one location to another. They have a common pattern. Each involves a source, destination, and a path. Each requires a difference in some parameter (eg., `temperature` or `pressure`) to occur. Since the contained-stuff ontology does not provide a means to define pieces of stuff independently from containers, we cannot describe the details of the traversal of stuff from one place to another. Nor do we need to, for the kinds of systems-level analyses which motivate this model. The fact that there is some "stuff" which is conserved during the flow is encoded by the constraints on the source and destination. In particular, each flow process has an associated rate, which provides a negative direct influence on some property of the source (thus modeling "stuff" leaving the source) and a positive direct influence on some property of the destination (thus modeling "stuff" entering the destination).

The basic flow processes in this domain model are `Heat-Flow` and `Fluid-Flow`. We describe each in turn.

4.3.1 Heat Flow

The abstractness of internal energy (no phases, no changes in existence) makes heat flow one of the simplest processes to model. Figure 25 defines the `Heat-Flow` process. The source (`?src`) and destination (`?dst`) are both `simple-thermal-physobs`, which ensures they both have `temperature`. (The astute reader will notice that this does not necessarily ensure that they both have `heat`. The reason for this is explained below.) They must be connected by a heat path (`?path`), as indicated by the `individuals` specification[11].

For heat flow to occur, the path must be capable of supporting heat flow (i.e., `heat-aligned`) and the temperature in the source must be greater than that of the destination, as the quantity conditions indicate. When heat flow is occuring, `heat-flow-rate` becomes an

[11] The order of the specifications is designed for efficient matching.

Figure 26: Modifications to heat flow

```
(defperspective (simple-heat-rate ?pi)
  Individuals ((?pi :type (process-instance heat-flow)
                    :conditions (Active ?pi)
                    (?pi src ?src)(?pi dst ?dst))
               (?path :conditions (?pi path ?path)
                      (not (Consider (thermal-conductance ?path)))))
  Relations ((Q= (heat-flow-rate ?pi) (Q- (temperature ?src :ABSOLUTE)
                                          (temperature ?dst :ABSOLUTE)))))
(defperspective (variable-heat-rate ?pi)
  Individuals ((?pi :type (process-instance heat-flow)
                    :conditions (Active ?pi)
                    (?pi src ?src)(?pi dst ?dst))
               (?path :conditions (?pi path ?path)
                      (Consider (thermal-conductance ?path))))
  Relations ((only-during (quantity (temperature ?src ?dst)))
             (Q= (temperature ?src ?dst) (- (temperature ?src :ABSOLUTE)
                                            (temperature ?dst :ABSOLUTE)))
             (Q= (heat-flow-rate ?pi) (*+ (temperature ?src ?dst)
                                          (thermal-conductance ?path)))))
(defentity heat-sink
  (simple-thermal-physob ?self)
  (not (quantity (Heat ?self))))
```

influence on the heats of the source and destination, thus modeling the basic effect of the flow.

The predicate `heat-aligned` provides a means to summarize a variety of physical effects. For instance, some paths require a working fluid, whose properties are otherwise not of interest, to have non-negligable heat flow. Modeling the space between two objects as a heat path may make sense when they are close together, but not when they are far apart. An external theory can use `heat-aligned` to communicate these changes to the QP model. Section 4.5.3 describes methods for exploring both possibilities, or for assuming heat paths are aligned by default.

So far there are no constraints on `heat-flow-rate`. The model provides two approximations for `heat-flow-rate`, according to whether or not thermal conductance is being considered. If it is, the perspective `variable-heat-rate` (see Figure 26) introduces a conductance for the path, and constrains the rate to be the product of the conductance and the temperature difference. If we are ignoring thermal conductance, then the `simple-heat-rate` perspective constrains `heat-flow-rate` to be the temperature difference.

The reason `(temperature ?src ?dst)` needs to be defined explicitly is that QPE's modeling language does not provide arbitrary nesting of algebraic expressions. As Section 3 described, every algebraic expression in the modeling language has a corresponding causal interpretation. `Q=`, for example, is defined as a set of equality statements and qualitative proportionalities. Allowing complex expressions would obscure the modeler's intent concerning causality.

Figure 26 also shows our representation of heat sinks. A `heat-sink` is a `simple-thermal-physob` which cannot have `heat`. Being a `simple-thermal-physob` means that heat sinks can participate in heat flows. We are exploiting a property of our modeling language: a direct influence on a parameter which doesn't exist has no effect. Thus the process will have no effect on the temperature of the sink.

This is not the only way to model such sinks in QP theory. For example, one could use a "replenisher" process which supplies or removes additional heat from the sink to keep its temperature constant. The disadvantage of this scheme is that it requires an extra process for each sink. Or, one could define a sink as having both `heat` and `temperature`, but without any causal connection between them. However, unlike the other two schemes, this does not put the heat sink completely outside the modeling realm—for example, if we wish to enforce steady state (see Section 4.5.1), we would not want to reject a state simply because some heat sink has an increasing heat quantity.

4.3.2 Fluid Flow

Models of fluid flow can be extremely complex: Many hours of supercomputer time are currently spent solving fluid dynamics problems. As might be expected, our models will be much simpler. This simplicity is appropriate given our focus on system-level rather than detailed "component-level" phenomena. We ignore the dynamics involved in accelerating the mass of fluid in the path. We ignore the distinction between turbulent and non-turbulent flow. Even so, the model we have developed contains some (perhaps surprising) sophistications.

Previous QP models have tended to use separate processes to describe the flow of liquids and the flow of gasses. While simple, it has the disadvantage of obscuring many important underlying similarities. Several distinctions introduced earlier, most notably the concepts of `Pressure-Definer` and `Possible-Path-Phase`, allow us to represent the common, core phenomena of fluid flow by a single process. This process is then modified by additional perspectives which encode the consequences pertaining to liquids or gasses as needed. In our model these consequences pertain to the interactions of thermal properties with fluid flow. This section describes the basic model, and Section 4.3.3 describes the associated thermal model of fluid flow.

Let us examine how this is done in detail. Figure 27 describes the basic fluid flow process. The variable `?path` is constrained to be a fluid path, which subsumes both liquid and gas paths (recall Figure 11). The containers attached to the path are `?src` and `?dst`, as indicated by the `connects-to` predication. The predication on `possible-path-phase` provides the phase (`?st`). The source contained stuff, `?src-cs`, has the form `(C-S ?sub ?st ?src)`, which must be a contained stuff. Thus for every path which can contain a particular phase, every distinct substance `?sub` would give rise to a distinct instance of `Fluid-Flow`. (This is for upward compatibility with future models for describing multiple-substance systems.) The trigger involving the destination container `?dst` simply ensures that it is a container. Given the rest of our current model it must be a container, of course. However, including this individual explicitly allows us to refer to "the destination of a fluid

Figure 27: Process definition for fluid flow

```
(defQuantity-Type flow-rate individual)
(defprocess (fluid-flow ?src-cs ?dst ?path)
  Individuals ((?path :type fluid-path
                      :conditions (possible-path-phase ?path ?st)
                      (connects-to ?path ?src ?dst))
               (?src-cs :type contained-stuff
                        :form (C-S ?sub ?st ?src)
                        :conditions (Filled ?path ?src-cs))
               (?dst :type container)
               (?pr-src :conditions (Pressure-Definer ?path ?src ?pr-src))
               (?pr-dst :conditions (Pressure-Definer ?path ?dst ?pr-dst)))
  Preconditions ((aligned ?path))
  QuantityConditions ((Greater-than (A (pressure ?pr-src :ABSOLUTE))
                                    (A (pressure ?pr-dst :ABSOLUTE))))
  Relations ((Quantity flow-rate))
  Influences ((I+ (Amount-of-in ?sub ?st ?dst) (A flow-rate))
              (I- (Amount-of-in ?sub ?st ?src) (A flow-rate))))
```

flow process instance". The final two triggers find the pressure definers for the source and destination. As described above, this insulates our model from the decision of whether or not to use portals.

Notice that we have used a contained stuff as the source of the flow, but only require a destination container, rather than a destination contained stuff. This assymetry is important. If the destination of the flow were an explicitly named contained stuff, that stuff would have to exist before the instance of `fluid-flow` could be active. This would mean that we couldn't have a flow of some stuff into a container unless a contained stuff of that kind were already there. For instance, we could never pour water into an empty container. This is also the reason that the pressures used to determine flows (as specified by the `pressure-definer` predicate) must belong to some individual other than the contained stuff which is flowing.

As with the analogous `Heat-Flow` process, `Fluid-Flow` occurs whenever the path is `Aligned` (i.e., not `Blocked`), and the pressure in the source is greater than the pressure in the destination. And, again like `Heat-Flow`, there is a flow rate (here `flow-rate`) which acts to decrease the source `amount-of-in` while simultaneously acting to increase the destination `amount-of-in`.

How `flow-rate` is constrained depends on what one assumes about fluid conductance. Figure 28 shows the alternatives, which are analogous to those of thermal conductance. The `Simple-Fluid-Rate` perspective, which holds when fluid conductance is not being considered, sets the flow rate to be equal to the pressure difference. The `variable-fluid-rate` perspective, which holds when considering fluid conductance, "folds in" a dependence on the `fluid-conductance` of the path (defined in Figure 14).

Figure 28: Modifying flow rates according to conductance assumptions

```
(defperspective (simple-fluid-rate ?pi)
  Individuals ((?pi :type (process-instance fluid-flow)
                    :conditions (Active ?pi)
                    (?pi pr-src ?pr-src)(?pi pr-dst ?pr-dst))
               (?path :conditions (?pi path ?path)
                      (not (Consider (fluid-conductance ?path)))))
  Relations ((Q= (flow-rate ?pi) (Q- (pressure ?pr-src :ABSOLUTE)
                                     (pressure ?pr-dst :ABSOLUTE)))))

(defperspective (variable-fluid-rate ?pi)
  Individuals ((?pi :type (process-instance fluid-flow)
                    :conditions (Active ?pi)
                    (?pi pr-src ?pr-src)(?pi pr-dst ?pr-dst))
               (?path :conditions (?pi path ?path)
                      (Consider (fluid-resistance ?path))))
  Relations ((Quantity (pressure ?pr-src ?pr-dst))
             (Q= (pressure ?pr-src ?pr-dst) (Q- (pressure ?pr-src :ABSOLUTE)
                                                (pressure ?pr-dst :ABSOLUTE)))
             (Q= (flow-rate ?pi) (*0+ (pressure ?pr-src ?pr-dst)
                                      (fluid-conductance ?path)))))
```

Figure 29: Transfer of heat during fluid flow

```
(defprocess (thermal-fluid-flow ?ff)
  Individuals ((?ff     :type (process-instance fluid-flow)
                        :conditions (?ff dst ?dst) (?ff path ?path) (?ff src-cs (C-S ?sub ?st ?src)))
               (?st     :type phase)
               (?src-cs :type Contained-Stuff
                        :form (C-S ?sub ?st ?src)
                        :conditions (Consider (thermal-properties ?path)))
               (?dst-cs :bind (C-S ?sub ?st ?dst)))
  QuantityConditions ((Active ?ff))
  Relations ((Quantity heat-flow-rate))
  Influences ((I- (heat ?src-cs) (A heat-flow-rate))
              (I+ (heat ?dst-cs) (A heat-flow-rate))))

(defperspective (liquid-heat-flow-rate ?tff)
  Individuals ((?tff    :type (process-instance thermal-fluid-flow)
                        :conditions (Active ?tff) (?tff st liquid) (?tff ff ?ff))
               (?src-cs :conditions (?tff src-cs ?src-cs)))
  Relations ((Q= (heat-flow-rate ?tff)
                 (*0+ (flow-rate ?ff)
                      (temperature ?src-cs :ABSOLUTE)))))

(defperspective (gas-heat-flow-rate ?tff)
  Individuals ((?tff    :type (process-instance thermal-fluid-flow)
                        :conditions (Active ?tff) (?tff st gas) (?tff ff ?ff))
               (?src-cs :conditions (?tff src-cs ?src-cs)))
  Relations ((Quantity (temperature ?tff :ABSOLUTE))
             (greater-than (A (temperature ?tff :ABSOLUTE))
                           (A (temperature ?src-cs :ABSOLUTE)))
             (Q= (heat-flow-rate ?tff) (*0+ (flow-rate ?ff) (temperature ?tff :ABSOLUTE)))))
```

Figure 30: Thermal mixing due to fluid flow

```
(defview (thermal-fluid-mixing ?ff)
  Individuals ((?ff :type (process-instance fluid-flow)
                    :conditions (?ff dst ?dst) (?ff path ?path)
                    (?ff src-cs (C-S ?sub ?st ?src))
                    (Consider (thermal-properties ?path)))
               (?src-cs :bind (C-S ?sub ?st ?src))
               (?dst-cs :bind (C-S ?sub ?st ?dst)))
  QuantityConditions ((active ?ff)
                      (not (equal-to (A (temperature ?src-cs :ABSOLUTE))
                                     (A (temperature ?dst-cs :ABSOLUTE))))))
```

4.3.3 Thermal Effects of Fluid Flow

If thermal properties are being considered, fluid flow has some interesting phase-dependent complications. Heat is transferred along with the working fluid, so we must install influences on `heat` as well as on `amount-of-in`. Otherwise, the `heat` will remain constant even as the fluid objects appear and disappear. Figure 29 defines the `thermal-fluid-flow` process which represents this transfer. One should think of this process as a modifier of `Fluid-Flow` (note the explicit dependence on `?ff`, an instance of `Fluid-Flow`) which adds additional influences when thermal properties are being considered.

The rate of heat transfer depends on the phase of the flowing stuff. For liquids the rate is simply the product of the source stuff's temperature and the `flow-rate` of the fluid. For gasses, there is an additional energy transfer due to the work being done by the source as it expands, and on the destination as it is compressed. This additional heat transfer is folded in with the normal heat carried by liquids, by defining and using a temperature greater than the temperature of the source gas. The heat flow rate is then the product of the mass flow rate with this new temperature. For lack of a better owner, we let this temperature belong to the process itself.

As described above, the **temperature** of a full physob is defined as a ratio of **heat** and **mass**, which results in the following dependencies:

$$\textbf{temperature} \propto_{Q+} \textbf{heat}; \qquad \textbf{temperature} \propto_{Q-} \textbf{mass}$$

This can often result in ambiguity; for example, both **heat** and **mass** are decreasing at the source of a fluid flow and increasing at the destination, so the net effect on the **temperatures** cannot be resolved in the usual way, given the ambiguous combination of the $\propto_{Q+}$ and $\propto_{Q-}$.

This problem motivated the development of a technique for resolving ratios. Basically, we pair up the influencers on numerator and denominator, resolving the net influence of each pair in isolation. As long as no two pairs provide opposite influences, the derivative of the ratio will be unambiguously resolved. Using this technique requires ensuring that the temperature differences between fluids is known (i.e., a choice for the

relationship between temperatures is part of the constituents of a qualitative state). The Thermal-Fluid-Mixing view of Figure 30 does this.

Augmented with this technique, QPE is powerful enough to reason that the temperature at the source of a liquid flow remains constant, while the temperature at the destination behaves according to the difference in temperatures. This technique also solved another problem: recognizing that flow into an empty container results in a contained-stuff at the same temperature as the flow coming in. By requiring that a massless contained-stuff have constant temperature, it follows that the initial temperature will be the same as that of the liquid flowing in (otherwise it would be changing, a contradiction). This constraint also covers cases of multiple flows of different temperatures into an empty container.

4.3.4 Pumped Flow

Pumps are used to drive fluid flow when gravity won't. Pumps are modeled like paths: They don't have stuffs in them, but they move stuffs from place to place.

There are several design decisions concerning pump models. First, we must choose how to model a pump's flow rate. The simplest model of a pump assumes a constant (positive) flow rate, as long as there is fluid in the source container to be pumped. This model corresponds to a positive–displacement pump for liquids. Alternatively, we can model the pump's flow rate as a function of the pressure rise (or drop) across the pump. This model is based on the (more common) centrifugal pump, in which the flow rate depends on the pressure rise (or drop) across the pump. The rate of flow decreases as the pressure rise increases, until some maximum pressure is reached. We express our choice between these alternatives with the Pumped-Flow-Variation assumption.

When considering Pumped-Flow-Variation, we may also wish to consider the possibility that the pressure rise across the pump exceeds its maximum pumping pressure, causing a net flow in the reverse direction.[12] This modeling choice is activated with the Pump-Lossage consider assumption.

One possible behavior of pumps of general concern is cavitation, where the pressure changes in a pump cause the liquid inside it to boil. We do not model cavitation in detail, except to note that it is likely to occur when pumping liquids that are already boiling. Such possibilities are detected only when the Pump-Cavitation assumption is in force.

The above modeling assumptions concern the actual operation of the pump. We may also wish to control whether or not we distinguish between different pump behaviors. For example, when a pump moves fluid from a lower to a higher pressure it is doing work, but when it is moving fluid from a higher pressure to a lower one, the pump is coasting. The (Consider Pump-Status) assumption causes this distinction to be made.

Details of the pump model The basic definitions for pumps is shown in Figure 31. These definitions parallel the definitions for fluid paths described in Section 4.2.4. A pump may be either a liquid-pump, a gas-pump, or both (e.g., a general-fluid-pump).

[12]This model of a pump is equivalent to a constant displacement pump in parallel with a (restricted) flow path.

Figure 31: Types of pumps

```
(defentity Fluid-Pump
  (Physob ?self)
  (Quantity (flow-rate ?self))) ; This is the pump's actual flow-rate;
(defentity Liquid-Pump (Possible-Pump-Phase ?self liquid))
(defperspective (Liquid-Pump ?pump)
  Individuals ((?pump :type General-Fluid-Pump
                      :conditions (Consider liquid))))
(defentity Gas-Pump (Possible-Pump-Phase ?self gas))
(defperspective (Gas-Pump ?pump)
  Individuals ((?pump :type General-Fluid-Pump
                      :conditions (Consider gas))))
(defpredicate (Possible-Pump-Phase ?pump ?st)
  (Fluid-Pump ?pump)
  (Consider ?st))
```

Figure 32: Expressing pump connectivity

```
(defpredicate (Pump-connection ?pump ?src ?dst)
  (Pumps-From ?pump ?src)     (Pumps-To ?pump ?dst)
  (Pump-Container ?pump ?src) (Pump-Container ?pump ?dst))
(defperspective (pressure-definer ?pump ?can (bottom ?can))
  Individuals ((?pump :type fluid-pump
                      :conditions (Pump-Container ?pump ?can)
                      (not (Consider Portals)))))
```

Both `liquid-pumps` and `gas-pumps` are instances of `fluid-pumps`. The two perspectives ensure that a `general-fluid-pump` will be allowed to act as a `liquid-pump` and as a `gas-pump` when the corresponding phases are being considered. The relation `(possible-pump-phase ?pump ?st)` gives us access to the possible phases(s) of the pump. This is needed for the general-purpose pumped-flow process described below.

Figure 32 defines how pumps are connected, which parallels that of the fluid paths. The relationship `Pump-Connection` indicates that a pump connects two containers. The relationships `Pumps-From` and `Pumps-To` distinguish the directions involved (unlike simpler fluid paths, pumps are generally not bi-directional). Some inferences require only knowing connectivity and not direction; the relationship `Pump-Container` provides this information. When portals are not considered, the pressure-definers for pumps are the same as for other fluid paths.

Figure 33 defines two approximations for the pump's flow rate. The simpler model is `constant-flow-pump`, used when `pumped-flow-variation` is false, which simply constrains the rate to be positive. Since the flow rate is otherwise unconstrained, it will never

Figure 33: Two models of pump flow rates

```
(defperspective (constant-flow-pump ?pump)
  Individuals ((?pump :type fluid-pump
                      :conditions (not (consider (pumped-flow-variation ?pump)))))
  Relations ((Greater-than (A (flow-rate ?pump)) ZERO)))
(defperspective (variable-flow-pump ?pump)
  Individuals ((?pump :type fluid-pump
                      :conditions (consider (pumped-flow-variation ?pump)) (Pump-Connection ?pump ?src ?dst)
                      (Pressure-Definer ?pump ?src ?pr-src) (Pressure-Definer ?pump ?dst ?pr-dst)))
  Relations ((Positive-Quantity (flow-rate (SPEC ?pump))) ; This is the pump's no-load flow-rate;
             (Qprop+ (flow-rate ?pump) (pressure ?pr-src :ABSOLUTE))
             (Qprop- (flow-rate ?pump) (pressure ?pr-dst :ABSOLUTE))
             (Ordered-Correspondence ((A (flow-rate ?pump)) (A (flow-rate (SPEC ?pump))))
                                    ((A (pressure ?pr-src :ABSOLUTE))
                                     (A (pressure ?pr-dst :ABSOLUTE))))))
```

change. On the other hand, the `variable-flow-pump` model constrains the rate according to the pressures in the source and destination. A new positive quantity (`flow-rate (spec ?pump)`) is introduced to define the pump's no load flow characteristics. The `Correspondence` ensures that the pump's flow rate equals the no-load rate when the source and destination pressures are equal.

Before a pump can flow, it must be *primed*. In our model, a pump is primed whenever there is fluid (in the appropriate phase) at its inlet. Because we allow the possibility of losing pumps (i.e., negative flow), we must be able to establish priming in both directions. In addition, since a single pumped-flow process (described below) handles both positive and negative flows, it is necessary to use a single predicate to cover both possibilities.[13]

A pump is *forward primed* when there is a contained stuff at its inlet and it is not losing. The first two views in Figure 34 provide two independent ways for establishing this, depending on whether portals are included in the model. The first `forward-primed` view is used when ignoring portals; it requires a contained stuff in the source and a non-negative pump flow-rate. The second `forward-primed` view adds the requirement that the portal be `exposed-to` the stuff. The `backward-primed` views, which require considering `Pump-Lossage`, work similarly, except that they look at the pump's outlet.

Each of the four priming views establishes two results: that the pump is `primed` (a prerequisite for the pumped-flow process) and what the pump is pumping (in the form of `(pumping ?pump ?src-cs)`. This latter fact is used to determine the thermal aspects of pumped flow, as described in Section ??. When the pump is not primed in any way (i.e., all four views fail to be active), then it is `unprimed`, which implies that its flow rate is equal to zero.

With all the prerequisites in place, the actual description of pumped flow is quite simple, as shown in Figure 35. The process `pumped-flow` is active whenever it is `Primed` and turned `On`. When the flow-rate of the pump is positive, it acts to increase the amount

[13]Our modeling language does not support explicit disjunctions in preconditions or quantity conditions.

Figure 34: Representing pump priming

```
(defview (Forward-Primed ?pump ?st)
  Individuals ((?pump :type Fluid-Pump
                      :conditions (not (Consider Portals))
                      (Pump-Connection ?pump ?src ?dst) (Possible-Pump-Phase ?pump ?st))
               (?src-cs :type Contained-Stuff
                        :form (C-S ?sub ?st ?src)))
  QuantityConditions ((not (less-than (A (flow-rate ?pump)) ZERO)))
  Relations ((pumping ?pump ?src-cs)
             (primed ?pump ?st)))
(defview (Forward-Primed ?pump ?st)
  Individuals ((?pump :type fluid-pump
                      :conditions (Consider Portals)
                      (Pump-Connection ?pump ?src ?dst) (Possible-Pump-Phase ?pump ?st))
               (?src-cs :type Contained-Stuff
                        :form (C-S ?sub ?st ?src)
                        :conditions (Exposed-to (PT ?src ?pump) ?src-cs)))
  QuantityConditions ((not (less-than (A (flow-rate ?pump)) ZERO)))
  Relations ((pumping ?pump ?src-cs)
             (primed ?pump ?st)))
(defview (Backward-Primed ?pump ?st)
  Individuals ((?pump :type fluid-pump
                      :conditions (Consider (Pump-Lossage ?pump)) (not (Consider Portals))
                      (Possible-Pump-Phase ?pump ?st) (Pump-Connection ?pump ?src ?dst))
               (?dst-cs :type contained-stuff
                        :form (C-S ?sub ?st ?dst)))
  QuantityConditions ((not (greater-than (A (flow-rate ?pump)) ZERO)))
  Relations ((pumping ?pump ?src-cs)
             (primed ?pump ?st)))
(defview (backward-primed ?pump ?st)
  Individuals ((?pump :type Fluid-Pump
                      :conditions (Consider (Pump-Lossage ?pump)) (Consider Portals)
                      (Possible-Pump-Phase ?pump ?st) (Pump-Connection ?pump ?src ?dst))
               (?dst-cs :type contained-stuff
                        :form (C-S ?sub ?st ?dst)
                        :conditions (Exposed-to (PT ?dst ?pump) ?dst-cs)))
  QuantityConditions ((not (greater-than (A (flow-rate ?pump)) ZERO)))
  Relations ((pumping ?pump ?src-cs)
             (primed ?pump ?st)))
(defClosed-Predicate Pumping)
(defClosed-Predicate Primed)
(defperspective (Unprimed ?pump ?st)
  Individuals ((?pump :type Fluid-Pump
                      :conditions (not (Primed ?pump ?st))))
  Relations ((equal-to (A (flow-rate ?pump)) ZERO)))
```

of stuff in the destination and to decrease the amount of stuff in the source. Note that if the pressure-rise across the pump is sufficiently high such that the pump is losing, the flow rate will be negative, and the effects of the influences will be reversed. Also note that—unlike the fluid-flow process—it is possible for the pumped-flow process to be active even though its flow-rate is zero. A zero flow-rate has no effect on the amount of stuff at the source or destination.

There may be times when we want to focus on the detailed behavior of pumps. By considering `Pump-Status`, we enable the views shown in Figure 36. The first view, `working-pump`, is active when a pump has a positive flow-rate, and the pressure at the destination is greater

Figure 35: A model of pumped flow

```
(defprocess (Pumped-Flow ?pump)
  Individuals ((?pump :type Fluid-Pump
                      :conditions (Primed ?pump ?st)
                      (Pump-Connection ?pump ?src ?dst))
               (?st :type Phase)
               (?sub :type Substance))
  Preconditions ((On ?pump))
  Influences ((I+ (Amount-of-in ?sub ?st ?dst) (A (flow-rate ?pump)))
              (I- (Amount-of-in ?sub ?st ?src) (A (flow-rate ?pump)))))
```

Figure 36: Different states of pumps

```
(defview (Working-Pump ?pump)
  Individuals ((?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump))
               (?pump :type fluid-pump
                      :conditions (Consider (Pump-Status ?pump)) (Pump-Connection ?pump ?src ?dst)
                      (Pressure-Definer ?pump ?src ?pr-src) (Pressure-Definer ?pump ?dst ?pr-dst)))
  QuantityConditions ((Active ?pf)
                      (greater-than (A (flow-rate ?pump)) ZERO)
                      (greater-than (A (pressure ?pr-dst :ABSOLUTE)) (A (pressure ?pr-src :ABSOLUTE)))))
(defview (Coasting-Pump ?pump)
  Individuals ((?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump))
               (?pump :type Fluid-Pump
                      :conditions (Pump-Connection ?pump ?src ?dst) (Consider (Pump-Status ?pump))
                      (Pressure-Definer ?pump ?src ?pr-src) (Pressure-Definer ?pump ?dst ?pr-dst)))
  QuantityConditions ((Active ?pf)
                      (less-than (A (pressure ?pr-dst :ABSOLUTE)) (A (pressure ?pr-src :ABSOLUTE)))))
(defview (Losing-Pump ?pump)
  Individuals ((?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump))
               (?pump :type Fluid-Pump
                      :conditions (Consider (Pump-Status ?pump)) (Consider (Pump-Lossage ?pump))))
  QuantityConditions ((Active ?pf)
                      (less-than (A (flow-rate ?pump)) ZERO)))
(defview (Cavitating-Pump ?pump)
  Individuals ((?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump)(?pf SUB ?sub)(?pf ST liquid))
               (?pump :type Fluid-Pump
                      :conditions (Consider (Pump-Cavitation ?pump)) (Pump-Connection ?pump ?src ?dst))
               (?src-cl :type Contained-Liquid
                        :form (C-S ?sub liquid ?src)))
  QuantityConditions ((Active ?pf)
                      (greater-than (A (flow-rate ?pump)) ZERO)
                      (not (less-than (A (temperature ?src-cl :ABSOLUTE))
                                      (A (temperature (BOIL ?src-cl) :ABSOLUTE)))))
  Relations ((Unsafe-Condition ?pump)))
```

than the pressure at the source of the pump—as specified by the `pressure-definer` predicate. Similarly, `Coasting-Pump` is active when the opposite pressure relation holds. In this case, the flow rate of the pump can only be positive, so that constraint is not imposed. If we are considering `Pump-Lossage` in addition to `Pump-Status`, then the view `Losing-Pump` will be active when the flow-rate of the pump is less than zero.

The last view in Figure 36, `cavitating-pump`, is active when a pump is pumping liquid with a positive flow rate and has boiling occurring in its inlet—that is, when the temperature of the contained-liquid in the source container is at or above the boiling point. This view results in an `unsafe-condition` in the pump, since cavitation can lead to catastrophic pump failure.

As noted above, we do not model the pressure and volume relationships within the model in enough detail to allow cavitation to be accurately signaled. In particular, cavitation depends on the existence and size of tiny cavities in the fluid, and occurs when the stagnation pressure is roughly that of the liquid's vapor pressure. A more reasonable macroscopic model could be organized by representing the cavitation number, an estimate of the probability of cavitation. The cavitation number is defined as

$$\frac{\rho\nu^2}{p - p_s} \approx 1$$

where ρ is density, ν is stream velocity, p is stream pressure, and p_s is saturation pressure [18]. However, we have not yet explored the consequences of adding this construct.

4.4 Phase Transition Processes

Many thermodynamic cycles involve phase changes. For example, most air conditioners and power plants involve the boiling or evaporation of liquids and the condensation of gasses. Developing realistic qualitative models of phase transitions is complicated. Qualitative models often involve analytic approximations for a phenomena, and it is important to characterize the circumstances under which the approximation is valid. There is no single quantitative model which completely covers either boiling or condensation. Boiling occurs in several distinct regimes, such as nucleate boiling versus film boiling, each of which can be further characterized (e.g., subcooled versus saturated nucleate boiling, or stable versus unstable film boiling) [18]. While these distinctions are important for many numerical prediction tasks, for our purposes it suffices to develop a simpler model which captures just the common features of the phenomena.

What are the central phenomena we must capture? Examining what we know about simple cases provides a useful focus. Consider some stuff in a container. Its phase is determined by the relationship between its temperature and two limit points – its *boiling point* and its *freezing point*. Since we are only concerned with fluid systems, we currently ignore the freezing point in this model[14]. If the stuff is a liquid, then when its temperature

[14]The freezing point should be included as an explicit limit point even if freezing were not modeled in detail as an important reality check on analyses. A numerical model which claims that the water being pumped from a steam plant condenser is $-10°F$, for instance, is seriously buggy.

rises to the boiling point boiling occurs. When it is a gas and its temperature drops to the boiling point condensation occurs.

We now know what processes we must model and something about the conditions under which they occur. What else must we know? An important fact is that the boiling point of a substance is not constant but increases with pressure. For example, boiling (and condensation) occur at a higher temperature in a pressurized vessel (such as a car's radiator or a pressure cooker) than in an open pan on a stove. Likewise, lukewarm water will boil in a vacuum, and superheated steam will condense when subjected to sufficiently high pressure. A qualitative proportionality suffices to model this fact. But where in the model should it be installed? If we were always considering phase changes, the natural place to include this fact is in the `Complex-Physob` description. By now the alert reader suspects that a more subtle representation is used instead, and this suspicion is correct.

Boiling and condensation are in nearly all respects symmetric processes, so we refer primarily to boiling in our discussion of phase transitions and describe condensation by highlighting the few ways in which it is different.

4.4.1 Thermal Behavior of Phase Transitions

Having a temperature equal to its boiling point is not sufficient for a liquid to boil – heat must be continually added to carry out the phase transformation. Since the internal energy of the water does not rise, its temperature remains roughly constant during boiling. If the heat flow is halted, boiling stops almost immediately. The amount of heat required to boil a unit measure of liquid, once it is heated to its boiling point, is known as the *latent heat of vaporization.*

Although our model of boiling is in terms of qualitative equations relating continuous parameters, to justify the model it is useful to conceptually decompose the boiling process into an equivalent sequence of simple events. For example, boiling may be decomposed in the following way:

1. An infinitesimal piece of liquid is selected as the next candidate to undergo the transition from liquid to gas. This infinitesimal piece of liquid is removed from the contained-liquid by subtracting out its mass and heat content from the corresponding properties of the contained-liquid.

2. To convert the piece of liquid into a piece of gas, additional heat corresponding to its latent heat of vaporization is transferred to the piece of liquid.

3. As the phase transition proceeds, the piece-of-stuff expands, thereby expending energy (heat) as it does work on its surrounding contained-gas.

4. Finally, the piece of gas is added to the contained-gas by incrementing its mass, heat and volume.

Notice that the internal energy of the stuff is conserved. Where does the latent heat of vaporization come from? There are several options. First, we could require an external

heat source, whose temperature is above the boiling point. The rate of boiling is then qualitatively proportional to the heat flow rate into the liquid.

This model captures several of our important intuitions about boiling, but has certain limitations. One problem concerns multiple heat flows into and out of the liquid. This model of boiling requires a net positive flow of heat into the liquid, but our model does not currently provide such a quantity[15]. Even if this quantity were available, it would be incorrect to define the boiling rate solely in terms of the net heat flow. In fact one can boil a liquid without adding any heat at all, simply by reducing the pressure and thus reducing the boiling point below the current temperature of the liquid.

This leads to the second model: Allow the latent heat of vaporization to flow from the boiling liquid itself. This model captures vaccum boiling, but introduces a new problem – what determines the rate at which it proceeds? There is no net heat flow rate to constrain it, unlike the first model. An analogy with liquid overflow suggests an answer. In liquid overflow, the idea that the level of the liquid was never higher than the top of the container is seen as an idealization. The reality is that for overflow to occur, the level must exceed the top height of the container, and the height difference determines the rate of overflow. Similarly, we can consider the rate of boiling to be qualitatively proportional to the degree to which the liquid's temperature exceeds its boiling temperature, if we realize that, like overflow, the temperature of a boiling liquid equalling its boiling point is actually an idealization.

At first this model may seem somewhat unintuitive. But, if you consider what happens when you remove the air from a flask containing water, you will notice that the boiling occurs faster when the flask pressure is lower (i.e., when the boiling point/temperature difference is greater). Thinking about a piece-of-stuff perspective provides further support. A boiling liquid does not actually have a single temperature; rather it has a distribution of temperatures which has some particular mean that we call "the" temperature. Dropping to the molecular level, this means some molecules will be moving faster than others. If we view the boiling process as Maxwell's demon grabbing and removing only the fastest molecules, then clearly the average temperature of the liquid is reduced as a result.

The first model may be viewed as a *time-scale abstraction* ([14]) of the second model, just as our fluid flow model abstracts away any inertial effects of the fluid in the path. One drawback of the second model is that it allows boiling to occur even when there is no heat flow into the liquid *and* the boiling point is constant. While this phenomenon may actually occur, it is of such short duration that we would prefer not to include it in our model. The removal of latent heat from a boiling liquid should be sufficiently high to prevent the liquid from heating up more than infinitesimally above the boiling point. Without order of magnitude reasoning, however, we have no way to express this constraint.

Each of these models for boiling provides different advantages, so the domain model includes them both. The second model is predicated on the assumption (`Consider`

[15]The current QPE modeling language does not implement a primitive for taking sums over explicit domain-specific sets, which is necessary to define a `net-heat-flow` parameter. Overcoming this limitation appears to be straightforward, but we have not had time to implement it yet.

Complex-Boiling); otherwise the first model is used. The next section details the implementation of boiling.

4.4.2 Core of the Boiling Model

The conditions under which boiling is modeled are defined in Figure 37. The liquid-might-boil introduces the boiling temperature ((temperature (boil ?cl) :ABSOLUTE)) and enforces some consequences of related modeling assumptions. In particular, when the boiling point is allowed to vary (the (Consider Variable-Boiling-Point) assumption) it is made qualitatively proportional to the pressure in the container. Otherwise the boiling point remains constant. When complex-boiling is not considered, the temperature of the contained-liquid is constrained to never exceed its boiling point.

The other two perspectives provide the conditions under which (Boiling-Allowed-In ?can) holds, which is used to predicate instances of boiling. Both require that Boiling-in be considered for that container, as well as considering gasses globally[16]. In addition to being predicated on the distinct possibilities for the Complex-Boiling assumption, the Simple-Boiling-Allowed-In perspective requires a heat flow whose destination is the contained liquid, while the Complex-Boiling-Allowed-In perspective does not. Making Boiling-Allowed-In a closed predicate ensures that if we do not know what we should think about boiling in a container, we ignore the possibility.

The core of the boiling process, shown in Figure 38, is simple. If one is considering boiling for some container and there is a liquid in it, boiling occurs when the liquid's temperature is not less than its boiling point. The transfer of fluid from one phase to another is captured by the direct influences on amount-of-in. As usual, no constraints are placed on generation-rate in the core process since they depend on which model of boiling is being used.

Figure 39 defines the boiling rate constraints for each model. The Simple-Boiling-Rate perspective pegs it to the heat-flow-rate of the heat flowing into the liquid. It further constrains the generation rate to be zero when the heat flow rate is zero. Notice that this constraint implicitly assumes that only a single heat flow has the liquid as its destination: Otherwise, one flow might drop to zero while another was still positive, which would contradict this correspondence (and hence this perspective). We also make the temperature of the contained-liquid qualitatively proportional to its boiling point. This allows the temperature of the liquid to follow the boiling point up or down as the pressure in the container changes. The other perspective, Complex-Boiling-Rate, defines the generation rate as the product of the mass of the liquid and the difference between the liquid's temperature and its boiling point.

Recall that when a piece of liquid boils it carries heat as well as mass into the gas. This heat may be decomposed into two sources: the heat which existed in the liquid before it boiled, and the latent heat of vaporization which was required to boil the liquid. Our model for boiling implements two boiling heat flow processes—one for each source. These

[16]It would be more modular to encode this dependence as (Consider (Gas-in ?can))!

Figure 37: Establishing when boiling can occur

```
(defperspective (liquid-might-boil ?cl)
  Individuals ((?cl :type contained-liquid
                    :form (C-S ?sub liquid ?can))
               (?can :conditions (Consider (Boiling-in ?can))))
  Relations ((only-during (Positive-Quantity (temperature (Boil ?cl) :ABSOLUTE)))
             (equal-to (D (Volume ?cl)) (D (Mass ?cl))) ;; needed for ratio code...
             (When (Consider Variable-boiling-point)
               (Qprop (temperature (Boil ?cl) :ABSOLUTE) (pressure (gas-in ?can) :ABSOLUTE)))
             (When (not (Consider Complex-Boiling))
               (not (greater-than (A (temperature ?cl :ABSOLUTE))
                                  (A (temperature (boil ?cl) :ABSOLUTE)))))))

(defperspective (Simple-Boiling-Allowed-In ?can)
  Individuals ((?hf :type (process-instance heat-flow)
                    :conditions (active ?hf)(?hf DST (C-S ?sub liquid ?can)))
               (?cl :type contained-liquid
                    :form (C-S ?sub liquid ?can))
               (?can :conditions (Consider Gas) (Consider (Boiling-in ?can))
                     (not (Consider Complex-Boiling))))
  Relations ((Boiling-Allowed-In ?can)))

(defperspective (Complex-Boiling-Allowed-In ?can)
  Individuals ((?can :type container
                     :conditions (Consider Gas) (Consider (Boiling-in ?can))
                     (Consider Complex-Boiling)))
  Relations ((Boiling-Allowed-In ?can)))

(defClosed-Predicate Boiling-Allowed-In)
```

Figure 38: Core of boiling process

```
(defQuantity-Type Generation-Rate Individual)

(defprocess (Boiling ?CL)
 Individuals ((?CL :type Contained-Liquid
                   :form (C-S ?sub liquid ?can))
              (?can :conditions (Boiling-Allowed-in ?can)))
 QuantityConditions ((not (less-than (A (temperature ?CL :ABSOLUTE))
                                     (A (temperature (boil ?CL) :ABSOLUTE)))))
 Relations ((Quantity generation-rate))
 Influences ((I- (amount-of-in ?sub liquid ?can) (A generation-rate))
             (I+ (amount-of-in ?sub gas ?can) (A generation-rate))))
```

Figure 39: Defining the rate of the boiling process

```
(defperspective (Simple-Boiling-Rate ?bp ?hf)
  Individuals ((?bp :type (Process-Instance Boiling)
                    :conditions (Active ?bp) (?bp CL ?cl)
                    (not (Consider Complex-Boiling)))
               (?hf :type (Process-Instance Heat-Flow)
                    :conditions (Active ?hf) (?hf DST ?cl)))
  Relations ((Qprop (generation-rate ?bp) (heat-flow-rate ?hf))
             ;; Assumes there will only be one of these!!
             (Ordered-Correspondence ((A (generation-rate ?bp)) ZERO)
                                     ((A (heat-flow-rate ?hf))  ZERO))
             ; Keep it boiling in a rising pressure:
             (Qprop (temperature ?CL :ABSOLUTE) (temperature (boil ?CL) :ABSOLUTE))))

(defperspective (complex-boiling-rate ?bp)
  Individuals ((?bp :type (process-instance boiling)
                    :conditions (active ?bp)(?bp CL ?cl)
                    (Consider Complex-Boiling)))
  Relations ((Quantity (temperature ?cl (boil ?cl)))
             (Q= (temperature ?cl (boil ?cl))
                 (- (temperature ?cl :ABSOLUTE)
                    (temperature (boil ?cl) :ABSOLUTE)))
             (Q= (Generation-Rate ?bp)
                 (*0+ (temperature ?cl (boil ?cl)) (mass ?CL)))))
```

are given in Figure 40. The boiling-heat-flow process accounts for the heat transfer due to the transfer of fluid from the liquid to the gas. This process will only be active when we are considering Thermal-Boiling. The heat-flow-rate of the process is defined as the product of the generation-rate of the boiling process times the temperature of the boiling liquid. This influence on heat—together with the influence of the generation-rate on mass—will result in a zero net influence on the temperature of the liquid.

The latent heat of vaporization must be added to a piece of liquid as it boils, and is assumed to flow from the contained-liquid to the contained-gas. The second process in Figure 40 implements this flow of latent heat of vaporization. Boiling-Latent-Heat-Flow is active when we are considering latent-heat-of-vaporization, and provides a second influence on the heat of the liquid and the gas. The heat-flow-rate of this process is made qualitatively equal to the generation-rate of the boiling process.

Recall that for the simple model of boiling, the generation-rate is equal to the heat-flow-rate of the heat-flow process which is driving the boiling. Thus for simple boiling, these two influences on the heat of the liquid cancel each other, leaving only the influence of the boiling-heat-flow process described above. In the case of complex-boiling, the heat of the contained-liquid is negatively influenced both by the removal of liquid and by the drain caused by the latent heat of vaporization, so the net influence on the liquid's temperature is negative. This provides a stabilizing influence on the boiling liquid by pushing its temperature back below the boiling point.

Figure 40: Thermal effects of boiling

```
(defprocess (boiling-heat-flow ?bp)
  Individuals ((?bp :type (process-instance boiling)
                    :conditions (active ?bp)
                    (?bp CL (C-S ?sub liquid ?can))
                    (Consider thermal-boiling))
               (?cl :type Contained-Liquid
                    :form (C-S ?sub liquid ?can))
               (?cg :bind (C-S ?sub gas ?can)))
  Relations ((Quantity Heat-Flow-Rate)
             (Q= Heat-Flow-Rate (*0+ (Generation-Rate ?bp)
                                     (temperature ?cl :ABSOLUTE))))
  Influences ((I- (heat ?cl) (A Heat-Flow-Rate))
              (I+ (heat ?cg) (A Heat-Flow-Rate))))
(defprocess (boiling-latent-heat-flow ?bp)
  Individuals ((?bp :type (process-instance boiling)
                    :conditions (active ?bp)
                    (?bp CL (C-S ?sub liquid ?can))
                    (Consider latent-heat-of-vaporization))
               (?cl :type Contained-Liquid
                    :form (C-S ?sub liquid ?can))
               (?cg :bind (C-S ?sub gas ?can)))
  Relations ((Quantity Heat-Flow-Rate)
             (Q= Heat-Flow-Rate (Generation-Rate ?bp)))
  Influences ((I- (heat ?cl) (A Heat-Flow-Rate))
              (I+ (heat ?cg) (A Heat-Flow-Rate))))
```

4.4.3 Condensation

Condensation is defined by direct analogy with boiling. There is `condensing` process for a contained gas whose temperature is at or below the boiling point. When the process is active it has a `generation-rate` which acts to decrease the amount of gas and to increase the amount of liquid in the container. As with boiling, the rate is defined differently depending on whether one considers complex versus simple condensing. We actually use the same consider assumption (`Complex-Boiling`) to ensure that we treat phase changes symmetrically.

Figure 41 defines three perspectives which together establish whether condensing is allowed in a particular container. These are exactly analogous to the perspectives in Figure 37, but involving `Condensing-In` instead of `Boiling-In`. Similarly, Figure 42 describes the core `Condensing` process, Figure 43 defines the constraints on the rate of condensation (i.e., the `generation-rate`, this time of liquid instead of gas), and Figure 44 defines the thermal effects of condensing.

4.5 Controlling the Model

4.5.1 Representing and enforcing the steady-state assumption

As models and scenarios become increasingly complex, it becomes more costly to generate total envisionments (e.g., all possible behaviors). Often one is only interested in a particu

Figure 41: Establishing when condensing might occur

```
(defperspective (Gas-Might-Condense ?cg)
  Individuals ((?cg :type Contained-Gas
                    :form (C-S ?sub gas ?can))
               (?can :conditions (Consider (Condensing-in ?can))))
  Relations ((only-during (Quantity (temperature (condense ?cg) :ABSOLUTE)))
             (greater-than (A (temperature (condense ?cg) :ABSOLUTE)) ZERO)
             (When (Consider Variable-boiling-point)
               (Qprop (temperature (condense ?cg) :ABSOLUTE)
                      (pressure ?cg :ABSOLUTE)))
             (When (not (Consider Complex-Boiling))
               (not (less-than (A (temperature ?cg :ABSOLUTE))
                               (A (temperature (condense ?cg) :ABSOLUTE)))))))

(defperspective (Simple-Condensing-Allowed-In ?can)
  Individuals ((?hf :type (process-instance heat-flow)
                    :conditions (active ?hf)(?hf SRC (C-S ?sub gas ?can)))
               (?cl :type contained-gas
                    :form (C-S ?sub gas ?can))
               (?can :conditions (Consider Liquid) (Consider (Condensing-in ?can))
                     (not (Consider Complex-Boiling))))
  Relations ((Condensing-Allowed-In ?can)))

(defperspective (Complex-Condensing-Allowed-In ?can)
  Individuals ((?can :type container
                     :conditions (Consider Gas) (Consider (Condensing-in ?can))
                     (Consider Complex-Boiling)))
  Relations ((Condensing-Allowed-In ?can)))

(defClosed-Predicate Condensing-Allowed-In)
```

Figure 42: Condensation process

```
(defQuantity-Type Generation-Rate Individual)

(defprocess (Condensing ?CG)
 Individuals ((?CG :type Contained-Gas
                   :form (C-S ?sub gas ?can))
              (?can :conditions (Condensing-Allowed-in ?can)))
 QuantityConditions ((not (greater-than (A (temperature ?CG :ABSOLUTE))
                                        (A (temperature (condense ?CG) :ABSOLUTE)))))
 Relations ((Quantity Generation-Rate))
 Influences ((I- (Amount-of-in ?sub gas ?can) (A Generation-Rate))
             (I+ (Amount-of-in ?sub liquid ?can) (A Generation-Rate))))
```

Figure 43: Rate of condensation

```
(defperspective (simple-condensing-rate ?cp ?hf)
  Individuals ((?cp :type (process-instance condensing)
                    :conditions (active ?cp) (?cp CG ?cg)
                    (not (Consider Complex-boiling)))
               (?hf :type (process-instance heat-flow)
                    :conditions (active ?hf) (?hf SRC ?cg)))
  Relations ((Qprop (Generation-Rate ?cp) (Heat-Flow-Rate ?hf))
             ;; Assumes there will only be one of these!!
             (Ordered-Correspondence ((A (Generation-Rate ?cp)) ZERO)
                                     ((A (Heat-Flow-Rate ?hf))  ZERO))
             ; Keep it condensing in a falling pressure:
             (Qprop (temperature ?cg :ABSOLUTE) (temperature (condense ?cg) :ABSOLUTE))))
(defperspective (complex-condensing-rate ?cp)
  Individuals ((?cp :type (process-instance condensing)
                    :conditions (active ?cp)(?cp CG ?cg)
                    (Consider Complex-boiling)))
  Relations ((Quantity (temperature (condense ?cg) ?cg))
             (Q= (temperature (condense ?cg) ?cg)
                 (- (temperature (condense ?cg) :ABSOLUTE)
                    (temperature ?cg :ABSOLUTE)))
             (Q= (Generation-Rate ?cp)
                 (*0+ (temperature (condense ?cg) ?cg) (mass ?CG)))))
```

Figure 44: Heat flow in condensation

```
(defprocess (condensing-heat-flow ?cp)
  Individuals ((?cp :type (process-instance condensing)
                    :conditions (active ?cp)
                    (?cp CG (C-S ?sub gas ?can))
                    (Consider thermal-boiling))
               (?cg :type Contained-Gas
                    :form (C-S ?sub gas ?can))
               (?cl :bind (C-S ?sub liquid ?can)))
  Relations ((Quantity Heat-Flow-Rate)
             (Q= Heat-Flow-Rate (*0+ (Generation-Rate ?cp)
                                     (temperature ?cg :ABSOLUTE))))
  Influences ((I- (heat ?cg) (A Heat-Flow-Rate))
              (I+ (heat ?cl) (A Heat-Flow-Rate))))
```

Figure 45: The logic of steady-state

```
(defperspective (Steady-State-Quantity ?qty)
  Individuals ((?qty :type Quantity
                     :conditions (Consider Steady-State))))
(defperspective (Steady-State-Quantity (?qty-type ?ind . ?rest))
  Individuals ((?ind :conditions (Consider (Steady-State-Individual ?ind)))
               (?qty-type :conditions (Quantity (?qty-type ?ind . ?rest)))))
(defperspective (Steady-State-Quantity (?qty-type . ?inds))
  Individuals ((?qty-type :conditions (Consider (Steady-State-Quantity-Type ?qty-type))
                          (Quantity (?qty-type . ?inds)))))
(defperspective (Steady-State-Quantity ?qty)
  Individuals ((?qty :type Quantity
                     :conditions (Consider (Steady-State-Quantity ?qty)))))
(defpredicate Steady-State-Quantity
  (Equal-to (D ?self) ZERO))
```

lar behavior or class of behaviors. One common simplifying assumption in engineering problem solving is the steady state assumption. That is, all state variables have achieved an equilibrium, and whatever is happening in the system is occurring continuously, without transitions. This section introduces a variety of steady state assumptions, ranging in scope from a single quantity to an entire system.

Figure 45 shows how several different levels of assumptions about steady-state can be encoded uniformly. In all cases, the predicate `Steady-State-Quantity` is used to enforce the result of these assumptions, pinning the derivative of the quantity to zero. The four perspectives each correspond to a different level of specificity. The first triggers on the global assumption `Steady-State`, and the second triggers on assuming steady-state for a particular individual. The third enforces steady-state on particular classes of quantities, while the fourth enforces it for a particular, given quantity.

4.5.2 Representing Nominal Values

Monitoring a system often involves tracking whether or not certain parameters are within specific tolerances. Figure 46 defines a simple model of tolerances. The modeling assumption `(Tolerances ?qty)` indicates that the tolerances on quantity `?qty` should be monitored. The `Bounded` perspective introduces two new quantities which serve as the upper and lower bounds (i.e., the `lower-limit` and `upper-limit` of `?qty`). Notice that, with the exception of the obvious constraint that the upper limit is greater than the lower limit, we have not constrained these limits at all. Typically, a model for a specific scenario would include extra inequalities to tie this generic concept to something more specific (i.e., $\pm 10\%$). The views `Under-Tolerance` and `Over-Tolerance` make these quantities into limit points by relying on them as quantity conditions. As usual, this means that states will be distinguished according to whether or not a parameter is over, under, or within its tolerances. We further stipulate that the relation `Alarm` indicates the existence of some

Figure 46: Representing tolerances

```
(defQuantity-Type lower-limit individual)
(defQuantity-Type upper-limit individual)

(defperspective (Bounded ?qty)
  Individuals ((?qty :type Quantity
                     :conditions (Consider (Tolerances ?qty))))
  Relations ((Quantity (lower-limit ?qty))
             (Quantity (upper-limit ?qty))
             (greater-than (A (upper-limit ?qty)) (A (lower-limit ?qty)))))

(defview (Under-Tolerance ?qty)
  Individuals ((?qty :type Quantity
                     :conditions (Consider (Tolerances ?qty))))
  QuantityConditions ((less-than (A ?qty) (A (Lower-Limit ?qty))))
  Relations ((Alarm (Under-Tolerance ?qty))))

(defview (Over-Tolerance ?qty)
  Individuals ((?qty :type Quantity
                     :conditions (Consider (Tolerances ?qty))))
  QuantityConditions ((greater-than (A ?qty) (A (Upper-Limit ?qty))))
  Relations ((Alarm (Over-Tolerance ?qty))))
```

"interesting" condition. By including `Alarm` statements in the `Relations` field, we indicate explicitly what quantities are out of tolerance in a state.

4.5.3 Enforcing Relationships Between Modeling Assumptions

The previous sections used modeling assumptions to allow alternate model fragments to peacefully coexist. This section discusses how relationships between modeling assumptions are enforced, which of course is essential to ensuring that only coherent scenario models are constructed. These relationships take two forms. First, global assumptions about properties of a system entail choices for assumptions about the parts of that system. Second, certain combinations of modeling assumptions are mutually incompatible. We describe each kind in turn.

We must distinguish two types of modeling assumptions: *global assumptions* and *local assumptions*. Global assumptions apply universally to the entire scenario[17], while local assumptions refer to a particular object or subpart of the system. The entailments of global modeling assumptions can be captured in part by propagation. For example, when globally considering `Geometric-Properties`, we want them to be locally considered for every `Container`. We express this by asserting:

```
(Propagate-Consideration Geometric-Properties Container)
```

The two perspectives in Figure 47 implement the global-to-local propagation of modeling assumptions. The `Globally-Consider` perspective looks for a `Propagate-Consideration`

[17] In [4] global assumptions are represented as predicates on the distinguished symbol :SCENARIO, rather than as abstract tokens. We plan to convert the FSThermo model to this convention soon.

Figure 47: Inheriting modeling assumptions

```
(defperspective (Globally-Consider ?cnsdr ?type ?obj)
  Individuals ((?cnsdr :conditions (Propagate-Consideration ?cnsdr ?type) (Consider ?cnsdr))
               (?obj :type ?type))
  Relations ((Consider (?cnsdr ?obj))))
(defperspective (Never-Consider ?cnsdr ?type ?obj)
  Individuals ((?cnsdr :conditions (Propagate-Consideration ?cnsdr ?type) (not (Consider ?cnsdr)))
               (?obj :type ?type))
  Relations ((not (Consider (?cnsdr ?obj)))))
;;;; Containers:
(assertq (Propagate-Consideration Geometric-Properties Container))
(assertq (Propagate-Consideration Thermal-Properties Container))
(assertq (Propagate-Consideration Overflow Container))
(assertq (Propagate-Consideration Empty-Container Container))
(assertq (Propagate-Consideration Full-Container Container))
;;;; Paths:
(assertq (Propagate-Consideration Heat-Alignment Heat-Path))
(assertq (Propagate-Consideration Thermal-Conductance Heat-Path))
(assertq (Propagate-Consideration Fluid-Conductance Fluid-Path))
(assertq (Propagate-Consideration Valves Fluid-Path))
(assertq (Propagate-Consideration Changing-Valves Fluid-Path))
;;;; Pumps:
(assertq (Propagate-Consideration Pump-Status Fluid-Pump))
(assertq (Propagate-Consideration Pump-Lossage Fluid-Pump))
(assertq (Propagate-Consideration Pumped-Flow-Variation Fluid-Pump))
(assertq (Propagate-Consideration Pump-Cavitation Fluid-Pump))
(assertq (Propagate-Consideration Pump-Switch Fluid-Pump))
```

form, the matching consider assumption, and an object of the specified type. It then justifies the modeling assumption about the object in terms of the global assumption. The `Never-Consider` perspective does just the opposite. That is, when we are globally not considering some modeling choice, then we are not allowed to consider it for particular objects. If a modeling assumption is not made globally, it can be made locally on an individual basis. In such a case, the global assumption would be neither believed true nor believed false; it would simply not be mentioned.

The assertions at the bottom of Figure 47 specify what kinds of objects the global modeling assumptions are to be propagated to. This representation provides a clean mechanism for adding new assumptions as the model continues to evolve.

Many combinations of local modeling assumptions are inconsistent. For example, one cannot consider `Portals` without considering `Geometric-Properties`. These dependencies are captured in our model through assertions of the form:

`(Requires-Consideration` ⟨*dependent*⟩ ⟨*required*⟩`)`

That is, ⟨*dependent*⟩ cannot hold unless ⟨*required*⟩ does. Figure 48 lists the current set of `Requires-Consideration` assertions.

Figure 48 also contains four perspectives which enforce the semantics of `Requires-Consideration`. The first two enforce consistency of local modeling choices. `Enforce-Consider-Dependencies` triggers on every `Requires-Consideration` statement and all

Figure 48: Perspectives for controlling modeling assumptions

```
(defperspective (Enforce-Consider-Dependencies ?c1 ?c2 ?obj)
  Individuals ((?c1 :conditions (Requires-Consideration ?c1 ?c2) (Consider (?c1 ?obj))))
  Relations ((Consider (?c2 ?obj))))
(defperspective (Enforce-Consider-Backward-Dependencies ?c1 ?c2 ?obj)
  Individuals ((?c2 :conditions (Requires-Consideration ?c1 ?c2) (not (Consider (?c2 ?obj)))))
  Relations ((not (Consider (?c1 ?obj)))))
(defperspective (Enforce-Global-Consider-Dependencies ?c1 ?c2)
  Individuals ((?c1 :conditions (Requires-Consideration ?c1 ?c2) (Consider ?c1)))
  Relations ((Consider ?c2)))
(defperspective (Enforce-Global-Consider-Backward-Dependencies ?c1 ?c2)
  Individuals ((?c2 :conditions (Requires-Consideration ?c1 ?c2) (not (Consider ?c2))))
  Relations ((not (Consider ?c1))))
;;;; Consistency relations:
(assertq (Requires-Consideration Portals Geometric-Properties))
(assertq (Requires-Consideration Geometric-Properties Gravity))
(assertq (Requires-Consideration Pump-Lossage Pumped-Flow-Variation))
(assertq (Requires-Consideration Changing-Valves Valves))
(assertq (Requires-Consideration Changing-Valves Fluid-Conductance))
```

objects for which ⟨*dependent*⟩ holds, and justifies belief in the ⟨*required*⟩ assumption for that object. `Enforce-Consider-Backward-Dependencies` works in the opposite direction, forcing ⟨*dependent*⟩ to be false when ⟨*required*⟩ is false. The last two perspectives provide the same services for global modeling assumptions.

Not all relationships between modeling assumptions can be captured by these two techniques. These miscellanous relationships are encoded in ATMoSphere rules, the problem-solving language underlying QPE. Figure 49 shows the complete set. The first rule enforces the notion that if we aren't considering valves, then all paths should be considered as aligned. The second rule provides an analogous service for pumps – if we are not considering that a pump has a switch, assume that it is always on. The last four rules simply encode the consequences of thinking about particular phases. Assuming that gas or liquid should be considered justifies asserting them as phases. Similarly, if one phase isn't considered, then we should explicitly forbid further thinking about it via asserting its negation to hold.

5 Examples

The FSThermo domain model has been used to build models for a variety of scenarios. This section describes some of these examples in detail. We show the initial description of each scenario and analyze the envisionment(s) that result under different modeling and operating assumptions.

The examples described below have been run under a variety of modeling assumptions. For the sake of brevity, we list a "standard" set of assumptions in Figure 50. For each example, we indicate only the deviations from this standard set.

Figure 49: ATMoSphere rules for modeling assumption consequences

```
;;;; Preconditions:
(adb::rule :INTERN ((not (Consider (Valves ?path))) :var ?f1)
  (adb::rjustify (Aligned ?path) (?f1) :FORCED-ALIGNMENT))
(adb::rule :INTERN ((not (Consider (Pump-Switch ?pump))) :var ?f1)
  (adb::rjustify (On ?pump) (?f1) :PUMP-FORCED-ON))
;;;; Phases
(adb:rule :INTERN ((Consider gas) :var ?f1)
  (adb:rjustify (Phase gas) (?f1) :CONSIDER-GAS))
(adb:rule :INTERN ((Consider liquid) :var ?f1)
  (adb:rjustify (Phase liquid) (?f1) :CONSIDER-LIQUID))
(adb:rule :INTERN ((not (Consider gas)) :var ?f1)
  (adb:rjustify (not (Phase gas)) (?f1) :CONSIDER-GAS))
(adb:rule :INTERN ((not (Consider liquid)) :var ?f1)
  (adb:rjustify (not (Phase liquid)) (?f1) :CONSIDER-LIQUID))
```

Figure 50: Typical settings for modeling assumptions

```
(Set-Model-Assumption (Consider Gas) :FALSE)
(Set-Model-Assumption (Consider Liquid) :TRUE)
(Set-Model-Assumption (Consider Capable-Containers) :TRUE)
(Set-Model-Assumption (Consider Changing-Existence) :TRUE)
(Set-Model-Assumption (Consider Empty-Container) :TRUE)
(Set-Model-Assumption (Consider Full-Container) :FALSE)
(Set-Model-Assumption (Consider Overflow) :FALSE)
(Set-Model-Assumption (Consider Gravity) :TRUE)
(Set-Model-Assumption (Consider Geometric-Properties) :FALSE)
(Set-Model-Assumption (Consider Thermal-Properties) :FALSE)
(Set-Model-Assumption (Consider Fluid-Conductance) :FALSE)
(Set-Model-Assumption (Consider Thermal-Conductance) :FALSE)
(Set-Model-Assumption (Consider Heat-Alignment) :FALSE)
(Set-Model-Assumption (Consider Portals) :FALSE)
(Set-Model-Assumption (Consider Valves) :FALSE)
(Set-Model-Assumption (Consider Pump-Lossage) :TRUE)
(Set-Model-Assumption (Consider Pump-Status) :TRUE)
(Set-Model-Assumption (Consider Pump-Switch) :FALSE)
(Set-Model-Assumption (Consider Pump-Cavitation) :FALSE)
(Set-Model-Assumption (Consider Pumped-Flow-Variation) :TRUE)
(Set-Model-Assumption (Consider Complex-Boiling) :FALSE)
(Set-Model-Assumption (Consider Thermal-Boiling) :FALSE)
(Set-Model-Assumption (Consider Latent-Heat-of-Vaporization) :FALSE)
```

Figure 51: A path connecting two containers

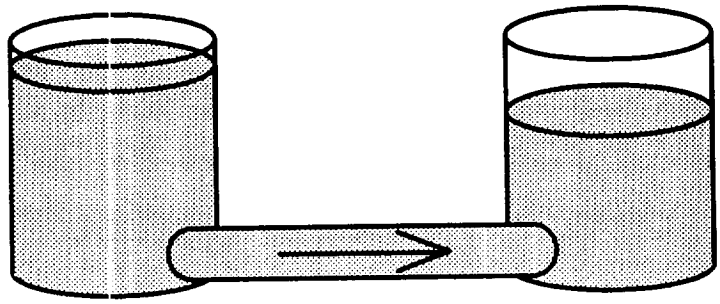


Figure 52: Scenario input for a path between two containers

```
(assertq (Substance WATER))
(assertq (Container CAN1))
(assertq (Container CAN2))
(assertq (Liquid-Path PATH1))
(assertq (Fluid-Connection PATH1 CAN1 CAN2))
```

5.1 Modeling a Simple Fluid Flow System

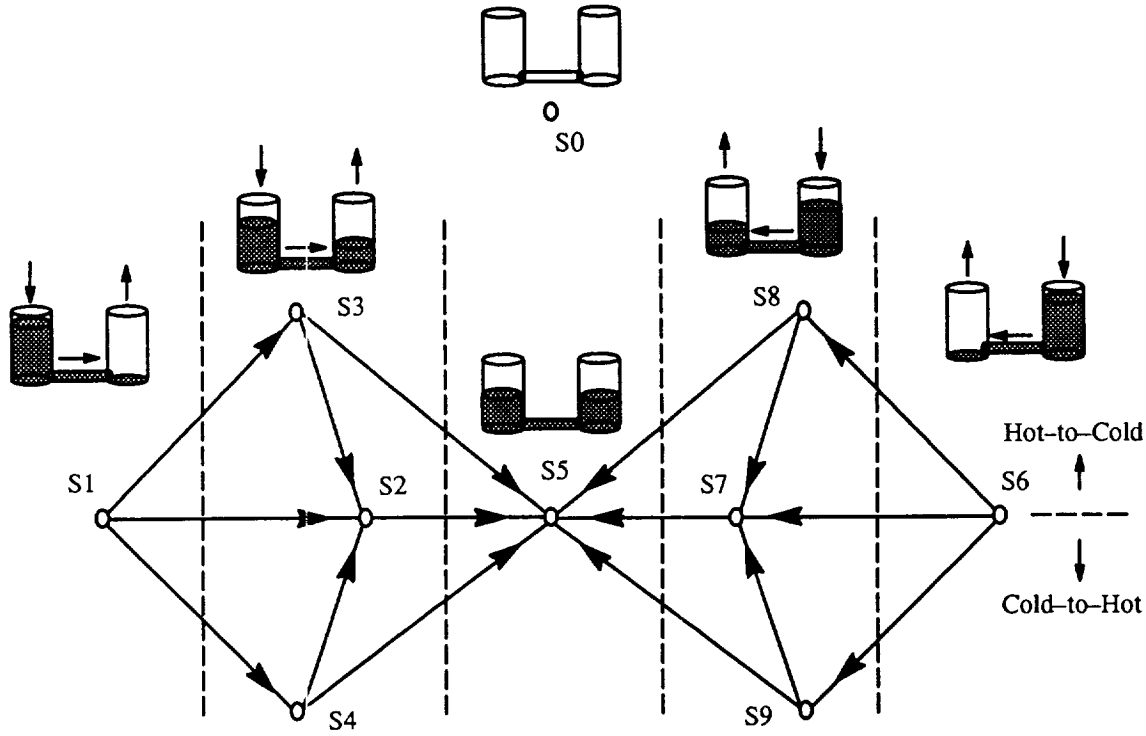
Here we describe a simple scenario consisting of two containers connected by a fluid path, as depicted in Figure 51. The corresponding scenario description is shown in Figure 52. We define a substance: `water`; two containers: `CAN1` and `CAN2`; and a fluid-path: `PATH1`. In addition, we specify that `PATH1` connects `CAN1` and `CAN2`.

We have envisioned this simple example under a variety of modeling assumptions. In particular, we have toggled the consider assumptions for `Thermal-Properties`, `Geometric-Properties`, `Portals` and `Fluid-Conductance`, both in isolation and in combination. The results for these as well as the other examples discussed below are summarized in Table 1. Run times for all examples were measured using a Symbolics 3670 running Genera 6.2.

One important feature to notice is that as additional modeling assumptions are enabled, run times increase, as do the number of quantities, inequalities, and TMS structures required to support the reasoning. Avoiding this extra complexity when it isn't needed is of course the primary reason for introducing modeling assumptions into the domain model.

Figure 53 shows the envisionment produced when thermal properties are considered, and Figure 54 shows the envisionment for the remaining cases. The other modeling as-

Figure 53: Envisionment for simple flow with thermal properties

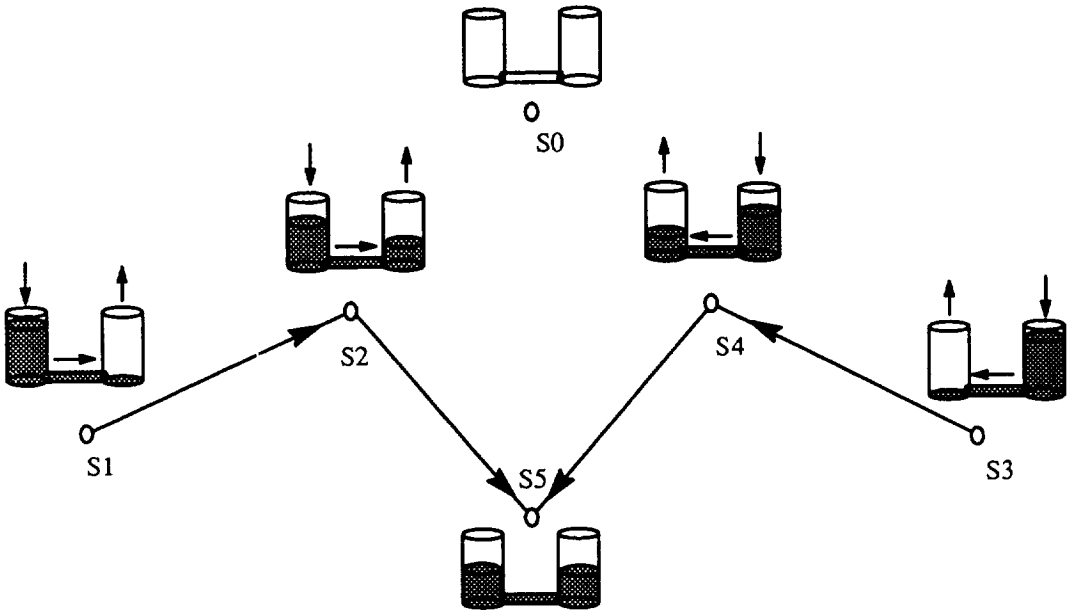


sumptions have no effect on the shape of the envisionment. Both envisionments are similar in that flow into an empty container leads instantly to flow from higher to lower contained liquid, which eventually leads to equilibrium. Considering thermal properties adds a distinction regarding the relative temperatures of the flowing liquids. The temperature of the source liquid is always constant, while the destination liquid gets hotter or colder, according to the relative temperature of the source of the flow.

If we were to consider full containers, the envisionment would include additional states in which one or both containers are full of liquid. If both containers are the same height and only one of them is full, a liquid flow out of the full container will instantly remove some of the liquid. Similarly, if one container is taller than the other and both are full, then a liquid flow from the taller container will instantly initiate an overflow in the shorter container. These behaviors are not explicitly encoded in the domain model, but are natural consequences of the inequalities which hold between levels and container heights.

Another interesting emergent behavior of the model is observed when gasses are considered in the two container liquid flow example. If there is a non-zero amount of gas in a container, then it is impossible for a liquid flow into the container to cause it to become

Figure 54: Envisionment for simple flow without thermal properties



full of liquid. Intuitively this is because the gas insists on occupying some space, so the liquid is not allowed the entire volume of the container. Again, the model did not need to anticipate this specific behavioral constraint, which falls out from the definitions of the properties of the gas.

5.2 Modeling a Pumped Flow System

Figure 55 depicts a scenario in which two containers are connected by a pump and a fluid-path in parallel. This example is worth considering because, although its structure is simple (see Figure 56), the resulting behaviors are non-trivial. Under the standard assumptions, the model runs in a reasonably short time (about two minutes), and yields a total envisionment which is small enough (11 states, 9 transitions) to be analysed in detail. At the same time, this example is non-trivial in that it involves competing processes, and includes in its possible behaviors a steady state where the competing processes exactly cancel each other.

The envisionment for this scenario is shown in Figure 57. One isolated state (S0) represents the somewhat uninteresting possibility of no liquid in either container. The three *eden states* (S1, S4 and S6) represent situations where one of the two containers is empty. S1 and S4 differ in that in S1 the pump is losing, while in S4 the pump is

Figure 55: A pump and a path connecting two containers

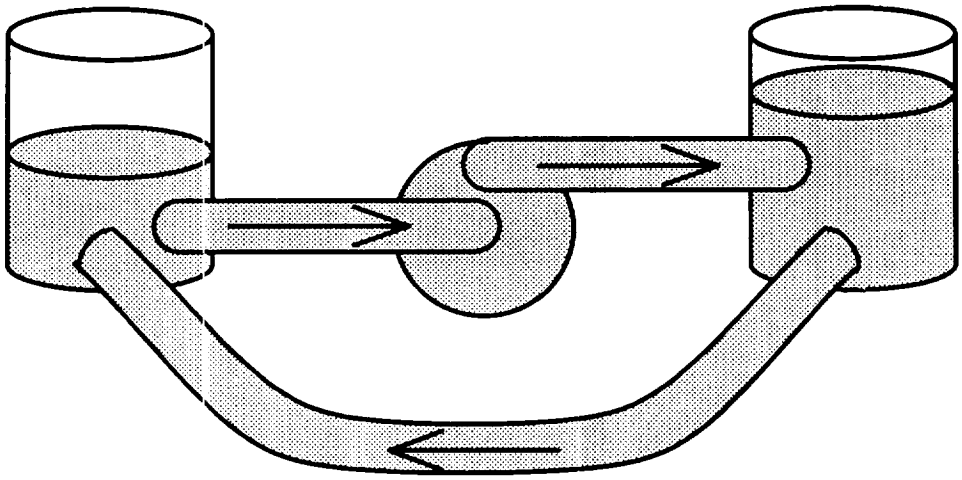


Figure 56: Scenario input for a pump and a return path between two containers

```
(assertq (Substance WATER))
(assertq (Container CAN1))
(assertq (Container CAN2))
(assertq (Liquid-Path PATH1))
(assertq (Liquid-Pump PUMP1))
(assertq (Fluid-Connection PATH1 CAN1 CAN2))
(assertq (Pump-Connection PUMP1 CAN1 CAN2))
```

simply not primed. These states last for only an instant, and then lead to intervals in which neither container is empty, and the previously empty container is filling up. If the pump was initially losing (S1, S2) it eventually reaches its no-flow pressure (S3) and then immediately begins pumping forward, but still not fast enough to keep up with the return path (S5). Gradually the two flow rates approach each other, until a state of equilibrium (S10) is achieved, where the flow rates are equal in magnitude but opposite in direction. On the other hand, if the pump is initially coasting (S6, S7) the two levels will at some point become equal for an instant (S8). This immediately leads to a state (S9) where the pump is working, but still flowing faster than the return path. This also eventually leads to equilibrium (S10).

Figure 57: Envisionment for the Pump Cycle example (without thermal properties)

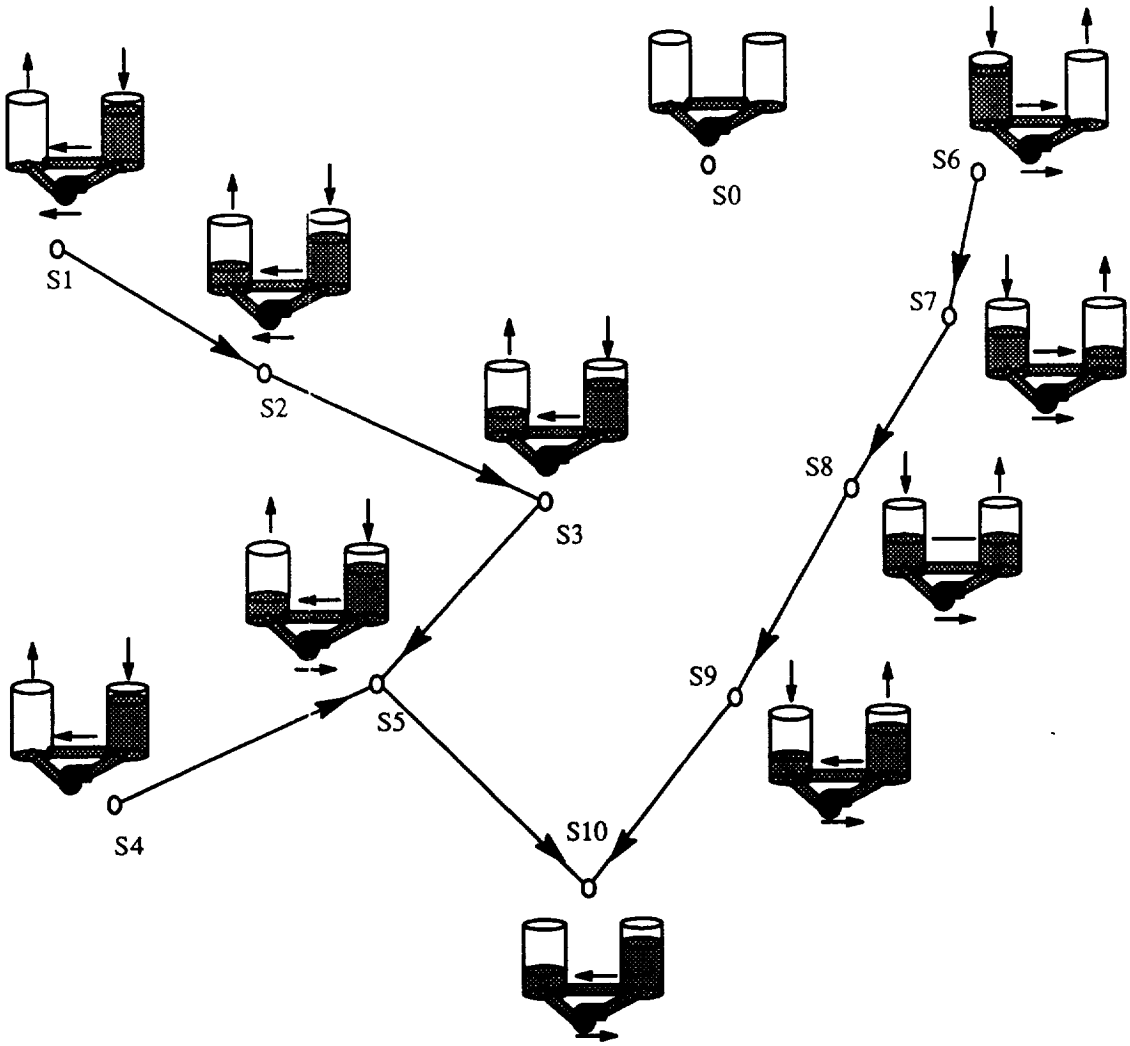
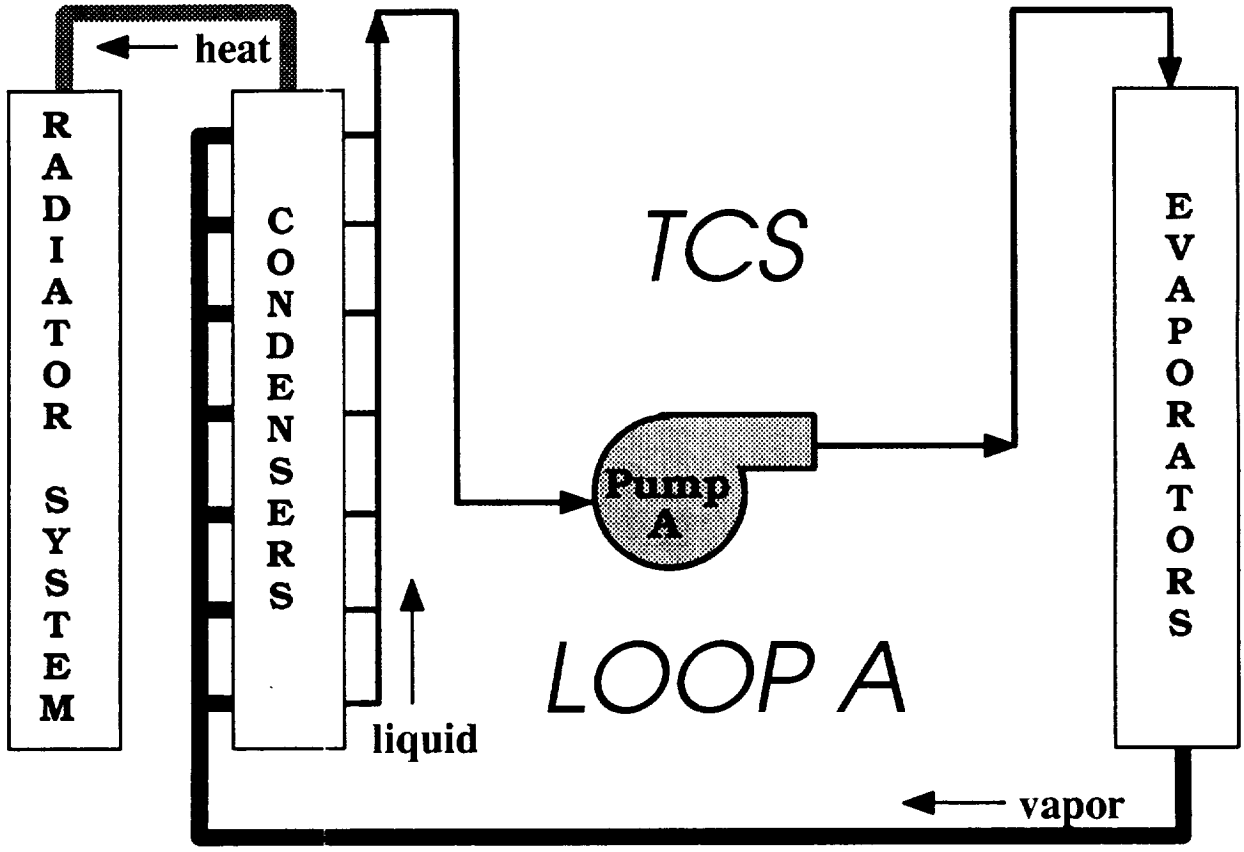


Figure 58: High-level sketch of Thermal Control System design



5.3 Modeling a Thermal Control System

A major goal driving the development of our qualitative model has been to model the thermal control system of the proposed space station Freedom. Modeling a system described at the level of engineering blueprints is not yet feasible, since it requires formalization of the techniques engineers use to compute *structural abstractions* from structural descriptions [3]. However, an extremely abstract sketch of one proposed design is shown in Figure 58. We have assembled a scenario description for the core of this system. Figure 59 shows the portion which defines Loop-A of Figure 58. We first define the working fluid: ammonia, and the containers representing the evaporator and condenser. The containers are then connected by a fluid-path and a pump. Finally, heat-sinks are placed in thermal contact with the two containers (actually, with their contents), to represent the internal environment of the space station and the external radiators, respectively.

Figure 59: Scenario Description for TCS LOOP A:

```
;; Define working fluid:
(assertq (substance NH3))
;; Define containers:
(assertq (container EVAPORATOR-A))
(assertq (container CONDENSER-A))
;; Set up expansion valve:
(assertq (Gas-Path VAPOR-PATH-A))
(assertq (Connects-to VAPOR-PATH-A EVAPORATOR-A CONDENSER-A))
;; Set up Pump:
(assertq (Liquid-Pump PUMP-A))
(assertq (Pump-Connection PUMP-A CONDENSER-A EVAPORATOR-A))
;; Set up heat-flow from Inside Space Station:
(assertq (Heat-Sink INSIDE-STATION))
(assertq (Heat-Path EVAP-SURFACE-A))
(assertq (Thermally-Connects-To EVAP-SURFACE-A INSIDE-STATION (C-S NH3 liquid EVAPORATOR-A)))
;; Set up heat-flow to RADIATOR-SYSTEM
(assertq (Heat-Sink RADIATOR-SYSTEM))
(assertq (Heat-Path COND-SURFACE-A))
(assertq (Thermally-Connects-To COND-SURFACE-A (C-S NH3 gas CONDENSER-A) RADIATOR-SYSTEM))
```

If we envision the Thermal Control System under the operating assumption of steady-state, the result is a single state in which all mass and heat flows are balanced. The liquid ammonia is pumped from the condenser into the evaporator. There it receives heat from the inside of the space-station, and begins to boil. The gaseous ammonia then flows into the condenser, where it rejects heat to the radiator. As it cools, the gaseous ammonia condenses, adding to the liquid in the condenser, and thus completing the cycle.

In this example as well as others, steady state assumptions provide the focus necessary to turn an otherwise intractable problem into a relatively simple one. Without this constraint, this example runs for many hours, generating literally thousands of qualitative states, before eventually bringing our Symbolics 3670 to a grinding halt. Work in progress on incremental envisioning techniques should provide more flexible strategies for searching the space of possible behaviors for even more complex scenarios.

5.4 Modeling a Refrigerator

Modeling a refrigerator constitutes a significant test of the domain theory, since it involves most of the defined process types. A two-phase refrigerator involves liquid and gas flow, heat flow, and phase transitions between the liquid and gaseous phases.

Figure 60 depicts the configuration for a simple two-phase refrigeration system. For simplicity, the evaporator and condenser coils have been modeled as closed-containers rather than path-type heat exchangers. The contained-liquid in the evaporator and the contained-gas in the condenser are in thermal contact with their surroundings, so that heat flows can support the respective phase transitions. A compressor moves gas from the

Figure 60: A two-phase refrigeration system

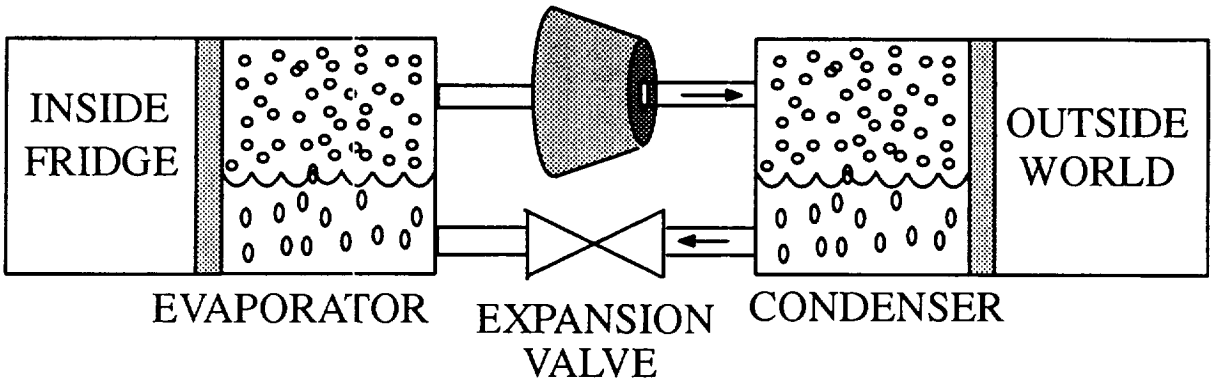


Figure 61: Scenario description for the refrigerator

```
;; Define working fluid:
(assertq (substance Freon))
;; Define containers:
(assertq (container EVAPORATOR))
(assertq (container CONDENSER))
;; Set up expansion valve:
(assertq (Liquid-Path EXPANSION-VALVE))
(assertq (Connects-To EXPANSION-VALVE CONDENSER EVAPORATOR))
;; Set up Compressor:
(assertq (Gas-Pump COMPRESSOR))
(assertq (Pump-Connection COMPRESSOR EVAPORATOR CONDENSER))
;; Set up heat-flow from Inside Fridge:
(assertq (Heat-Sink INSIDE-FRIDGE))
(assertq (Heat-Path EVAP-SURFACE))
(assertq (Thermally-Connects-To EVAP-SURFACE INSIDE-FRIDGE (C-S Freon liquid EVAPORATOR)))
;; Set up heat-flow to Room:
(assertq (Heat-Sink ROOM))
(assertq (Heat-Path COND-SURFACE))
(assertq (Thermally-Connects-To COND-SURFACE (C-S Freon gas CONDENSER) ROOM))
```

evaporator to the condenser, and a simple fluid path serves as an expansion valve, allowing liquid to return to the evaporator.

For tractability, we again used the steady-state operating assumption. The resulting envisionment consists of a single situation representing the normal operating mode of the refrigerator. The situation consists of six active process instances: a liquid flow, a compressed gas flow, two heat flows, and one of each phase transition process type. The steady-state operation of the refrigerator can be described in terms of these processes as follows:

1. The pressure in the condenser is greater than that in the evaporator, so liquid flows through the expansion valve into the evaporator.

2. The liquid immediately begins to evaporate, due to the low boiling point associated with the low pressure in the evaporator. The rate of liquid flow exactly matches the rate of evaporation, thus maintaining a constant amount of liquid in the evaporator. However, the heat carried into the evaporating liquid by the flow through the expansion valve is less than the heat taken away by the evaporated gas.

3. In order to maintain constant temperature, a heat flow process from the refrigerator interior must make up the difference. Thus the steady-state temperature of the liquid in the evaporator is lower than the inside temperature of the fridge.

4. The gas in the evaporator is compressed and moved into the condenser. The work done by the compressor raises the heat and temperature of the gas as it is compressed.

5. The gas is now hotter than room temperature, but below the higher boiling point in the high-pressure condenser. Condensation occurs.

6. As the gas condenses, it gives off heat, which flows into the room. The condensed liquid is now ready to flow through the expansion valve, thus completing the cycle.

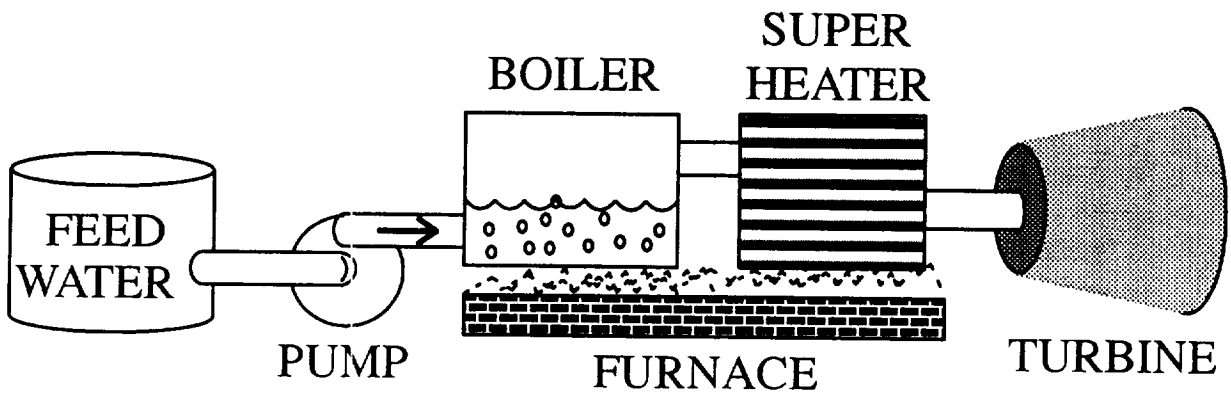
This scenario represents one of the largest models run by QPE to date. Although it created only ten view instances and eight process instances, these resulted in 332 inequality relations among 173 numbers. QPE used about ten minutes of processor time on a Symbolics 3600 to produce the highly-constrained envisionment.

5.5 Modeling a shipboard propulsion plant

Figure 62 depicts a greatly simplified model of a shipboard propulsion plant. We focus on the behavior of the fluid while in the boiler and immediately afterward. We ignore the details surrounding the turbines, instead modeling them as a simple fluid path. Our scenario description is given in Figure 63. The boiler and superheater are both modeled as simple containers[18]. The furnace is a `heat-sink` which is constrained to be hotter than both the liquid in the boiler and the gas in the superheater.

[18]Modeling heat exchangers correctly requires an alternate ontology for fluids—namely, our *Molecular Collection Ontology*, as discussed in [1].

Figure 62: A simplified shipboard propulsion plant



Because we are only interested in the behaviors in the boiler and superheater, we do not globally consider `Thermal-Properties`, but instead only consider it for those two containers. This prevents us from unnecessarily reasoning about thermal behaviors in the feedwater tank or in the environment.

Note that we cannot model the propulsion plant under the global assumption of steady state, because it does not form a cycle. Thus in order for anything interesting to be happening, some quantities must be changing. But we can make use of some of the more specific steady state assumptions. For example, we may reason about those states in which the quantities belonging to the fluid in the boiler or the superheater are constant, using the `Stead-State-Individual` consider assumption.

The constraints of partial steady state, along with the temperature relations shown in Figure 63, are sufficient to narrow the possibilities to six distinct situations, each representing a minor variation on the normal operating mode of a propulsion plant. We believe that this model could provide the basis for answering questions such as: "Given an increase in feedwater temperature, what happens to the steam temperature at the superheater outlet?"[19]

5.6 Summary of Examples

Table 1 summarizes the results of the examples described above. The table includes numbers of states, transitions, quantities and inequalities, as well as run times and internal statistics (numbers of nodes, environments and justifications). These provide some indication of the amount of inferencing going on "under the hood".

All of the examples presented here run in less than ten minutes, with the average run

[19]Understanding what happens in this situation is one of the hardest problems given at the U.S. Navy Surface Warfare Officer's School, in NewPort, R.I.

Figure 63: Scenario description for simplified propulsion plant

```
;; Define working fluid:
(assertq (substance water))

;; Define Containers:
(assertq (container FEEDWATER-TANK))
(assertq (container BOILER))
(assertq (container SUPERHEATER))
(assertq (container ENVIRONMENT))

;; Set up Pump:
(assertq (Liquid-Pump PUMP1))
(assertq (Pump-Connection PUMP1 FEEDWATER-TANK BOILER))

;; Set up fluid paths to and from Superheater:
(assertq (Gas-Path PATH1))
(assertq (Connects-To PATH1 BOILER SUPERHEATER))
(assertq (Gas-Path TURBINES))
(assertq (Connects-To TURBINES SUPERHEATER ENVIRONMENT))

;; Set up heat-flows from FURNACE
(assertq (Heat-Sink FURNACE))
(assertq (Heat-Path BOILER-SURFACE))
(assertq (Thermally-Connects-To BOILER-SURFACE FURNACE (C-S Water liquid BOILER)))
(assertq (Heat-Path SUPERHEATER-SURFACE))
(assertq (Thermally-Connects-To SUPERHEATER-SURFACE FURNACE (C-S water gas SUPERHEATER)))

;; Make furnace HOT!
(assertq (less-than (A (temperature (C-S water liquid BOILER) :absolute))
                    (A (temperature furnace :absolute))))
(assertq (less-than (A (temperature (C-S water gas superheater) :absolute))
                    (A (temperature furnace :absolute))))
```

time being just under four minutes. Without the ability to apply simplifying assumptions—primarily concerning steady state—several of the larger examples would be beyond the capabilities of current hardware technology.

The examples presented here are intended to provide some indication of the capabilities of the FSThermo model, and by no means represent an exhaustive set. Given the composability offered by QP theory, the number of specific scenarios to which the FSThermo model is applicable is limited only by the available computing resources.

Having ratios be operating

6 Discussion

This paper has presented the FSThermo domain model for engineering thermodynamics. While certainly not the last word in qualitative models for engineering thermodynamics, we believe it represents substantial progress. Furthermore, we have tried to make the motivations for major design decisions explicit, and discussed the issues involved in developing a large-scale qualitative domain model. While not a tutorial, we have tried to write down some of the "lore" that has been accumulated by our group in developing domain models. We hope that other researchers might find this useful in developing their own domain models.

Table 1:

Example	Sits	Sclasses	Trans	Quants	Ineqs	Run-Time	Nodes	Justs	Envs
2 Cans	6	6	4	21	57	78 (sec)	1087	1613	191
2 Cans w/ Portals	6	6	4	33	106	153	1743	3027	179
2 Cans w/ Thermal	12	10	16	35	89	154	1667	2566	383
2 Cans w/ Geometry	6	6	4	27	78	94	1381	2227	192
2 Cans w/ Conductance	6	6	4	27	69	152	1349	1997	553
2 Cans w/ Side-Portals	28	19	15	33	109	243	1833	3169	1446
2 Cans w/ Portals, Conductance and Thermal	12	10	16	49	142	266	2437	4077	908
Pump-Cycle	11	11	9	23	66	117	1343	2091	572
Pump-Cycle w/ Portals	11	11	9	41	151	332	2489	4946	1083
TCS (steady state)	1	1	0	68	201	299	3687	5816	86
Refrigerator	9	4	0	67	196	557	3663	6146	269
Steam Plant	9	6	0	93	273	442	4697	7608	190

While the FSThermo domain model captures a number of important phenomena in engineering thermodynamics, several extensions are needed to bring it closer to capturing the full range of the qualitative aspect of an engineer's knowledge. These include:

Geometric knowledge: Currently no processes affect geometric properties. Thus the only systems which can be described are those whose geometry is constant over time. Modeling many systems and components, such as internal combustion engines or flush toilets, requires smoothly integrating a sophisticated dynamics with geometric reasoning. Kim [11,12] is working on such an integration.

Multiple substances: If we think only about the working fluid in a power plant, the single-substance assumption is not onerous. But more often multiple substances are required. In some cases the interactions are thermal, and (assuming nothing is wrong) the substances do not mix. Examples include some lubrication systems and cooling systems. In other cases the chemical properties of the mixture are of paramount importance to the model (e.g., distillation plants). A general domain model for engineering thermodynamics must be able to model multiple substances.

As noted in Section 4.2.2, our current model has been designed with such extensions in mind. In particular, the properties of the `liquid-in` and the `gas-in` for each con-

tainer would be based on combinations of the properties of their constituents, rather than equalities or constants. Chemical interactions between constituents need to be modeled by new processes, whose individuals are restricted to being contents of the same abstract individual. We suspect that most of the non-chemical consequences of the properties of the `liquid-in` and the `gas-in` (e.g., the interaction between the pressure of the `gas-in` and the pressure of the `liquid-in`) can remain unchanged.

Head: The current model presumes that the properties of the portals at the ends of a path suffice to establish whether or not there is a flow. If both portals are at the same height this presumption is correct, but otherwise it is not, since it does not take into account the force of gravity acting on the fluid in the path itself. The current lack of internal geometric structure in fluid paths prevents us from modeling this.

In hydraulics the concept of *head* is introduced to properly account for the factors effecting flow through non-level paths. Within the perspective of contained-stuffs, head could be defined qualitatively as the sum of pressure and height. This ignores any contribution from velocity, since the liquid in the path still is not explicitly represented. (Representing the velocity of the liquid in the path would be easy with the molecular collection ontology, but problematic within the contained-stuff view.) The minimal geometric extension to fluid paths to support this definition of head would be to divide the path up into segments, and associate a height with each segment.

New ontologies: Do the contained-stuff and molecular collection ontologies span the space of entities needed for physical stuffs in engineering thermodynamics? Clearly not, as our discussion of head indicates. Finding new ways to individuate pieces of stuff, to describe fluid objects that are larger than molecular collections and not co-extensive with some externally-defined container is an interesting open problem.

7 Acknowledgements

This domain model incorporates many ideas developed in joint discussions with Dennis DeCoste, Brian Falkenhainer, and Gordon Skorstad. This research was supported in part by the National Aeronautics and Space Administration, Contract No. NASA NAG-9137.

References

[1] Collins, J. and Forbus, K. "Reasoning about Fluids via Molecular Collections", in *Proceedings of the National Conference on Artificial Intelligence*, Seattle, July, 1987.

[2] de Kleer, J. and Brown, J. "A Qualitative Physics based on Confluences", *Artificial Intelligence*, **24**, 1984.

[3] 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.

[4] Falkenhainer, B. and Forbus, K. "Compositional Modeling: Forming the relevant model for the job", *submitted for publication, 1990.*

[5] Forbus, K. "Qualitative Process Theory" *Artificial Intelligence*, **24**, 1984.

[6] Forbus, K. "The Problem of Existence", in Proceedings of the Cognitive Science Society, 1985.

[7] Forbus, K. "Introducing actions into qualitative simulation", Proceedings of IJCAI-89, August, 1989.

[8] Forbus, K. D. The Qualitative Process Engine in *Readings in Qualitative Reasoning about Physical Systems*, Weld, D. and de Kleer, J. (Eds.), Morgan Kaufmann, 1990.

[9] Hayes, P. "The Naive Physics Manifesto", in *Expert Systems in the Micro-Electronic Age*, D. Michie (Ed.), Edinburgh University Press, 1979.

[10] Hayes, P. "Naive Physics 1: Ontology for Liquids", in Hobbs, J. and Moore, B. (Eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corporation, 1985.

[11] Kim, H. "Qualitative Kinematics of Linkages", Department of Computer Science Technical Report No. UIUCDCS-R-90-1603.

[12] Kim, H. "Qualitative Reasoning about the Geometry of Fluid Flow", To appear in the Proceedings of CogSci-90, July, 1990

[13] Kuipers, B. "Common Sense Causality: Deriving Behavior from Structure", *Artificial Intelligence*, **24**, 1984.

[14] Kuipers, B. "Abstraction by Time-Scale in Qualitative Simulation", in *Proceedings of the National Conference on Artificial Intelligence*, Seattle, July, 1987.

[15] Weld, D. "Switching Between Discrete and Continuous Process Models to Predict Genetic Activity", MIT Artificial Intelligence Lab TR-793, October, 1984.

[16] Williams, B. "Qualitative Analysis of MOS Circuits", *Artificial Intelligence*, **24**, 1984.

[17] Iwasaki, Y., and Simon, H. "Causality in Device Behavior", *Artificial Intelligence*, **29**, 1986.

[18] Considine, D. M. (Ed.) *Van Nostrand's Scientific Encyclopedia*, sixth edition, Van Nostrand Reinhold Company, 1983.