[67] Mark Shirley and Randall Davis. Generating distinguishing tests based on hierarchical models and symptom information. In *Proceedings of the International Conference on Computer Design: VLSI in Computers*, pages 455–458. IEEE Computer Society Press, 1983.

[68] Reid G. Simmons and John L. Mohammed. Causal modeling of semiconductor fabrication. *Artificial Intelligence in Engineering*, 4:2–21, 1989.

[69] Narinder Singh. Saturn: An automatic test generation system for digital circuits. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 778–783, Los Altos, CA, 1986. Morgan Kaufmann.

[70] S. Slaughter, R. Hartzell, and T.-C. Chang. A deep level reasoning tool for semiconductor process simulation. In *IEEE/SEMI International Semiconductor Manufacturing Science Symposium*, page 117, Los Alamitos, CA, 1989. IEEE Computer Society Press.

[71] J. M. Smith and H. C. Van Ness. *Introduction to Chemical Engineering Thermodynamics*. McGraw-Hill, Inc., New York, third edition, 1975.

[72] G. Stephanopoulos, G. Henning, and H. Leone. MODEL.LA. a modeling language for process engineering–i. the formal framework. *Computers and Chemical Engineering*, 14(8):813–846, 1990.

[73] Peter Struss. Diagnosis as a process. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 408–418. Morgan Kaufmann, San Mateo, CA, 1992.

[74] Peter Struss. What's in SD? towards a theory of modeling for diagnosis. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 419–449. Morgan Kaufmann, San Mateo, CA, 1992.

[75] Peter Struss and Oskar Dressler. Physical negation: Introducing fault models into the general diagnostic engine. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1318–1323, San Mateo, CA, 1989. Morgan Kaufmann.

[76] V. Venkatasubramanian and S. H. Rich. An object-oriented two-tier architecture for integrating compiled and deep-level knowledge for process diagnosis. *Computers and Chemical Engineering*, 12:903–921, 1988.

[77] Daniel S. Weld. Approximation reformulations. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 407–412, Menlo Park, CA, 1990. AAAI Press/MIT Press.

[78] Daniel S. Weld. Query-directed approximations. In Boi Faltings and Peter Struss, editors, *Recent Advances in Qualitative Physics*, pages 101–116. MIT Press, Cambridge, MA, 1992.

[79] Daniel S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56:255–300, 1992.

[54] O. O. Oyeleye, F. E. Finch, and M. A. Kramer. Qualitative modeling and fault diagnosis of dynamic processes by MIDAS. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 262–275. Morgan Kaufmann, San Mateo, CA, 1992.

[55] Robert H. Perry and Cecil H. Chilton. *Chemical Engineers' Handbook*. McGraw-Hill, Inc., New York, 1973.

[56] D. R. Pitts and L. E. Sissom. *Theory and Problems of Heat Transfer*. McGraw-Hill, Inc., New York, 1977.

[57] Olivier Raiman. Order of magnitude reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 100–104, San Mateo, CA, 1988. Morgan Kaufmann.

[58] Olivier Raiman. The alibi principle. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 66–70. Morgan Kaufmann, San Mateo, CA, 1992.

[59] Olivier Raiman, Johan de Kleer, Vijay Saraswat, and Mark Shirley. Characterizing non-intermittent faults. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 849–854, Menlo Park, CA, 1991. AAAI Press/MIT Press.

[60] Shankar A. Rajamoney and San Hoe Koo. Qualitative reasoning with microscopic theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 401–406, Menlo Park, CA, 1990. AAAI Press/MIT Press.

[61] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[62] S. H. Rich and V. Venkatasubramanian. Model-based reasoning in diagnostic expert systems for chemical process plants. *Computers and Chemical Engineering*, 11:111–122, 1987.

[63] Jeff Rickel and Bruce Porter. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1191–1198, Menlo Park, CA, 1994. AAAI Press/MIT Press.

[64] Geoffrey W. Rutledge and Ross Shachter. A method for the dynamic selection of models. In P. Cheesman and R. W. Oldford, editors, *Selecting Models from Data: AI and Statistics IV*. Springer-Verlag, 1994.

[65] Sharad Saxena and Amy Unruh. Diagnosis of semiconductor manufacturing equipment and processes. *IEEE Transactions on Semiconductor Manufacturing*, 7:220–232, 1994.

[66] Ethan A. Scarl, John R. Jamieson, and Carl I. Delaune. A fault detection and isolation method applid to liquid oxygen for the space shuttle. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 414–416, Los Altos, CA, 1985. Morgan Kaufmann.

[41] M. A. Kramer. Malfunction diagnosis using quantitative models with non-boolean reasoning in expert systems. *AICHE Journal*, 33:130–140, 1987.

[42] Benjamin Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–388, 1986.

[43] Douglas B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Rpresenatation and Inference in the Cyc Project.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.

[44] Octave Levenspiel. *Chemical Reaction Engineering.* John Wiley & Sons, Inc., New York, NY, 1972.

[45] Zheng-Yang Liu. Integrating two ontologies for electronics. In Boi Faltings and Peter Struss, editors, *Recent Advances in Qualitative Physics*, pages 153–168. MIT Press, Cambridge, MA, 1992.

[46] Zheng-Yang Liu and Arthur M. Farley. Shifting ontological perspectives in reasoning about physical systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 395–400, Menlo Park, CA, 1990. AAAI Press/MIT Press.

[47] M. L. Mavrovouniotis and G. Stephanopoulos. Formal order-of-magnitude reasoning in process engineering. *Computers and Chemical Engineering*, 12:867–880, 1988.

[48] Michael Modell and Robert C. Reid. *Thermodynamics and its Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1983.

[49] John L. Mohammed. *Combining Experiential and Theoretical Knowledge in the Domain of Semiconductor Manufacturing Equipment.* PhD thesis, Stanford University, August 1994. Published by Stanford Computer Science Department as technical report STAN-CS-94-1526, also KSL-94-62.

[50] John L. Mohammed and Reid G. Simmons. Qualitative simulation of semiconductor fabrication. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 794–799, Los Altos, CA, 1986. Morgan Kaufmann.

[51] P. Pandurang Nayak. *Automated Modeling of Physical Systems.* PhD thesis, Stanford University, September 1992. Published by Stanford Computer Science Department as technical report STAN-CS-92-1443.

[52] P. Pandurang Nayak, Leo Joskowicz, and Sanjaya Addanki. Automated model section using context-dependent behaviors. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 710–716, Menlo Park, CA, 1992. AAAI Press/MIT Press.

[53] Hwee Tou Ng. Model-based, multiple fault diagnosis of time-varying, continuous physical systems. In *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, pages 9–15. IEEE, 1990.

[27] Gregory G. Freeman. *Application of Analytic Models to Computer-Aided Integrated Circuit Device Diagnosis and Parametric Test Selection.* PhD thesis, Stanford University, March 1991. Published by Stanford Electrical Engineering Department as technical report G834-2.

[28] K. Funakoshi and K. Mizuno. A rule-based VLSI process flow validation system with macroscopic process simulation. *IEEE Transactions on Semiconductor Manufacturing*, 3:239–246, 1990.

[29] Massimo Gallanti, Marco Roncato, and Alberto Stefanini. A diagnostic algorithm based on models at different levels of abstraction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1350–1355, San Mateo, CA, 1989. Morgan Kaufmann.

[30] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, New York, 1979.

[31] Michael R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1–3):411–436, December 1984.

[32] S. D. Grantham and L. H. Ungar. A first principles approach to automated troubleshooting of chemical plants. *Computers and Chemical Engineering*, 14:783–798, 1990.

[33] R. A. Greenkorn and D. P. Kessler. *Transfer Operations.* McGraw-Hill, Inc., New York, 1972.

[34] Walter C. Hamscher. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51(1–3), October 1991.

[35] Patrick J. Hayes. Naive physics I: Ontology for liquids. In J.R. Hobbs and B.C. Moore, editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex Publishing Corporation, Norwood, NJ, 1985.

[36] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[37] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *To appear in Artificial Intelligence*, 72, 1995.

[38] Yumi Iwasaki and Alon Y. Levy. Automated model section for simulation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1183–1190, Menlo Park, CA, 1994. AAAI Press/MIT Press.

[39] Yumi Iwasaki and Herbert A. Simon. Causality in device behavior. *Artificial Intelligence*, 29:3–32, 1986.

[40] Kenneth Man kam Yip. Model simplification by asymptotic order of magnitude reasoning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 634–640, Menlo Park, CA, 1993. AAAI Press/MIT Press.

[12] Ernest Davis. Order of magnitude reasoning in qualitative differential equations. In D. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.

[13] Randall Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24(1–3):347–410, December 1984.

[14] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–163, 1986.

[15] Johan de Kleer. Focusing on probable diagnoses. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 842–848, Menlo Park, CA, 1991. AAAI Press/MIT Press.

[16] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[17] Johan de Kleer, Olivier Raiman, and Mark Shirley. One step lookahead is pretty good. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*, pages 138–142. Morgan Kaufmann, San Mateo, CA, 1992.

[18] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[19] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1324–1330, San Mateo, CA, 1989. Morgan Kaufmann.

[20] J. Patrick Dishaw and Jeff Y.-C. Pan. AESOP: a simulation-based knowledge system for CMOS process diagnosis. *IEEE Transactions on Semiconductor Manufacturing*, 2:94–103, 1989.

[21] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[22] John W. Dufour and William E. Nelson. *Centrifugal Pump Sourcebook*. McGraw-Hill, New York, 1992.

[23] Brian Falkenhainer and Kenneth D. Forbus. Setting up large scale qualitative models. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 301–306, San Mateo, CA, 1988. Morgan Kaufmann.

[24] Brian Falkenhainer and Kenneth D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51(1–3):95–143, October 1991.

[25] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[26] Kenneth D. Forbus and Peter B. Whalley. Using qualitative physics to build articulate software for thermodynamics education. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1175–1182, Menlo Park, CA, 1994. AAAI Press/MIT Press.

Bibliography

[1] Sanjaya Addanki, Roberto Cremonini, and J. Scott Penberthy. Graphs of models. *Artificial Intelligence*, 51:145–177, 1991.

[2] R. Byron Bird, Warren E. Stewart, and Edwin N. Lightfoot. *Transport Phenomena*. John Wiley & Sons, Inc., New York, NY, 1960.

[3] Andrea Bonarini and Piera Sassaroli. Opportunistic multimodel-based diagnosis: framing all the knowledge we have to diagnose complex artifacts. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, pages 429–436, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[4] Scott Bublin and R. L. Kashyap. Generating fault hypotheses with a functional model in machine-fault diagnosis. *Applied Artificial Intelligence*, 6:353–382, 1992.

[5] Tom Bylander and B. Chandrasekaran. Understanding behavior using consolidation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 450–454, Los Altos, CA, 1985. Morgan Kaufmann.

[6] Nicholas P. Cheremisinoff and Paul N. Cheremisinoff. *Pumps and Pumping Operations*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[7] John W. Collins and Kenneth D. Forbus. Reasoning about fluids via molecular collections. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 503–507, Los Altos, CA, 1987. Morgan Kaufmann.

[8] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. In Walter Hamscher, Luca Console, and Johan de Kleer, editors, *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, CA, 1992.

[9] Stephen A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.

[10] Philippe Dague, Olivier Raiman, and Philippe Devès. Troubleshooting: when modeling is the trouble. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 600–605, Los Altos, CA, 1987. Morgan Kaufmann.

[11] David Dalle Molle. *Qualitative Simulation of Dynamic Chemical Processes*. PhD thesis, University of Texas at Austin, August 1989. Published as technical report AI-89-107.

have an edge to the same number of vertices in U' as there were literals in the corresponding clause of the SAT problem. The vertices in U' each have one edge to a vertex in C'.

Since this instance of SAT has a satisfying truth assignment for its variables, then at least one literal from each clause is true, and that satisfying literal cannot be the complement of the satisfying literal of any other clause. Therefore, any vertex in U' that corresponds to a satisfying literal must have edges to all vertices corresponding to satisfying literals, forming a clique of at least size $|C|$. Furthermore, since a vertex ${u'_i}^k$ or ${\overline{u}'_i}^k$ in U' cannot have an edge to any other ${u'_j}^k$ or ${\overline{u}'_j}^k$ that has an edge to the same vertex c'_k in C', then the clique will be of exactly size $|C|$ and its vertices will each have an edge to a different vertex in C' corresponding to those edges from vertices in X to those in C'. This clique of vertices in U' having the required matching and correspondence properties to the clique in X is a MODEL MATCH.

Conversely, suppose that graph $G = (X \cup C' \cup U', E)$ has a clique amongst the vertices in U' that corresponds to the clique containing all vertices in X, where the correspondence requires that each vertex of the clique in U' has exactly one edge to a vertex in C' and vice versa. This clique is of exactly size $|C|$, the number of clauses in the corresponding instance of SAT. Since no vertex ${u'_i}^k$ can have an edge to a vertex ${\overline{u}'_i}^l$, that represents the complementary literal, then the vertices of this clique form a truth assignment for this instance of SAT in the following way. Variables that are represented in the clique vertices as positive literals are assigned T, and those represented as negative literals in the clique vertices are assigned F. Variables not represented in the clique are assigned arbitrarily. Since the clique has each of its vertices attached to a different vertex in C', the set of vertices representing the clauses, the truth assignment contains one true literal for each clause, and thus is a satisfying truth assignment for this instance of SAT.

F.2.3 Transformation is Polynomial

Let the size of an instance $S = (U, C)$ of SAT be measured by the number of of literals, u_i or $\overline{u}_i$, that may be formed from the variables u_i of U and the number of clauses c_j in C. If n is the number of variables and m the number of clauses, than the size of SAT can be no more than $2nm$. To generate the MODEL MATCHING graph, each of the following steps were performed:

1. Generate the vertices in C', one for each clause in C.
   (m vertices)

2. Generate the vertices in X, one for each clause in C.
   (m vertices)

3. Generate the edges between corresponding vertices in X and C'.
   (m edges)

4. Generate edges between every pair of vertices in X.
   ($(m-1)m/2$ edges)

5. Generate vertices in U', one for each appearance of a literal in a clause.
   ($2nm$ vertices maximum)

6. Generate edges from each vertex ${u'_i}^k$ or ${\overline{u}'_i}^k$ in U' to the vertex c'_k in C' which represents its clause.
   ($2nm$ edges maximum)

7. Generate edges between vertices in U' according to the criteria given in the transformation.
   ($2nm(2nm+1)/2$ operations maximum, since each pair of vertices in U' would be compared to the criteria to determine if an edge should be placed there.)

The transformation is clearly polynomial, being composed of the sum of the polynomial operations described, with the most complex operation being $O(n^2m^2)$.

F.2.4 SAT is Satisfiable if and only if there is a Match

It remains to be shown that the clauses C of SAT are satisfiable if and only if the graph $G = (X \cup C' \cup U', E)$ has a subgraph on a subset of vertices in U' that matches and corresponds to the subgraph on vertices in X.

First, suppose that an instance of SAT, $S = (U, C)$, has a satisfying truth assignment, $t : U \rightarrow \{T, F\}$. Then at least one literal from each clause in C, either a variable u_i or its complement $\overline{u}_i$, must be assigned T. The graph $G = (X \cup C' \cup U', E)$ constructed from S has its vertices partitioned into three sets, where the sets X and C' each contain one vertex for each clause in C and each x_i in X has exactly one edge to a vertex c'_j in C', and each c'_j has exactly one edge to a vertex x_i. In addition, each x_i has an edge to every other vertex in X, forming a clique of size $|C|$. U' contains a vertex ${u'_i}^k$ or ${\overline{u}'_i}^k$ for each literal appearing in each clause of C in SAT and each vertex in C' will

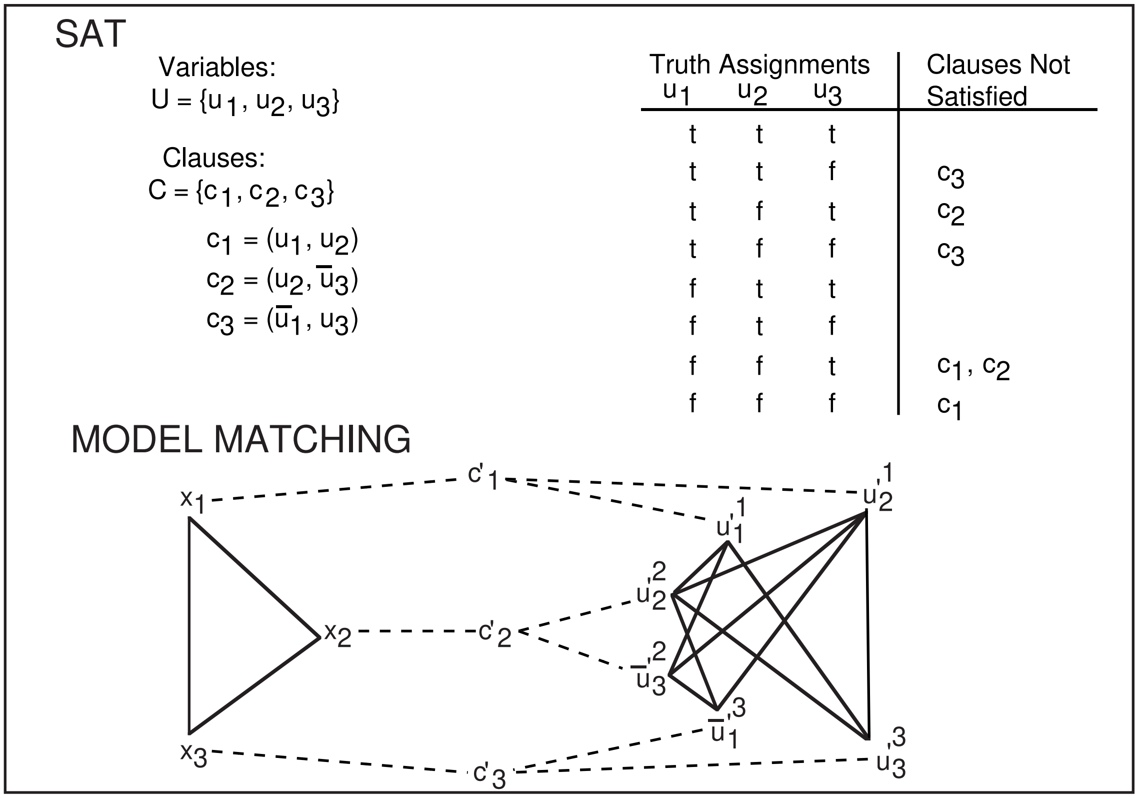


Figure F.2: Transformation of an instance of SAT that has three satisfying truth assignments to an instance of MODEL MATCHING that has at least one match for each satisfying truth assignment.

them unless they correspond to the same variable or unless they correspond to literal appearances in the same clause.)

3. $(\overline{u}'^{k}_i, \overline{u}'^{m}_l)$ where $k \neq m$
   (Any two vertices representing negative literals have an edge between them unless they correspond to literal appearances in the same clause.)

When the SAT instance has no satisfying truth assignments, the MODEL MATCHING instance will have no matches as illustrated in Figure F.1. When the SAT instance has a satisfying truth assignment, this transformation produces instances of MODEL MATCHING that have at least as many matches as the originating instance of SAT had satisfying truth assignments. Figure F.2 shows an example with three satisfying truth assignments and four different matches. The additional matches result from the different possible ways that a satisfying literal may be chosen from the literals in a clause. From the example in Figure F.2 for the single truth assignment ($u_1 = T, u_2 = T, u_3 = T$), we may choose the satisfying literals from the three clauses to be either u_1, u_2, u_3 or u_2, u_2, u_3.

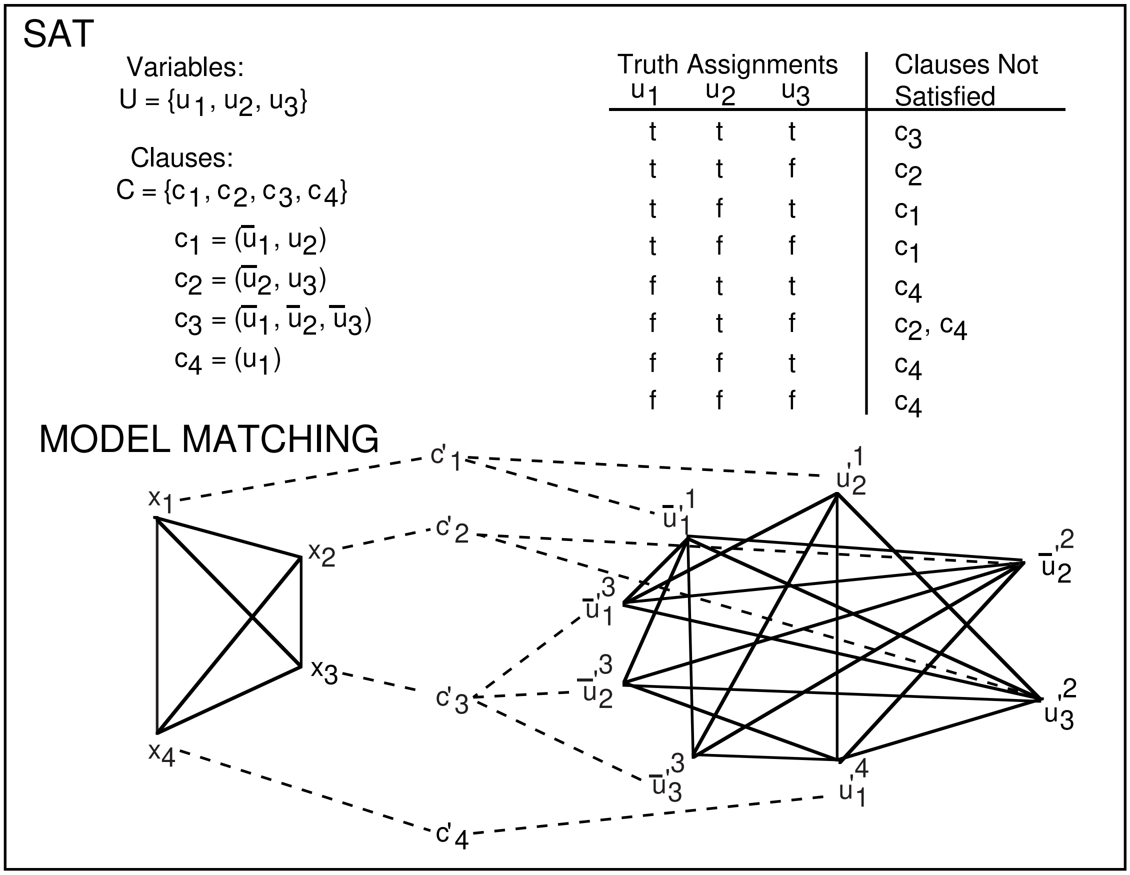


Figure F.1: Tranformation of an instance of SAT that has no satisfying truth assignment to an instance of MODEL MATCHING that has no match.

either set is connected to more than one vertex in the other set. The edges $(x_j,\ c_j')$ determine correspondence in the instance of model matching. The vertices in X each have edges to every other vertex x_i in X. The vertices in X and the edges $(x_i,\ x_j)$ constitute the model in MODEL MATCHING. See Figures F.1 and F.2 for examples.

For each appearance of a literal in a clause of SAT, the graph G contains a vertex ${u_i}'^k$ (or ${\overline{u}_i}'^k$) forming the vertex set U'. The subscript indicates which variable the vertex represents and the superscript indicates in which clause the literal appears, since the same literal may appear in more than one clause. Edges $({u_i'}^k,\ c_j')$ (or $({\overline{u}_i'}^k,\ c_j')$) connect each ${u_i'}^k$ to exactly one vertex c_j' where $k = j$, representing the appearance of the literal u_i (or $\overline{u}_i$) in the clause c_j.

The graph G also contains edges among the vertices in U of the following forms:

1. $({u_i'}^k, {u_l'}^m)$ where $k \neq m$

   (Any two vertices representing positive literals have an edge between them unless they correspond to literal appearances in the same clause.)

2. $({\overline{u}_i'}^k, {u_l'}^m)$ where $i \neq l$ and $k \neq m$

   (Any two vertices representing a positive literal and a negative literal have an edge between

SAT had a solution. This sequence of proofs shows that MODEL MATCHING is at least as hard as SAT, and if $P \neq NP$ then MODEL MATCHING is NP-complete.

F.2.1 MODEL-MATCHING is in NP

MODEL MATCHING $\in$ NP if a nondeterministic algorithm may guess a subset of vertices in U' and edges amongst those vertices as a possible solution to MODEL MATCHING and then verify in polynomial time that the guess is a solution. Since verification requires a comparison between a graph on the vertices in X and C' to a subgraph on vertices in U' and C', the complexity is $O(|X| + |C'|)^2$, which is clearly polynomial in the size of the MODEL MATCHING instance. MODEL MATCHING is therefore in NP.

F.2.2 Transformation from SAT to MODEL-MATCHING

To show that MODEL MATCHING is NP-complete, we provide an algorithm to transform an arbitrary instance of a known NP-complete problem, the Satisfiability Problem (SAT), into MODEL MATCHING. SAT was proved NP-complete by Cook [9] and was the first such known problem. In Section F.2.3 we demonstrate that this transformation can be performed in polynomial time.

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of Boolean variables. We can then define *truth assignment*, *literal*, *clause*, and *SAT* as in [30].

Definition F.2 (truth assignment) *A truth assignment for U is a function $t : U \rightarrow \{T, F\}$.*

If $t(u) = T$, then u is said to be "true" under t, and if $t(u) = F$ then we say u is "false."

Definition F.3 (literal) *For $u \in U$, u and $\overline{u}$ are literals over U. A literal u is true under truth assignment t if and only if the variable u is true under t. The literal $\overline{u}$ is true under t if and only if the variable u is false.*

Definition F.4 (clause) *A clause over U is a set of literals over U representing the disjunction of those literals.*

A clause is said to be satisfied under a truth assignment t if at least one of the literals in the clause is true under t.

Definition F.5 (The Satisfiability Problem–SAT) *Given a set U of variables and a collection C of clauses over U, is there a truth assignment that satisfies all the clauses in C?*

We must construct a graph, $G = (X \cup C' \cup U', E)$ where G has a subgraph among the vertices in U' that matches the graph on the vertices in X and corresponds through the vertices in C' if and only if the instance of SAT $S = (U, C)$ from which we built G has a satisfying truth assignment. For each clause c_j in C, the graph G has a vertex c'_j forming the required set C' of vertices. For each clause c_j, there is a vertex x_j forming the set X. Exactly $|C|$ edges connect the vertices in X and the vertices from C' so that each edge covers one vertex from each set, and no vertex from equipment descriptions, and the model descriptions, each of which can only originate and terminate at particular types of objects. For example, the `STREAM-OF-PORT` link may only orignate at an object that exemplifies some port type and may only terminate at some object the exemplifies some stream type. For the purposes of this proof, we exclude descriptions that use the 5 disaggregation links. Some models that we have implemented, such as the Dittus-Boelter model (Section C.2), do not use these links. If these links happen to reduce the complexity of matching from NP-complete to polynomial, but problems that do not use them can be shown to be NP-complete, then the general problem is still NP-complete. So if we exclude the disaggregation links, any link name could be determined unambiguously from the types of objects at either end. For the purposes of this proof, we can treat links as unlabeled edges.

The simplifications described result in a graph $G = (V, E)$ where the vertices V are partitioned into three subsets, the set of model description objects X, the set of type hierarchy objects C', and the set of equipment description objects U'. The set of edges E, made up of pairs of vertices from V, (v_i, v_j), have either both $v_i, v_j \in X$, both $v_i, v_j \in U'$, $v_i \in X$ and $v_j \in C'$, or $v_i \in U'$ and $v_j \in C'$. G contains no edges with one vertex in X and one in U'. In other words, no model description objects are linked directly to equipment description objects. The graph $G = (X \cup C' \cup U', E)$ is the input for the formalized model matching problem, as defined below.

F.1.4 The Formal Problem

To match a model, a subset U'_{sub} of U' must "correspond" to X, and a subgraph containing the vertices of U'_{sub} must be isomorphic to the graph on X. U'_{sub} is said to *correspond* to X if there is a one-to-one correspondence between the elements in X and the elements in U' where for each x_i in X, the corresponding u'_i in U'_{sub} has edges to all the same $c_1, c_2, ...c_n$ in C' as x_i. We now define MODEL MATCHING to be the formalized version of the original problem.

Definition F.1 (MODEL MATCHING) *Given a graph $G = (X \cup C' \cup U', E)$ where no edge in E is incident on both a vertex x_i in X and a vertex u'_i in U' and where X, C', and U' are disjoint sets of vertices, is there a subgraph on a subset of U' that corresponds and is isomorphic to the subgraph on X?*

This definition states MODEL MATCHING as a decision problem so that the techniques for proving NP-completeness may be applied.

F.2 Complexity of Model-Matching

In this section, MODEL MATCHING is shown to be NP-complete according to the proof procedure described by [30]. First MODEL MATCHING is shown to be in the class NP. A transformation from an instance of a known NP-complete problem, The Satisifiability Problem (SAT), to an instance of MODEL MATCHING is given and shown to require polynomial time. The transformation is shown to produce an instance of MODEL MATCHING that has a match if and only if the instance of two objects are required to match, they act as an anchored starting point for the problem. (The equipment parameter and model parameter correspond to the input to the M-R portion of the algorithms in this thesis, as described in Chapter 5.)

F.1.2 The Informal Problem

A model matches part of an equipment description when all of the model objects (possibly excluding parameters) have corresponding equipment objects of the same type as the model object or a subtype of the model object's type and the links between the model's objects have corresponding links between the equipment objects. In other words, objects match when the model object is more abstract or equally abstract as the equipment object, and links match only if they have the same name. For the problem at hand we assume we have one object in the model matched and anchored to one object in the equipment.

F.1.3 The Formal Input

Since an equipment description may be large and mostly unrelated to the model at hand, irrelevant parts may be pruned as follows. Notice that a model description's objects and an equipment description's objects are connected indirectly through the type hierarchies, since all objects must have at least one link to a type hierarchy object. First traverse all the EXEMPLIFIES links from the model objects to find thier associated type objects (called "model's types") in the type hierarchy. Only parts of the equipment description that are objects of these types or of subtypes of the model's types may match the model. By following the trees rooted at the model's types and made up of CAN-BE-A and EXEMPLIFIED-BY links, we reach all the correct type objects in the equipment description. For the purposes of formal analysis, each path to the leaves of these trees is replaced by a single EXEMPLIFIES link from the leaf to the root. If we then exclude all other objects not linked to our tree roots and include any links where the objects at both ends are in the selected set of objects, the resulting graph contains no objects of irrelevant types. This operation is a tree search so will require time polynomial in the number of nodes in the trees.

In the matching procedure, failure occurs because of either lack of objects of appropriate type, lack of correct links between objects, or unsatisfied value requirements for the attributes of parameter objects. Since parameter objects have a fixed number of attributes (See Section ref?.) which may take values from a fixed set of value types, then checking a model's requirements on a parameter's attributes requires constant time (assuming that we limit the representation of large numbers to some fixed number of significant digits.) Since the previously described tree search to find equipment objects exemplifying model types is polynomial, ignoring these constant time attribute checking operations will not affect the order of complexity in the final analysis.

The links between objects may also be simplified for analysis, since, except for the 5 disaggregation links (`has-part`, `has-component`, `has-phase-component`, `has-spatial-portion`, and `has-instance`), the link names are redundant with the types of the objects they connect. The representation method defined in this thesis uses only 13 links for use in the type hierarchies, the

Appendix F

Proof of NP-Completeness for the Matching Problem

F.1 Formalizing the Model-Matching Problem

To make the problem of matching a model description to an equipment description amenable to formal analysis, this section introduces some representational modifications that preserve the original problem content. Some polynomial time operations to select parts of the equipment description graph produce a subgraph used as input for the formalized model matching problem. Since the following section shows the formalized model matching problem to be NP-complete, these polynomial subgraph selections do not significantly impact the complexity of the whole problem.

F.1.1 The Informal Input

Recall from Chapter 4 that type hierarchies (materials, geometries, functional types, etc.) form a vocabulary that describes both models and the equipment being diagnosed. Objects to which models may be applied, physical parts of the equipment, and parameters are represented by frame-like objects with links indicating which types they exemplify. Parameter objects are linked to the object they describe and have a set of attributes which may take on various types of values. Objects may also have links between them indicating physical connections in the equipment or other relationships. This representation scheme uses 13 types of links (and their inverses) to capture these possible relationships.

An equipment description is a set of objects each with at least one link to the type hierarchies and possibly having links chosen from the 13 link types indicating relationships between them. Some of these objects represent parameters and thus have a particular set of attributes, each of which may take on various types of values.

A model description contains objects (some of them being parameters) and links as in an equipment description, but here these objects represent what equipment the model may be applied to rather than representing a physically existing system as the equipment description does. Varying levels of abstraction in the model description allow it to be applied to more or less situations. The parameter objects represent what parameters are involved in the model's calculation, either as applicability conditions or as input or output. Besides the model and equipment descriptions, an equipment parameter and a matching model parameter are assumed to be given. Since these

```
       (user::load "sys:site;nq-92.translations")
       (user::make-system 'nq-92)
       (load "x23:murdock.models;dalle-molle-interface.lisp")

  (eval (funcall #'macroexpand
 `(qsim::simulate-mixing-tank
    ,coldflow-before
    ,coldflow-after
    ,coldflow-q-space
    ,hotflow-before
    ,hotflow-after
    ,hotflow-q-space
    ,height
    ,height-q-space
    ,tempout
    ,tempout-q-space))
))
```

E.3 An Auxiliary Function that Retrieves Output

Since the Dalle Molle model and QSIM do not return values, we built this function to retrieve the values it generated rather than letting them be sent to a visual display intended for humans.

```
;;;Edited by MURDOCK               5 Aug 94  12:29
;;;Edited by MURDOCK               7 Oct 94  13:38
(defun state-successors-leafs (start-state)
  "The function STATE-SUCCESSORS-LEAFS takes the initial state generated by a
   QSIM simulation, held by the global variable *initial-state*, and returns
   a list of the final states generated in the simulation."

  (let ((new-queue (user::make-queue)))
    (user::enqueue new-queue start-state)
    (do ((search-queue  new-queue )
 (leaf-list nil))

((user::empty-queue-p search-queue) leaf-list)

      (let ((succ-states
              (cdr (qsim::state-successors
                     (user::queue-front search-queue))))
    )
(if (null succ-states)
    (setq leaf-list (cons (user::queue-front search-queue) leaf-list))
    (mapcar #'(lambda (this-state) (user::enqueue search-queue  this-state))
    succ-states))
(user::dequeue search-queue)  ;Since dequeue works by side-effect,
                              ;it can't be used as the stepper form for
                              ;the SEARCH-QUEUE variable of the DO.
))))
```

E.4 An Auxiliary Function to Call the Macro

Since our system required models to be executed only by *function* call, we wrote this function as a wrapper for the macro above.

```
(defun dalle-molle-simulation (coldflow-before
coldflow-after
coldflow-q-space
hotflow-before
hotflow-after
hotflow-q-space
height
height-q-space
tempout
tempout-q-space
)
```

```
 DT cf*DT hf*DT cf*DT/if hf*DT/if ti-to if*ti-to)
(other (ignore-qdirs dT/dt))
(layout (height tempout) (netflow dT/dt))
)

      (setq *analytic-functions-only* t)
      (setq *new-landmarks-at-initial-state* nil)

      (let ((normal (make-initial-state hot-cold-mixing-tank
               (quote
                ((coldtemp (,tc* std)) (hottemp (,th* std))
                (hotflow (,hotflow-b std)) (coldflow (,coldflow-b std))
                (tempout (,tempout-value std)) (height (,height-value std))))
             "Normal"))
            (initial (make-initial-state hot-cold-mixing-tank
             (quote
              ((coldtemp (,tc* std)) (hottemp (,th* std))
               (hotflow (,hotflow-a std)) (coldflow (,coldflow-a std))
               (tempout (,tempout-value nil)) (height (,height-value nil))))
            "Step increase or decrease in either/both coldflow or hotflow."))
    )
(qsim initial)
(let ((final-states-list (delete-if #'(lambda (this-state)
(or (inconsistent-p this-state)
    (incomplete-p this-state)
    (null (state-status this-state))))
    (state-successors-leafs initial)))
      )
  (format t "~% Final states are: ~S" final-states-list)
  (cond ((and (eq (length final-states-list) 1)
      (quiescent-p (car final-states-list)))
 (let* ((final-state (car final-states-list))
(final-qvalues (state-qvalues final-state))
)
   (in-package 'user)
   (list (qsim::qmag (qsim::qval 'height final-state))
 (qsim::qspace 'height final-state)
 (qsim::qmag (qsim::qval 'tempout final-state))
 (qsim::qspace 'tempout final-state)
 (qsim::qmag (qsim::qval 'time final-state))
 (qsim::qspace 'time final-state))))
(t (in-package 'user)
   nil)))

      ))))
```

```
        (tempout-value (car tempout))

)


   `(progn
      (define-qde hot-cold-mixing-tank
 (text "Revised model of tank mixing a hot and cold stream with
                redundant constraints.")
(quantity-spaces
  (volume (0 v* inf))
  (height (0 ,h* inf))
  (outflow (0 of* inf))
  (inflow (0 if* inf))
  (netflow (minf 0 inf))
  (coldflow (0 ,fc- ,fc* ,fc+ inf))
  (hotflow (0 ,fh- ,fh* ,fh+ inf))
  (tempout (0 ,tc* ,ts* ,th* inf))
  (coldtemp (0 ,tc* inf))
  (hottemp (0 ,th* inf))
  (DT (0 ,th*-tc* inf))
  (cf*DT (0 ,fc*DT inf))
  (hf*DT (0 ,fh*DT inf))
  (cf*DT/if (0 ,fc*DT/if* inf))
  (hf*DT/if (0 ,fh*DT/if* inf))
  (tempin (0 ,tc* ,ts* ,th* inf))
  (dT/dt (minf 0 inf))
  (ti-to (minf 0 inf))
  (if*ti-to (minf 0 inf))
  )

(constraints
  ((m+ volume height) (v* ,h*) (inf inf) (0 0))
  ((m+ outflow height) (of* ,h*) (inf inf) (0 0))
  ((add outflow netflow inflow) (of* 0 if*) (inf 0 inf) (0 0 0))
  ((add coldflow hotflow inflow) (,fc* ,fh* if*) (0 0 0))
  ((d/dt volume netflow))
  ((add DT coldtemp hottemp) (,th*-tc* ,tc* ,th*) (0 0 0))
  ((add tempin cf*DT/if hottemp) (,th* 0 ,th*) (,ts* ,fc*DT/if* ,th*) (0 0 0))
  ((add coldtemp hf*dt/if tempin) (,tc* 0 ,tc*) (,tc* ,fh*DT/if* ,ts*) (0 0 0))
  ((mult cf*DT/if inflow cf*DT) (,fc*DT/if* if* ,fc*DT))
  ((mult hf*DT/if inflow hf*DT) (,fh*DT/if* if* ,fh*DT))
  ((mult coldflow DT cf*DT) (,fc* ,th*-tc* ,fc*DT))
  ((mult hotflow DT hf*DT) (,fh* ,th*-tc* ,fh*DT))
  ((add tempout ti-to tempin) (,th* 0 ,th*) (,tc* 0 ,tc*) (,ts* 0 ,ts*))
  ((d/dt tempout dT/dt))
  ((mult volume dT/dt if*ti-to) (v* 0 0))
  ((mult inflow ti-to if*ti-to) (if* 0 0))
  )

(independent hottemp coldtemp coldflow hotflow)
(history volume height outflow inflow netflow tempout tempin dT/dt
```

```
  '((coldtemp (tc* std)) (hottemp (th* std))
    (hotflow (fh* std)) (coldflow (fc+ std))
    (tempout (ts* nil)) (height (h* nil))
    )
  "Step increase in coldflow"))
    (qsim-display (qsim initial) :reference-states
  `((normal ,normal)))
    ))
```

E.2 The Macro that Generates the Dalle Molle Model

This macro was built to generate the functions that the Dalle Molle model normally required humans to enter. Notice the `progn` construct that contains the identical functions as above.

```
;;;Edited by MURDOCK              25 Oct 94  16:16
;;;Edited by MURDOCK              26 Oct 94  11:28
;;;Edited by MURDOCK              26 Oct 94  12:20
(defmacro simulate-mixing-tank (coldflow-before      ;before the step input
coldflow-after       ;after the step change
coldflow-q-space     ;list of 5, first is 0, last is inf
hotflow-before       ;before the step change
hotflow-after        ;after the step change
hotflow-q-space      ;list of 5, first is 0, last is inf
height
height-q-space       ;list of 3, first is 0, last is inf
tempout
tempout-q-space      ;list of 5, first is 0, last is inf
)
  (let* ((h* (second height-q-space))
 (fc- (second coldflow-q-space))
 (fc* (third coldflow-q-space))
 (fc+ (fourth coldflow-q-space))
 (fh- (second hotflow-q-space))
 (fh* (third hotflow-q-space))
 (fh+ (fourth hotflow-q-space))
 (tc* (second tempout-q-space))
 (ts* (third tempout-q-space))
 (th* (fourth tempout-q-space))
 (th*-tc* (make-concat-symbol (make-concat-symbol th* '-) tc*))
 (fc*DT (make-concat-symbol fc* 'DT))
 (fh*DT (make-concat-symbol fh* 'DT))
 (fc*DT/if* (make-concat-symbol fc*DT '/if*))
 (fh*DT/if* (make-concat-symbol fh*DT '/if*))
 (coldflow-b (car coldflow-before))
 (coldflow-a (car coldflow-after))
 (hotflow-b  (car hotflow-before))
 (hotflow-a  (car hotflow-after))
 (height-value (car height))
```

```
  (constraints
    ((m+ volume height) (v* h*) (inf inf) (0 0))
    ((m+ outflow height) (of* h*) (inf inf) (0 0))
    ((add outflow netflow inflow) (of* 0 if*) (inf 0 inf) (0 0 0))
    ((add coldflow hotflow inflow) (fc* fh* if*) (0 0 0))
    ((d/dt volume netflow))
    ((add DT coldtemp hottemp) (th*-tc* tc* th*) (0 0 0))
    ((add tempin cf*DT/if hottemp) (th* 0 th*) (ts* fc*DT/if* th*) (0 0 0))
    ((add coldtemp hf*dt/if tempin) (tc* 0 tc*) (tc* fh*DT/if* ts*) (0 0 0))
    ((mult cf*DT/if inflow cf*DT) (fc*DT/if* if* fc*DT))
    ((mult hf*DT/if inflow hf*DT) (fh*DT/if* if* fh*DT))
    ((mult coldflow DT cf*DT) (fc* th*-tc* fc*DT))
    ((mult hotflow DT hf*DT) (fh* th*-tc* fh*DT))
    ((add tempout ti-to tempin) (th* 0 th*) (tc* 0 tc*) (ts* 0 ts*))
    ((d/dt tempout dT/dt))
    ((mult volume dT/dt if*ti-to) (v* 0 0))
    ((mult inflow ti-to if*ti-to) (if* 0 0))
; constraints for p controller
;   ((add errheight height setheight) (0 h- h-) (0 h* h*) (0 h+ h+))
;   ((m+ coldflow errheight) (fc* 0))
    )

  (independent hottemp coldtemp coldflow hotflow)
; (independent hottemp coldtemp setheight hotflow)
  (history volume height outflow inflow netflow tempout tempin dT/dt
   DT cf*DT hf*DT cf*DT/if hf*DT/if ti-to if*ti-to)
; (history volume height outflow inflow netflow tempout tempin dT/dt
;          DT cf*DT hf*DT cf*DT/if hf*DT/if ti-to if*ti-to
;          err-height coldflow)
  (other (ignore-qdirs dT/dt))
; (other (ignore-qdirs dT/dt ti-to if*ti-to))
  (layout (height tempout) (netflow dT/dt)
  )
  )


;;;Edited by MURDOCK               3 Aug 94  22:05
;;;Edited by MURDOCK               3 Aug 94  22:08
(defun hct-fc+ ()
  (let ((normal nil)
(initial nil)
(analytic-functions-only t)
(*new-landmarks-at-initial-state* nil))
    (declare (special normal initial analytic-functions-only))
    (setq *analytic-functions-only* t)
    (setq normal (make-initial-state hot-cold-mixing-tank
 '((coldtemp (tc* std)) (hottemp (th* std))
   (hotflow (fh* std)) (coldflow (fc* std))
   (tempout (ts* std)) (height (h* std)))
 "Normal"))
    (setq initial (make-initial-state hot-cold-mixing-tank
```

Appendix E

Interface to the Dalle Molle Model

E.1 The Original Dalle Molle Model

To run the Dalle Molle model, one had to enter these two functions, where the second function sets the values of the input parameters. Then the model was exectucted by calling the second function without any arguments. The model calls QSIM to do its simulation and then normally generates a visual display for humans rather than returning values.

```
;;;Edited by MURDOCK              3 Aug 94  19:18
;;;Edited by MURDOCK              3 Aug 94  20:33
;;;Edited by MURDOCK              3 Aug 94  22:05
(define-qde hot-cold-mixing-tank
  (text "Revised model of a tank mixing a hot and a cold stream with
         redundant constraints")
  (quantity-spaces
    (volume (0 v* inf))
    (height (0 h* inf))          ;used for open-loop case
;   (height (0 h* h+ inf))       ;used for feed back control
    (outflow (0 of* inf))
    (inflow (0 if* inf))
    (netflow (minf 0 inf))
    (coldflow (0 fc- fc* fc+ inf)) ;used for open-loop case
;   (coldflow (0 fc* inf))
    (hotflow (0 fh- fh* fh+ inf))
    (tempout (0 tc* ts* th* inf))
    (coldtemp (0 tc* inf))
    (hottemp (0 th* inf))
    (DT (0 th*-tc* inf))
    (cf*DT (0 fc*DT inf))
    (hf*DT (0 fh*DT inf))
    (cf*DT/if (0 fc*DT/if* inf))
    (hf*DT/if (0 fh*DT/if* inf))
    (tempin (0 tc* ts* th* inf))
    (dT/dt (minf 0 inf))
    (ti-to (minf 0 inf))
    (if*ti-to (minf 0 inf))
; quantity spaces for p controller
;   (setheight (0 h- h* h+))
;   (errheight (minf 0 inf))
    )
```

D.17.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
No reconfigurations were made.			
req-p -- required parameter p-obj -- positive object	opt-p -- optional parameter neg-n -- member of the nth negative set matched		

(**any-common-portion?**

m9-shear-stress-1	time	m9-shear-stress-1	time-type
m9-shear-stress-1	time-interval-type	m9-shear-stress-1	time-precision
m9-shear-stress-1	time-precision-type	m9-viscosity-1	time
m9-viscosity-1	time-type	m9-viscosity-1	time-interval-type
m9-viscosity-1	time-precision	m9-viscosity-1	time-precision-type)

(**equal** m9-viscosity-1 time-units m9-viscosity-2 time-units)

(**equal** m9-viscosity-1 time-units m9-shear-stress-2 time-units)

(**equal** m9-viscosity-1 time-units m9-shear-stress-1 time-units)

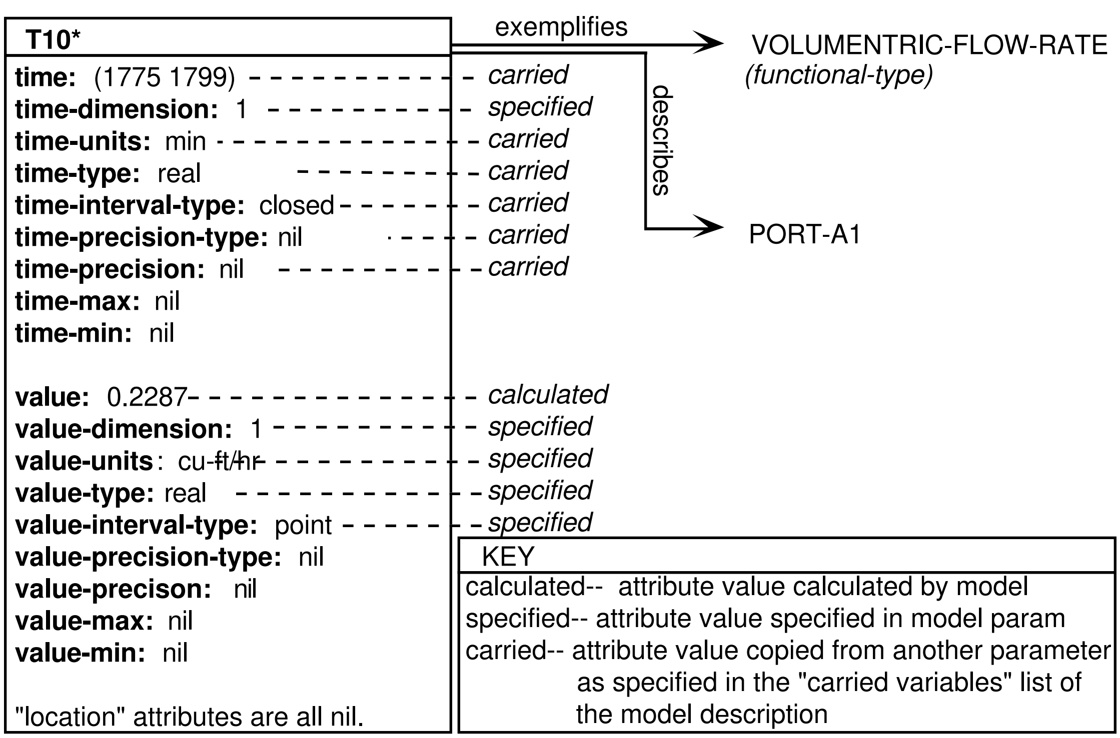
(**equal** m9-viscosity-1 time-units m9-inlet-pressure-2 time-units))

D.17.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
T10

SIDE-EFFECTS: The parameter object, T10, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

D.17.4 Checking Model Conditions

INPUTS:

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:

```
T
((m9-hagen::is-flow-steady-state?
  m9-inlet-pressure        value        m9-inlet-pressure-2     value)

 (m9-hagen::fluid-is-newtonian?
  m9-shear-stress-1      value        m9-viscosity-1     value
  m9-shear-stress-2      value        m9-viscosity-2     value)

 (equal    m9-shear-stress-1    value-units        m9-shear-stress-2    value-units)

 (m9-hagen::flow-is-laminar?
   m9-pipe-inside-diameter     value        m9-vol-flow-rate     value
   m9-density                  value        m9-viscosity-1       value)

 (possibly-equal?
   m9-pressure-deriv   value                     m9-pressure-deriv   value-type
   m9-pressure-deriv   value-interval-type       m9-pressure-deriv   value-precision
   m9-pressure-deriv   value-precision-type      nil                 0
   nil                 real                      nil                 point
   nil                 nil                       nil                 nil)

 (not-any-common-portion?
   m9-shear-stress-1    time                    m9-shear-stress-1    time-type
   m9-shear-stress-1    time-interval-type      m9-shear-stress-1    time-precision
   m9-shear-stress-1    time-precision-type     m9-shear-stress-2    time
   m9-shear-stress-2    time-type               m9-shear-stress-2    time-interval-type
   m9-shear-stress-2    time-precision          m9-shear-stress-2    time-precision-type)

 (any-common-portion?
   m9-shear-stress-2    time                    m9-shear-stress-2    time-type
   m9-shear-stress-2    time-interval-type      m9-shear-stress-2    time-precision
   m9-shear-stress-2    time-precision-type     m9-viscosity-2       time
   m9-viscosity-2       time-type               m9-viscosity-2       time-interval-type
   m9-viscosity-2       time-precision          m9-viscosity-2       time-precision-type)

 (any-common-portion?
   m9-inlet-pressure    time                    m9-inlet-pressure    time-type
   m9-inlet-pressure    time-interval-type      m9-inlet-pressure    time-precision
   m9-inlet-pressure    time-precision-type     m9-vol-flow-rate     time
   m9-vol-flow-rate     time-type               m9-vol-flow-rate     time-interval-type
   m9-vol-flow-rate     time-precision          m9-vol-flow-rate     time-precision-type)
```

D.17.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm9-port-1b 'm9-inlet-pressure 'port-a1 'inlet-pressure-1)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m9-inlet-pressure	req-p	inlet-pressure-a	
1	m9-port-1b	p-obj	port-a1	
2	m9-stream-1	p-obj	stream-x2-a1	
3	m9-edge-1b	p-obj	pipe-1-edge-2-1	
4	m9-fluid-in-pipe	p-obj	region-a	
5	m9-port-3b	p-obj	pipe-2-port-3a	
6	m9-port-3a	p-obj	port-a3	
7	m9-port-2a	p-obj	port-a2	
8	m9-viscosity-1	req-p	mu-a	
9	m9-stream-3	p-obj	stream-pipe-2-a3	
10	m9-pipe	p-obj	pipe-2	
11	m9-edge-2a	p-obj	edge-a2-1	
12	m9-length	req-p	region-a-length	
13	m9-stream-2	p-obj	stream-a2-b1	
14	m9-outlet-pressure	req-p	outlet-pressure-a	
15	m9-pipe-inside-diameter	req-p	pipe-2-id	
16	m9-surface-normal-1b	opt-p	port-normal-a1	
17	m9-vol-flow-rate	opt-p	nil	
18	m9-inlet-pressure-2	opt-p	nil	
19	m9-viscosity-2	opt-p	nil	
20	m9-density	opt-p	rho-a	
21	m9-shear-stress-1	opt-p	nil	
22	m9-shear-stress-2	opt-p	nil	
23	m9-surface-normal-2a	opt-p	port-normal-a2	
24	m9-pipe-axial-direction	opt-p	pipe-2-axial-direction	
25	m9-neg-edge-1b	neg-1	nil	
26	m9-neg-s-normal-1b	neg-2	nil	
27	m9-neg-endpt-1b	neg-3	nil	
28	m9-neg-port	neg-4	nil	
29	m9-neg-edge-3a	neg-5	nil	
30	m9-neg-edge-2a	neg-6	nil	
31	m9-neg-s-normal-2a	neg-7	nil	
32	m9-neg-endpt-2a	neg-8	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.17 Hagen-Poiseuille Model and PIPES-1 Equipment Description

D.17.1 Statistics on the Matching

Equipment Description:	PIPES-1
Model Description:	HAGEN-POISEUILLE
Goal:	find EFFECTS
Max. partial matches:	30
Complete Matches:	1
Rematches:	0
Run time:	0:02 (hr:min)
Objects created (kept):	1 (1)
Links created (kept):	2 (2)
Separately triggered reconfigurations:	0
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.17.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'inlet-pressure-a 'value 'describes 'port-a1 'effects)

OUTPUTS:
(
(inlet-pressure-a value describes port-a1 effects m9-outlet-pressure m9-port-2 a)
(inlet-pressure-a value describes port-a1 effects m9-inlet-pressure m9-port-1 b)
)

D.16.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m10-pipe-roughness	req-p		
1	m10-port-3b	p-obj		
2	m10-stream-3	p-obj		
3	m10-pipe	p-obj		
4	m10-edge-2a	p-obj		
5	m10-edge-1b	p-obj		
6	m10-port-3a	p-obj		
7	m10-pipe-inside-diameter	req-p		
8	m10-port-2a	p-obj		
9	m10-port-1b	p-obj		
10	m10-fluid-in-pipe	p-obj		
11	m10-stream-2	p-obj		
12	m10-stream-1	p-obj		
13	m10-friction-factor	opt-p		
14	m10-outlet-temperature	opt-p		
15	m10-inlet-temperature	opt-p		
16	m10-velocity	opt-p		
17	m10-density	opt-p		
18	m10-viscosity	opt-p		
			intensive	(region-c region-y) (region-c region-b)
19	m10-neg-endpt-2a	neg-1		
			intensive	(region-c region-d) (region-c region-e) (region-c region-y region-d) (region-c region-y region-e) (region-c region-y region-b) (region-c region-b region-d) (region-c region-b region-e) (region-c region-b region-a) (region-c region-d region-e)
20	m10-neg-endpt-1b	neg-2		
21	m10-neg-edge-3a	neg-3		
22	m10-neg-edge-2a	neg-4		
23	m10-neg-edge-1b	neg-5		
24	m10-neg-port	neg-6		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

D.16.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
(T3)

SIDE-EFFECTS: The Parameter Object, T3, is created with attributes and links shown.

T3*

- **time:** (1775 1780) - - - *carried*
- **time-dimension:** 1 - - - *specified*
- **time-units:** min - - - *carried*
- **time-type:** real - - - *specified*
- **time-interval-type:** closed - - - *carried*
- **time-precision-type:** absolute - - - *carried*
- **time-precision:** ±0 - - - *carried*
- **time-max:** nil
- **time-min:** nil

- **value:** 0.00529 - - - *calculated*
- **value-dimension:** 1 - - - *specified*
- **value-units**: dimensionless - - - *specified*
- **value-type:** real - - - *specified*
- **value-interval-type:** point
- **value-precision-type:** nil
- **value-precison:** nil
- **value-max:** nil
- **value-min:** nil

"location" attributes are all nil.

T3 $\xrightarrow{\text{exemplifies}}$ FRICTION-FACTOR *(functional-type)*

T3 $\xrightarrow{\text{describes}}$ T312 *(the port representing the portion of the inner surface of Pipe-2 adjacent to the liquid region)*

KEY

calculated-- attribute value calculated by model
specified-- attribute value specified in model param
carried-- attribute value copied from another parameter as specified in the "carried variables" list of the model description

* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

```
(any-common-portion?
    m10-density    time                     m10-density    time-type
    m10-density    time-interval-type       m10-density    time-precision
    m10-density    time-precision-type      m10-velocity   time
    m10-velocity   time-type                m10-velocity   time-interval-type
    m10-velocity   time-precision           m10-velocity   time-precision-type)

(any-common-portion?
 m10-pipe-inside-diameter  time                  m10-pipe-inside-diameter  time-type
 m10-pipe-inside-diameter  time-interval-type    m10-pipe-inside-diameter  time-precision
 m10-pipe-inside-diameter  time-precision-type   m10-velocity              time
 m10-velocity              time-type             m10-velocity              time-interval-type
 m10-velocity              time-precision        m10-velocity              time-precision-type)

(any-common-portion?
  m10-pipe-roughness   time                  m10-pipe-roughness   time-type
  m10-pipe-roughness   time-interval-type    m10-pipe-roughness   time-precision
  m10-pipe-roughness   time-precision-type   m10-velocity         time
  m10-velocity         time-type             m10-velocity         time-interval-type
  m10-velocity         time-precision        m10-velocity         time-precision-type)

(equal   m10-pipe-roughness time-units    m10-outlet-temperature time-units)

(equal   m10-pipe-roughness time-units    m10-inlet-temperature time-units))

(equal   m10-pipe-roughness   time-units    m10-velocity   time-units))
```

INPUTS:

```
(CHECK-MODEL-CONDITIONS  'approx-conditions  assoc-list  (second vector-list))
```

OUTPUTS:

```
T
((m10-friction::re-and-kd-ok?
   m10-pipe-inside-diameter     value     m10-velocity    value
   m10-density                  value     m10-viscosity   value
   m10-pipe-roughness           value)

 (m10-friction::re-ok?
   m10-pipe-inside-diameter     value     m10-velocity    value
   m10-density                  value     m10-viscosity   value)

 (possibly-equal?
   m10-outlet-temperature  value                      m10-outlet-temperature  value-type
   m10-outlet-temperature  value-interval-type        m10-outlet-temperature  value-precision
   m10-outlet-temperature  value-precision-type       m10-inlet-temperature   value
   m10-inlet-temperature   value-type                 m10-inlet-temperature   value-interval-type
   m10-inlet-temperature   value-precision            m10-inlet-temperature   value-precision-type)

 (equal    m10-inlet-temperature   value-units     m10-outlet-temperature value-units)

 (any-common-portion?
   m10-outlet-temperature  time                       m10-outlet-temperature  time-type
   m10-outlet-temperature  time-interval-type         m10-outlet-temperature  time-precision
   m10-outlet-temperature  time-precision-type        m10-velocity            time
   m10-velocity            time-type                  m10-velocity            time-interval-type
   m10-velocity            time-precision             m10-velocity            time-precision-type)

 (any-common-portion?
   m10-inlet-temperature   time                       m10-inlet-temperature   time-type
   m10-inlet-temperature   time-interval-type         m10-inlet-temperature   time-precision
   m10-inlet-temperature   time-precision-type        m10-velocity            time
   m10-velocity            time-type                  m10-velocity            time-interval-type
   m10-velocity            time-precision             m10-velocity            time-precision-type)

 (any-common-portion?
   m10-viscosity   time                       m10-viscosity   time-type
   m10-viscosity   time-interval-type         m10-viscosity   time-precision
   m10-viscosity   time-precision-type        m10-velocity    time
   m10-velocity    time-type                  m10-velocity    time-interval-type
   m10-velocity    time-precision             m10-velocity    time-precision-type)
```

D.16.4 Checking Model Conditions

INPUTS:

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:

T

```
((possibly-equal?
   m10-outlet-temperature  value                  m10-outlet-temperature  value-type
   m10-outlet-temperature  value-interval-type    m10-outlet-temperature  value-precision
   m10-outlet-temperature  value-precision-type   m10-inlet-temperature   value
   m10-inlet-temperature   value-type             m10-inlet-temperature   value-interval-type
   m10-inlet-temperature   value-precision        m10-inlet-temperature   value-precision-type)

 (equal   m10-inlet-temperature  value-units    m10-outlet-temperature value-units)

 (any-common-portion?
   m10-outlet-temperature  time                   m10-outlet-temperature  time-type
   m10-outlet-temperature  time-interval-type     m10-outlet-temperature  time-precision
   m10-outlet-temperature  time-precision-type    m10-velocity            time
   m10-velocity            time-type              m10-velocity            time-interval-type
   m10-velocity            time-precision         m10-velocity            time-precision-type)

 (any-common-portion?
   m10-inlet-temperature   time                   m10-inlet-temperature   time-type
   m10-inlet-temperature   time-interval-type     m10-inlet-temperature   time-precision
   m10-inlet-temperature   time-precision-type    m10-velocity            time
   m10-velocity            time-type              m10-velocity            time-interval-type
   m10-velocity            time-precision         m10-velocity            time-precision-type)

 (equal   m10-pipe-roughness time-units   m10-outlet-temperature time-units)

 (equal   m10-pipe-roughness time-units   m10-inlet-temperature time-units))
```

D.16.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm10-port-3b 'm10-pipe-roughness 'pipe-2-port-3c 'pipe-2c-roughness)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned: vector #1	vector #2
0	m10-pipe-roughness	req-p	pipe-2c-roughness	pipe-2c-roughness
1	m10-port-3b	p-obj	t312	t312
2	m10-stream-3	p-obj	t313	t313
3	m10-pipe	p-obj	pipe-2	pipe-2
4	m10-edge-2a	p-obj	t321	t320*
5	m10-edge-1b	p-obj	t320	t321*
6	m10-port-3a	p-obj	t303	t303
7	m10-pipe-inside-diameter	req-p	pipe-2-id	pipe-2-id
8	m10-port-2a	p-obj	t317	t314*
9	m10-port-1b	p-obj	t314	t317*
10	m10-fluid-in-pipe	p-obj	t302	t302
11	m10-stream-2	p-obj	t319	t316*
12	m10-stream-1	p-obj	t316	t319*
13	m10-friction-factor	opt-p	nil	nil
14	m10-outlet-temperature	opt-p	nil	nil
15	m10-inlet-temperature	opt-p	nil	nil
16	m10-velocity	opt-p	2d-velocity	nil*
17	m10-density	opt-p	rho-c	rho-c
18	m10-viscosity	opt-p	mu-c	mu-c
19	m10-neg-endpt-2a	neg-1	nil	nil
20	m10-neg-endpt-1b	neg-2	nil	nil
21	m10-neg-edge-3a	neg-3	nil	nil
22	m10-neg-edge-2a	neg-4	nil	nil
23	m10-neg-edge-1b	neg-5	nil	nil
24	m10-neg-port	neg-6	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

SIDE EFFECTS:

Before *After*

D.16 Friction Factor Model and PIPES-1 Equipment Description

D.16.1 Statistics on the Matching

Equipment Description:	PIPES-1
Model Description:	FRICTION-FACTOR
Goal:	find EFFECTS
Max. partial matches:	816
Complete Matches:	2
Rematches:	12
Run time:	1:38 (hr:min)
Objects created (kept):	319 (12)
Links created (kept):	1856 (69)
Separately triggered reconfigurations:	2
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	Yes

D.16.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'pipe-2c-roughness 'value 'describes 'pipe-2-port-3c 'effects)

OUTPUTS:
(
(pipe-2c-roughness value describes pipe-2-port-3c effects m10-pipe-roughness m10-port-3b)
)

D.15.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
0 m7-fluid-temperature	req-p		
1 m7-internal-fluid	p-obj		
2 m7-port-2a	p-obj		
3 m7-port-1b-inlet	p-obj		
4 m7-port-3a-outlet	p-obj		
5 m7-stream-2	p-obj		
6 m7-edge-1b	p-obj		
7 m7-edge-3a	p-obj		
8 m7-stream-1-inlet	p-obj		
9 m7-inlet-pressure	req-p		
10 m7-inlet-flow-rate	req-p		
11 m7-stream-3-outlet	p-obj		
12 m7-port-2b	p-obj		
		part/whole	(p76-static-seal-1 p76-stationary-seal-ring p76-rotating-seal-ring p76-static-seal-2 p76-spring-assembly p76-bearing-assembly p76-shaft p76-impeller p76-casing-wear-ring p76-front-casing p76-back-casing p76-discharge-nozzle p76-suction-nozzle)
13 m7-cent-pump	p-obj		
14 m7-impeller	p-obj		
		part/whole	(p76-casing-wear-ring p76-front-casing p76-back-casing p76-discharge-nozzle p76-suction-nozzle)
15 m7-pump-casing	p-obj		
16 m7-pump-speed	req-p		
17 m7-fluid-density	opt-p		
18 m7-inlet-surface-normal	opt-p		

req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

D.15.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

No condition checking because this isn't a real model and thus doesn't have any conditions.

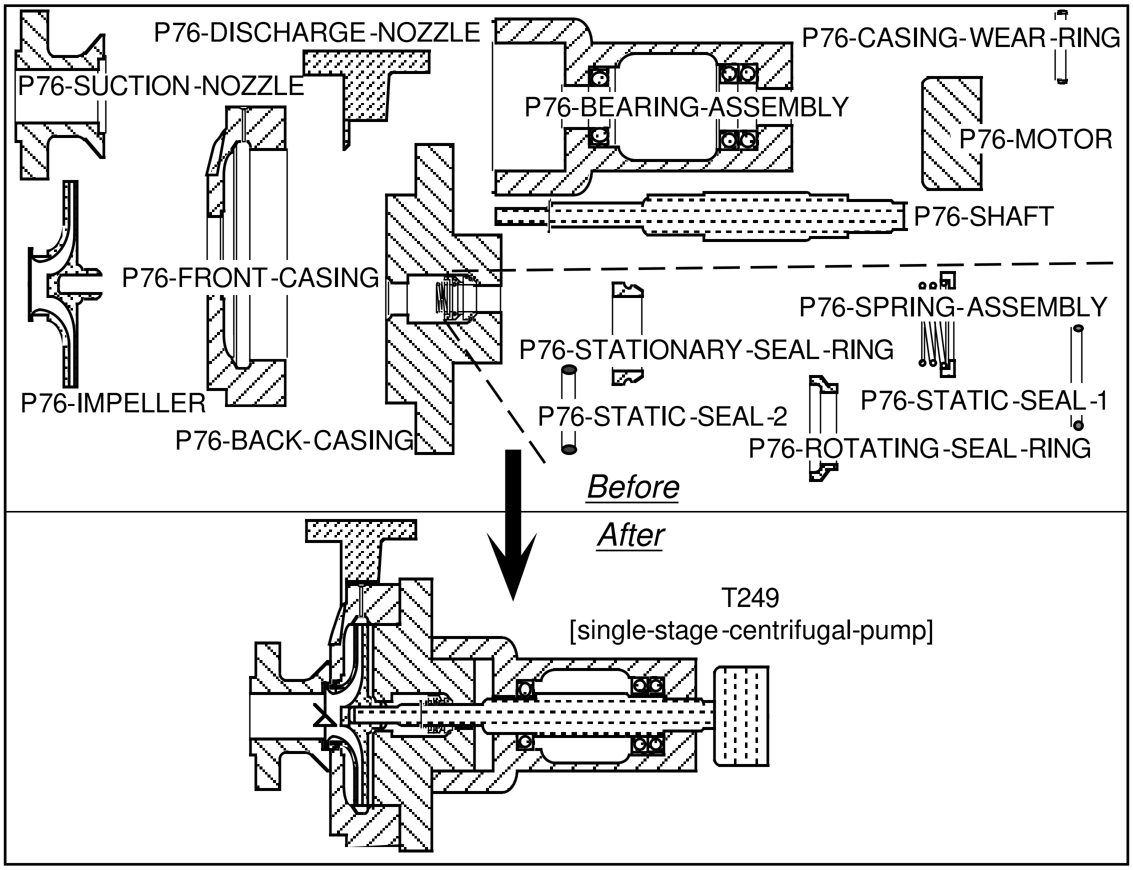
D.15.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

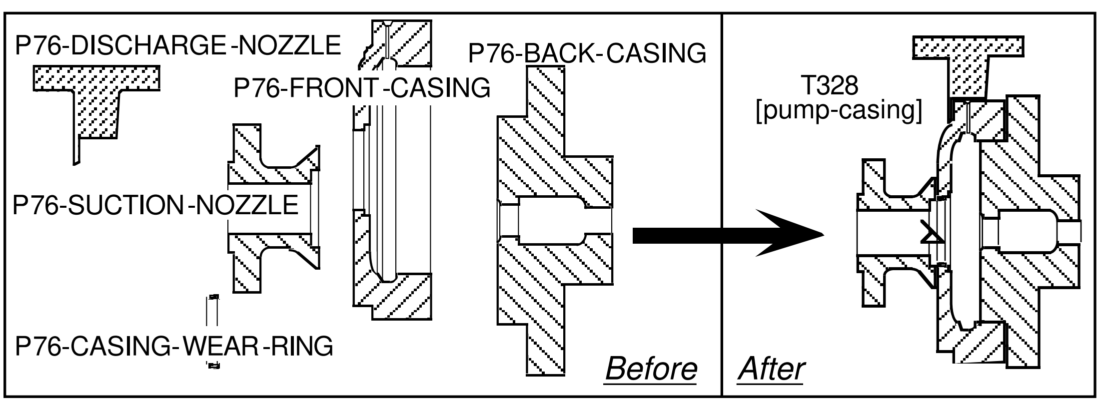
OUTPUTS:
No execution of model because this isn't a real model.

SIDE-EFFECTS: No side effects.

SIDE EFFECTS:



SIDE EFFECTS : continued



Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned: vector #1	vector #2
25	m7-adj-a-neg-stream	neg-2	nil	nil
26	m7-adj-a-neg-port-1	neg-2	nil	nil
27	m7-adj-a-neg-port-2	neg-2	nil	nil
28	m7-pump-neg-port-b	neg-3	nil	nil
29	m7-pump-neg-stream-b	neg-3	nil	nil
30	m7-adj-fluid-b	neg-3	nil	nil
31	m7-adj-b-neg-stream	neg-3	nil	nil
32	m7-adj-b-neg-port-1	neg-3	nil	nil
33	m7-adj-b-neg-port-2	neg-3	nil	nil
34	m7-neg-edge-2 a	neg-4	nil	nil
35	m7-neg-edge-1 b	neg-5	nil	nil
36	m7-neg-edge-3 a	neg-6	nil	nil
37	m7-neg-endpt-1 b	neg-7	nil	nil
38	m7-neg-endpt-3 a	neg-8	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned: vector #3	vector #4
25	m7-adj-a-neg-stream	neg-2	nil	nil
26	m7-adj-a-neg-port-1	neg-2	nil	nil
27	m7-adj-a-neg-port-2	neg-2	nil	nil
28	m7-pump-neg-port-b	neg-3	nil	nil
29	m7-pump-neg-stream-b	neg-3	nil	nil
30	m7-adj-fluid-b	neg-3	nil	nil
31	m7-adj-b-neg-stream	neg-3	nil	nil
32	m7-adj-b-neg-port-1	neg-3	nil	nil
33	m7-adj-b-neg-port-2	neg-3	nil	nil
34	m7-neg-edge-2 a	neg-4	nil	nil
35	m7-neg-edge-1 b	neg-5	nil	nil
36	m7-neg-edge-3 a	neg-6	nil	nil
37	m7-neg-endpt-1 b	neg-7	nil	nil
38	m7-neg-endpt-3 a	neg-8	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

INPUTS:
(DO-SIMPLE-MATCH 'm7-internal-fluid 'm7-fluid-temperature 'p76-liquid 'water-temperature)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #3	vector #4
0	m7-fluid-temperature	req-p	water-temperature	water-temperature
1	m7-internal-fluid	p-obj	t262	t262
2	m7-port-2a	p-obj	t253	t253
3	m7-port-1b-inlet	p-obj	t263*	t366*
4	m7-port-3a-outlet	p-obj	t265*	t265*
5	m7-stream-2	p-obj	t254	t254
6	m7-edge-1b	p-obj	t252*	t252*
7	m7-edge-3a	p-obj	t266*	t266*
8	m7-stream-1-inlet	p-obj	stream-liquid-182liq*	stream-liquid-182liq*
9	m7-inlet-pressure	req-p	pump-inlet-pressure*	pump-inlet-pressure*
10	m7-inlet-flow-rate	req-p	water-flow*	water-flow*
11	m7-stream-3-outlet	p-obj	stream-liquid-183liq*	stream-liquid-183liq*
12	m7-port-2b	p-obj	t250	t250
13	m7-cent-pump	p-obj	t249	t249
14	m7-impeller	p-obj	p76-impeller	p76-impeller
15	m7-pump-casing	p-obj	t328	t328
16	m7-pump-speed	req-p	pump-rpm	pump-rpm
17	m7-fluid-density	opt-p	nil	nil
18	m7-inlet-surface-normal	opt-p	nil	nil
19	m7-npsh	opt-p	nil	nil
20	m7-outlet-surface-normal	opt-p	nil	nil
21	m7-int-water-neg-port	neg-1	nil	nil
22	m7-pump-neg-port-a	neg-2	nil	nil
23	m7-pump-neg-stream-a	neg-2	nil	nil
24	m7-adj-fluid-a	neg-2	nil	nil

* different than vector #1.
req-p -- required parameter
p-obj -- positive object
opt-p -- optional parameter
neg-n -- member of the nth negative set matched

D.15.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm7-internal-fluid 'm7-fluid-temperature 'p76-liquid 'water-temperature)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m7-fluid-temperature	req-p	water-temperature	water-temperature
1	m7-internal-fluid	p-obj	t262	t262
2	m7-port-2a	p-obj	t253	t253
3	m7-port-1b-inlet	p-obj	t265	t368*
4	m7-port-3a-outlet	p-obj	t263	t263
5	m7-stream-2	p-obj	t254	t254
6	m7-edge-1b	p-obj	t251	t251
7	m7-edge-3a	p-obj	t264	t264
8	m7-stream-1-inlet	p-obj	stream-liquid-183liq	stream-liquid-183liq
9	m7-inlet-pressure	req-p	pump-outlet-pressure	pump-outlet-pressure
10	m7-inlet-flow-rate	req-p	pump-output-flow	pump-output-flow
11	m7-stream-3-outlet	p-obj	stream-liquid-182liq	stream-liquid-182liq
12	m7-port-2b	p-obj	t250	t250
13	m7-cent-pump	p-obj	t249	t249
14	m7-impeller	p-obj	p76-impeller	p76-impeller
15	m7-pump-casing	p-obj	t328	t328
16	m7-pump-speed	req-p	pump-rpm	pump-rpm
17	m7-fluid-density	opt-p	nil	nil
18	m7-inlet-surface-normal	opt-p	nil	nil
19	m7-npsh	opt-p	nil	nil
20	m7-outlet-surface-normal	opt-p	nil	nil
21	m7-int-water-neg-port	neg-1	nil	nil
22	m7-pump-neg-port-a	neg-2	nil	nil
23	m7-pump-neg-stream-a	neg-2	nil	nil
24	m7-adj-fluid-a	neg-2	nil	nil

* different than vector #1.
req-p -- required parameter
p-obj -- positive object
opt-p -- optional parameter
neg-n -- member of the nth negative set matched

D.15 Hypothetical Model and DETAILED-PUMP Equipment Description

D.15.1 Statistics on the Matching

Equipment Description:	DETAILED-PUMP
Model Description:	HYPOTHETICAL
Goal:	find EFFECTS
Max. partial matches:	52
Complete Matches:	4
Rematches:	2
Run time:	0:18 (hr:min)
Objects created (kept):	379 (85)
Links created (kept):	2612 (449)
Separately triggered reconfigurations:	2
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.15.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'water-temperature 'value 'describes 'p76-liquid 'effects)

OUTPUTS:
```
(
 (water-temperature value describes p76-liquid effects m3-water-temperature m3-internal-water)
 (water-temperature value describes p76-liquid effects m7-fluid-temperature m7-internal-fluid)
)
```

D.14.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m6-suction-pressure	req-p	part/whole	(p76-front-casing p76-discharge-nozzle p76-suction-nozzle p76-back-casing p76-casing-wear-ring p76-static-seal-1 p76-stationary-seal-ring p76-rotating-seal-ring p76-static-seal-2 p76-spring-assembly p76-bearing-assembly p76-shaft p76-impeller)
1	m6-port-liq-inlet	p-obj		
2	m6-liquid	p-obj		
3	m6-edge-liq-inlet	p-obj		
4	m6-flow-rate	req-p		
5	m6-suction-nozzle-id	req-p		
6	m6-port-liq-outlet	p-obj		
7	m6-port-liq-pump	p-obj		
8	m6-specific-gravity	req-p		
9	m6-port-pump-liq	p-obj		
10	m6-edge-liq-outlet	p-obj		
11	m6-discharge-nozzle-id	req-p		
12	m6-discharge-pressure	req-p		
13	m6-stream-pump-liq	p-obj		
14	m6-pump	p-obj		
15	m6-hydraulic-horsepower	opt-p		
16	m6-neg-edge-liq-inlet	neg-1		
17	m6-adj-a-neg-stream	neg-2		
18	m6-pump-neg-port-a	neg-2		
19	m6-pump-neg-stream-a	neg-2		
20	m6-adjacent-liq-a	neg-2		
21	m6-adj-a-neg-port-1	neg-2		
22	m6-adj-a-neg-port-2	neg-2		
23	m6-neg-port-liq	neg-3		
24	m6-neg-edge-liq-outlet	neg-4		
25	m6-adj-b-neg-stream	neg-5		
26	m6-pump-neg-port-b	neg-5		
27	m6-pump-neg-stream-b	neg-5		
28	m6-adjacent-liq-b	neg-5		
29	m6-adj-b-neg-port-1	neg-5		
30	m6-adj-b-neg-port-2	neg-5		

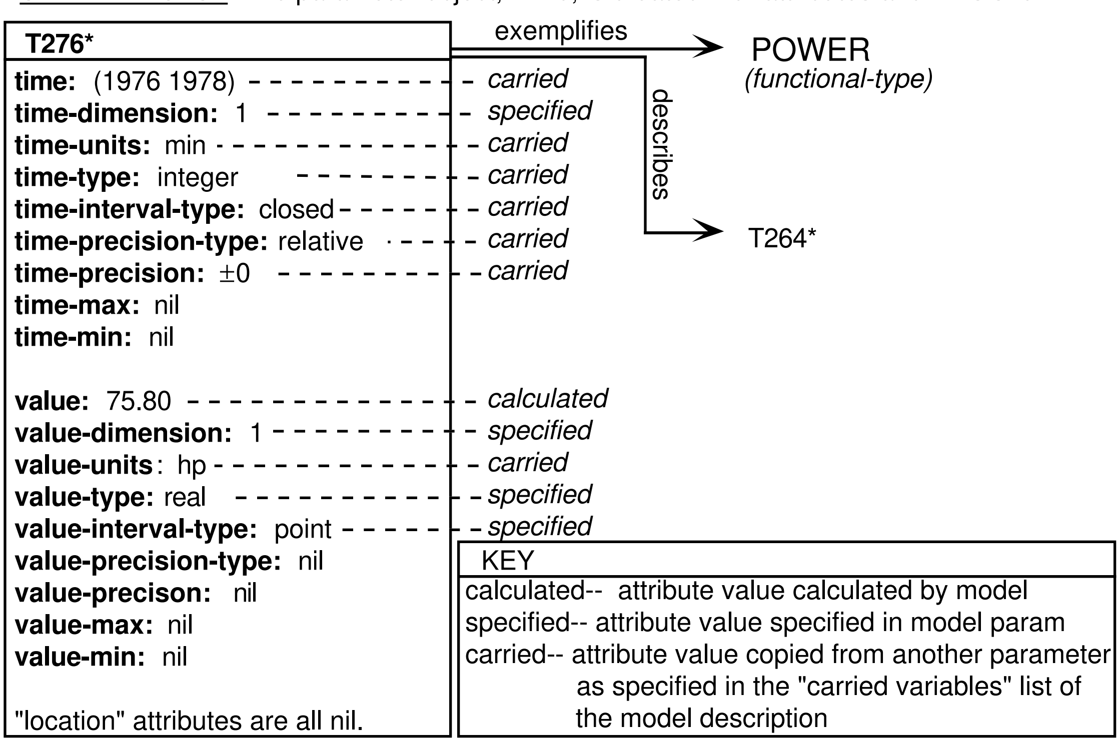
req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

D.14.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

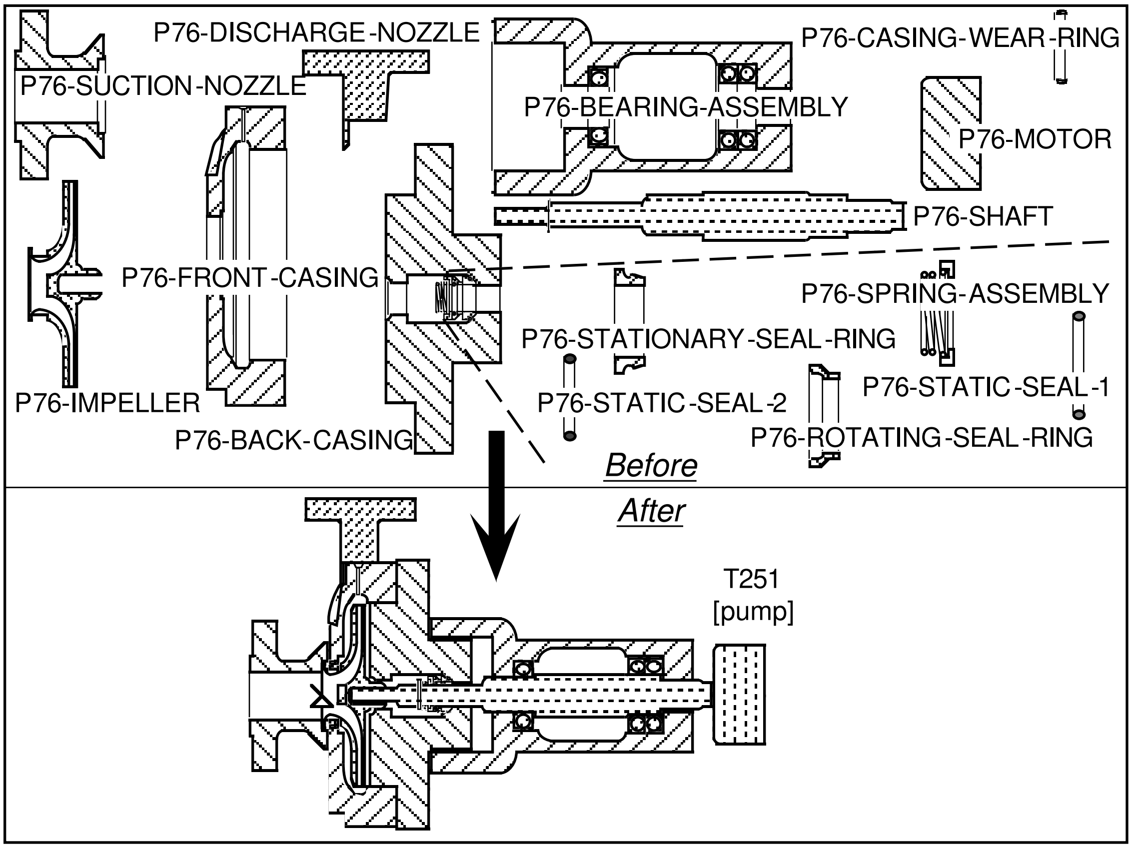
OUTPUTS:
(T276)

SIDE-EFFECTS: The parameter object, T276, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

SIDE EFFECTS:



D.14.4 Checking Model Conditions

INPUTS:

(CHECK-MODEL-CONDITIONS approx-conditions assoc-list (first vector-list))

OUTPUTS:

T

NIL

D.14.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm6-port-liq-inlet 'm6-suction-pressure 'port-liq-13 'pump-inlet-pressure)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m6-suction-pressure	req-p	pump-inlet-pressure	
1	m6-port-liq-inlet	p-obj	t265	
2	m6-liquid	p-obj	t264	
3	m6-edge-liq-inlet	p-obj	t266	
4	m6-flow-rate	req-p	water-flow	
5	m6-suction-nozzle-id	req-p	pump-inlet-diameter	
6	m6-port-liq-outlet	p-obj	t267	
7	m6-port-liq-pump	p-obj	t255	
8	m6-specific-gravity	req-p	water-specific-gravity	
9	m6-port-pump-liq	p-obj	t252	
10	m6-edge-liq-outlet	p-obj	t268	
11	m6-discharge-nozzle-id	req-p	pump-outlet-diameter	
12	m6-discharge-pressure	req-p	pump-outlet-pressure	
13	m6-stream-pump-liq	p-obj	t256	
14	m6-pump	p-obj	t251	
15	m6-hydraulic-horsepower	opt-p	nil	
16	m6-neg-edge-liq-inlet	neg-1	nil	
17	m6-adj-a-neg-stream	neg-2	nil	
18	m6-pump-neg-port-a	neg-2	nil	
19	m6-pump-neg-stream-a	neg-2	nil	
20	m6-adjacent-liq-a	neg-2	nil	
21	m6-adj-a-neg-port-1	neg-2	nil	
22	m6-adj-a-neg-port-2	neg-2	nil	
23	m6-neg-port-liq	neg-3	nil	
24	m6-neg-edge-liq-outlet	neg-4	nil	
25	m6-adj-b-neg-stream	neg-5	nil	
26	m6-pump-neg-port-b	neg-5	nil	
27	m6-pump-neg-stream-b	neg-5	nil	
28	m6-adjacent-liq-b	neg-5	nil	
29	m6-adj-b-neg-port-1	neg-5	nil	
30	m6-adj-b-neg-port-2	neg-5	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.14 Hydraulic Horsepower Model and DETAILED-PUMP Equipment

D.14.1 Statistics on the Matching

Equipment Description:	DETAILED-PUMP
Model Description:	HYDRAULIC-HORSEPOWER
Goal:	find EFFECTS
Max. partial matches:	59
Complete Matches:	1
Rematches:	1
Run time:	0:11 (hr:min)
Objects created (kept):	273 (31)
Links created (kept):	1851 (194)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.14.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS 'pump-inlet-pressure 'value 'describes 'port-liq-13 'effects)
```

OUTPUTS:

```
(
(pump-inlet-pressure value describes port-liq-13 effects m6-suction-pressure m6-port-liq-inlet)
(pump-inlet-pressure value describes port-liq-13 effects m6-discharge-pressure
        m6-port-liq-outlet)
)
```

D.13.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
0 m5-power-2	req-p		
1 m5-motor	p-obj		
2 m5-port-motor-pump	p-obj		
3 m5-power-1	req-p		
4 m5-stream-pump-motor	p-obj		
5 m5-port-pump-motor	p-obj		
		part/whole	(p76-shaft p76-stationary-seal-ring p76-rotating-seal-ring p76-static-seal-2 p76-static-seal-1 p76-spring-assembly p76-impeller p76-bearing-assembly p76-back-casing p76-front-casing p76-casing-wear-ring p76-discharge-nozzle p76-suction-nozzle)
6 m5-c-pump	p-obj		
7 m5-port-pump-liq	p-obj		
8 m5-imp-wear-ring	p-obj		
9 m5-cas-wear-ring	p-obj		
10 m5-impeller	p-obj		
11 m5-stream-pump-liq	p-obj		
12 m5-edge-fluid-inlet	p-obj		
13 m5-edge-fluid-outlet	p-obj		
14 m5-i-wear-ring-od	req-p		
15 m5-c-wear-ring-id	req-p		
16 m5-port-liq-pump	p-obj		
17 m5-port-fluid-inlet	p-obj		
18 m5-port-fluid-outlet	p-obj		
19 m5-fluid	p-obj		
20 m5-clearance	opt-p		
21 m5-rpm-1	opt-p		
22 m5-rpm-2	opt-p		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

D.13.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
(T4)

SIDE-EFFECTS: The parameter object, T4, is created with attributes and links shown.

T4*

exemplifies → LENGTH *(functional-type)*

describes → P76-CASING-WEAR-RING

time: 28010	*carried*
time-dimension: 1	*carried*
time-units: min	*carried*
time-type: integer	*carried*
time-interval-type: point	*carried*
time-precision-type: absolute	*carried*
time-precision: ±1	*carried*
time-max: nil	
time-min: nil	
value: 0.03887	*calculated*
value-dimension: 1	*specified*
value-units: in	*carried*
value-type: real	*specified*
value-interval-type: point	*specified*
value-precision-type: nil	
value-precison: nil	
value-max: nil	
value-min: nil	

"location" attributes are all nil.

KEY
calculated-- attribute value calculated by model
specified-- attribute value specified in model param
carried-- attribute value copied from another parameter as specified in the "carried variables" list of the model description

* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

```
(any-common-portion?
  m5-power-1  time
  m5-power-1  time-type
  m5-power-1  time-interval-type
  m5-power-1  time-precision
  m5-power-1  time-precision-type
  m5-rho-1    time
  m5-rho-1    time-type
  m5-rho-1    time-interval-type
  m5-rho-1    time-precision
  m5-rho-1    time-precision-type)

(any-common-portion?
  m5-power-1  time
  m5-power-1  time-type
  m5-power-1  time-interval-type
  m5-power-1  time-precision
  m5-power-1  time-precision-type
  m5-mu-1     time
  m5-mu-1     time-type
  m5-mu-1     time-interval-type
  m5-mu-1     time-precision
  m5-mu-1     time-precision-type)

(any-common-portion?
  m5-rpm-1    value
  m5-rpm-1    value-type
  m5-rpm-1    value-interval-type
  m5-rpm-1    value-precision
  m5-rpm-1    value-precision-type
  m5-rpm-2    value
  m5-rpm-2    value-type
  m5-rpm-2    value-interval-type
  m5-rpm-2    value-precision
  m5-rpm-2    value-precision-type)

(any-common-portion?
  m5-rho-1    value
  m5-rho-1    value-type
  m5-rho-1    value-interval-type
  m5-rho-1    value-precision
  m5-rho-1    value-precision-type
  m5-rho-2    value
  m5-rho-2    value-type
  m5-rho-2    value-interval-type
  m5-rho-2    value-precision
  m5-rho-2    value-precision-type)

(any-common-portion?
  m5-mu-1     value
  m5-mu-1     value-type
  m5-mu-1     value-interval-type
  m5-mu-1     value-precision
  m5-mu-1     value-precision-type
  m5-mu-2     value
  m5-mu-2     value-type
  m5-mu-2     value-interval-type
  m5-mu-2     value-precision
  m5-mu-2     value-precision-type)

(equal m5-rpm-1 value-units m5-rpm-2 value-units)

(equal m5-mu-1 value-units m5-mu-2 value-units)

(equal m5-rho-1 value-units m5-rho-2 value-units))
```

(**any-common-portion?**

m5-rho-1	value	m5-rho-1	value-type
m5-rho-1	value-interval-type	m5-rho-1	value-precision
m5-rho-1	value-precision-type	m5-rho-2	value
m5-rho-2	value-type	m5-rho-2	value-interval-type
m5-rho-2	value-precision	m5-rho-2	value-precision-type)

(**any-common-portion?**

m5-mu-1	value	m5-mu-1	value-type
m5-mu-1	value-interval-type	m5-mu-1	value-precision
m5-mu-1	value-precision-type	m5-mu-2	value
m5-mu-2	value-type	m5-mu-2	value-interval-type
m5-mu-2	value-precision	m5-mu-2	value-precision-type)

(**equal** m5-rpm-1 value-units m5-rpm-2 value-units)

(**equal** m5-mu-1 value-units m5-mu-2 value-units)

(**equal** m5-rho-1 value-units m5-rho-2 value-units))

INPUTS:

(CHECK-MODEL-CONDITIONS approx-conditions assoc-list (second vector-list))

OUTPUTS:

T

((**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-rpm-2	time
m5-rpm-2	time-type	m5-rpm-2	time-interval-type
m5-rpm-2	time-precision	m5-rpm-2	time-precision-type)

(**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-rho-2	time
m5-rho-2	time-type	m5-rho-2	time-interval-type
m5-rho-2	time-precision	m5-rho-2	time-precision-type)

(**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-mu-2	time
m5-mu-2	time-type	m5-mu-2	time-interval-type
m5-mu-2	time-precision	m5-mu-2	time-precision-type)

D.13.4 Checking Model Conditions

INPUTS:

(CHECK-MODEL-CONDITIONS approx-conditions assoc-list (first vector-list))

OUTPUTS:

T

((**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-rpm-2	time
m5-rpm-2	time-type	m5-rpm-2	time-interval-type
m5-rpm-2	time-precision	m5-rpm-2	time-precision-type)

(**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-rho-2	time
m5-rho-2	time-type	m5-rho-2	time-interval-type
m5-rho-2	time-precision	m5-rho-2	time-precision-type)

(**any-common-portion?**

m5-power-2	time	m5-power-2	time-type
m5-power-2	time-interval-type	m5-power-2	time-precision
m5-power-2	time-precision-type	m5-mu-2	time
m5-mu-2	time-type	m5-mu-2	time-interval-type
m5-mu-2	time-precision	m5-mu-2	time-precision-type)

(**any-common-portion?**

m5-power-1	time	m5-power-1	time-type
m5-power-1	time-interval-type	m5-power-1	time-precision
m5-power-1	time-precision-type	m5-rho-1	time
m5-rho-1	time-type	m5-rho-1	time-interval-type
m5-rho-1	time-precision	m5-rho-1	time-precision-type)

(**any-common-portion?**

m5-power-1	time	m5-power-1	time-type
m5-power-1	time-interval-type	m5-power-1	time-precision
m5-power-1	time-precision-type	m5-mu-1	time
m5-mu-1	time-type	m5-mu-1	time-interval-type
m5-mu-1	time-precision	m5-mu-1	time-precision-type)

(**any-common-portion?**

m5-rpm-1	value	m5-rpm-1	value-type
m5-rpm-1	value-interval-type	m5-rpm-1	value-precision
m5-rpm-1	value-precision-type	m5-rpm-2	value
m5-rpm-2	value-type	m5-rpm-2	value-interval-type
m5-rpm-2	value-precision	m5-rpm-2	value-precision-type)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
25	m5-mu-2	opt-p	nil	nil
26	m5-mu-1	opt-p	nil	nil
27	m5-pump-neg-port-a	neg-1	nil	nil
28	m5-pump-neg-stream-a	neg-1	nil	nil
29	m5-adjacent-fluid-a	neg-1	nil	nil
30	m5-adj-a-neg-port-1	neg-1	nil	nil
31	m5-adj-a-neg-port-2	neg-1	nil	nil
32	m5-adj-a-neg-stream	neg-1	nil	nil
33	m5-pump-neg-port-b	neg-2	nil	nil
34	m5-pump-neg-stream-b	neg-2	nil	nil
35	m5-adjacent-fluid-b	neg-2	nil	nil
36	m5-adj-b-neg-port-1	neg-2	nil	nil
37	m5-adj-b-neg-port-2	neg-2	nil	nil
38	m5-adj-b-neg-stream	neg-2	nil	nil
39	m5-neg-edge-fluid-inlet	neg-3	nil	nil
40	m5-neg-edge-fluid-outlet	neg-4	nil	nil
41	m5-neg-port-fluid	neg-5	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.13.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm5-motor 'm5-power 'p76-motor 'power-input-72)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m5-power-2	req-p	power-input-72	power-input-72
1	m5-motor	p-obj	p76-motor	p76-motor
2	m5-port-motor-pump	p-obj	port-motor-pump	port-motor-pump
3	m5-power-1	req-p	power-input-1	power-input-1
4	m5-stream-pump-motor	p-obj	stream-motor-pump	stream-motor-pump
5	m5-port-pump-motor	p-obj	t231	t231
6	m5-c-pump	p-obj	t217	t217
7	m5-port-pump-liq	p-obj	t223	t223
8	m5-imp-wear-ring	p-obj	p76-imp-wear-ring	p76-imp-wear-ring
9	m5-cas-wear-ring	p-obj	p76-casing-wear-ring	p76-casing-wear-ring
10	m5-impeller	p-obj	p76-impeller	p76-impeller
11	m5-stream-pump-liq	p-obj	t226	t226
12	m5-edge-fluid-inlet	p-obj	t219	t224
13	m5-edge-fluid-outlet	p-obj	t224	t219
14	m5-i-wear-ring-od	req-p	impeller-wear-ring-od	impeller-wear-ring-od
15	m5-c-wear-ring-id	req-p	casing-wear-ring-id	casing-wear-ring-id
16	m5-port-liq-pump	p-obj	t225	t225
17	m5-port-fluid-inlet	p-obj	t235	t237
18	m5-port-fluid-outlet	p-obj	t237	t235
19	m5-fluid	p-obj	t234	t234
20	m5-clearance	opt-p	nil	nil
21	m5-rpm-1	opt-p	pump-rpm	pump-rpm
22	m5-rpm-2	opt-p	nil	nil
23	m5-rho-2	opt-p	nil	nil
24	m5-rho-1	opt-p	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

Note # 1: This part/whole reconfiguration is not triggered by inability to find an equipment object to match M3-PORT-2B, but by the attmpt to find a "has-edge" link from objects maching M3-PORT-2A (4) to objects matching M3-EDGE-1B (8). At this point the matcher was finding all objects linked to M3-PORT-2A. It successfully found and matched M3-STREAM-2, but could not correctly match the "has-edge" link to the objects matching M3-EDGE-1B, which was already found as edge of the port M3-PORT-3A-OUTLET. The surface of the liquid region adjacent to the pump (M3-PORT-2B) must share edges with the ends of the outlet port (M3-PORT-3A-OUTLET), and this condition is not met. Some surface to a pump part has been matched to M3-PORT-2B, and to get the correct surface, the part/whole reconfiguration has to be made.

D.13 Wear Ring Model and DETAILED-PUMP Equipment Description

D.13.1 Statistics on the Matching

Equipment Description:	DETAILED-PUMP
Model Description:	WEAR-RING
Goal:	find CAUSES
Max. partial matches:	13
Complete Matches:	2
Rematches:	1
Run time:	0:10 (hr:min)
Objects created (kept):	261 (50)
Links created (kept):	1546 (275)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.13.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'power-input-72 'value 'describes 'p76-motor 'causes)

OUTPUTS:
(
(power-input-72 value describes p76-motor causes m5-power-2 m5-motor)
)

D.12.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	notes	Reconfig type	Reconfigurations tried
0 m3-water-temperature	req-p		
1 m3-internal-water	p-obj		
2 m3-port-1b-inlet	p-obj		
3 m3-port-3a-outlet	p-obj		
4 m3-port-2a	p-obj		
5 m3-stream-1-inlet	p-obj		
6 m3-inlet-pressure	req-p		
7 m3-inlet-flow-rate	req-p		
8 m3-edge-1b	p-obj		
9 m3-stream-3-outlet	p-obj		
10 m3-edge-3a	p-obj		
11 m3-stream-2	p-obj	part/whole-- note #1	(p76-front-casing p76-casing-wear-ring p76-discharge-nozzle p76-impeller p76-suction-nozzle p76-static-seal-2 p76-back-casing p76-static-seal-1 p76-stationary-seal-ring p76-shaft p76-rotating-seal-ring p76-spring-assembly p76-bearing-assembly)
12 m3-port-2b	p-obj		
13 m3-centrifugal-pump	p-obj		
14 m3-s-s-impeller	p-obj		
15 m3-pump-speed	req-p		
16 m3-water-density	opt-p		
17 m3-npsh	opt-p		
18 m3-inlet-surface-normal	opt-p		
19 m3-outlet-surface-normal	opt-p		
20 m3-int-water-neg-port	neg-1		
21 m3-pump-neg-port-a	neg-2		
22 m3-pump-neg-stream-a	neg-2		
23 m3-adjacent-water-a	neg-2		
24 m3-adj-water-a-neg-port-1	neg-2		
25 m3-adj-water-a-neg-port-2	neg-2		
26 m3-adj-water-a-neg-stream	neg-2		
27 m3-neg-edge-1b	neg-3		
28 m3-pump-neg-port-b	neg-4		
29 m3-pump-neg-stream-b	neg-4		
30 m3-adjacent-water-b	neg-4		
31 m3-adj-water-b-neg-port-1	neg-4		
32 m3-adj-water-b-neg-port-2	neg-4		
33 m3-adj-water-b-neg-stream	neg-4		
34 m3-neg-edge-3a	neg-5		
35 m3-neg-edge-2a	neg-6		
36 m3-neg-endpt-1b	neg-7		
37 m3-neg-endpt-3a	neg-8		

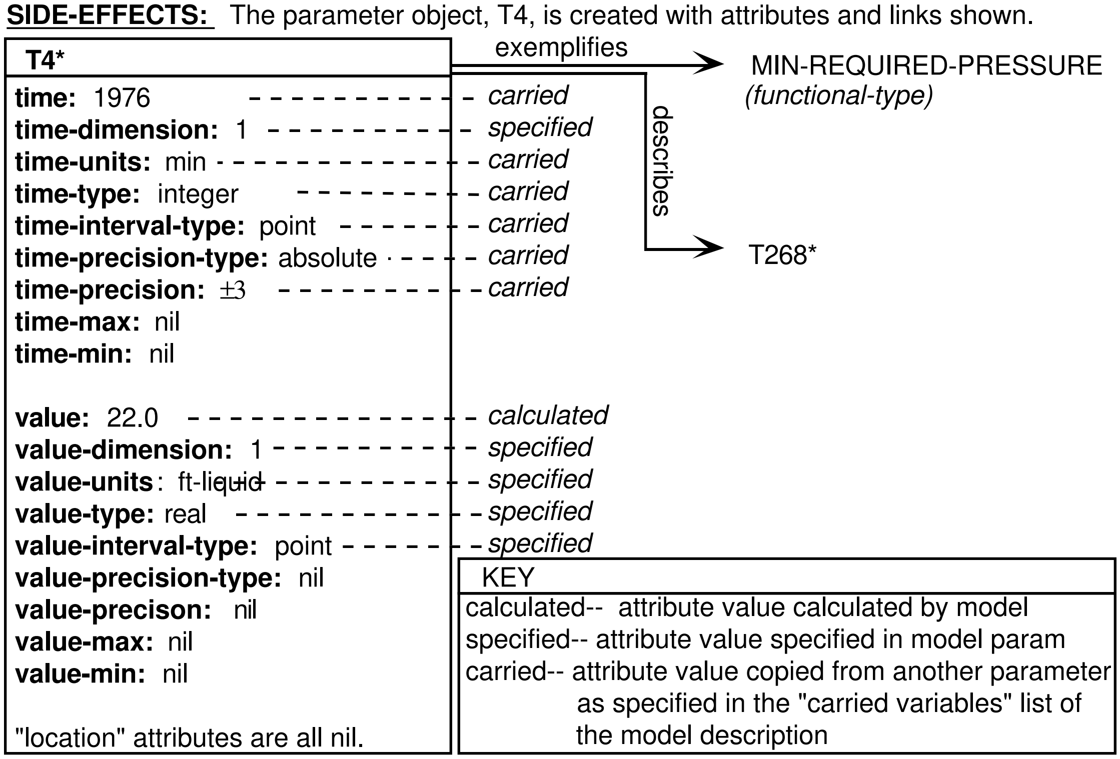
req-p -- required parameter　　opt-p -- optional parameter
p-obj -- positive object　　neg-n -- member of the nth negative set matched

D.12.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
(T4)

SIDE-EFFECTS: The parameter object, T4, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

D.12.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS approx-conditions assoc-list (first vector-list))

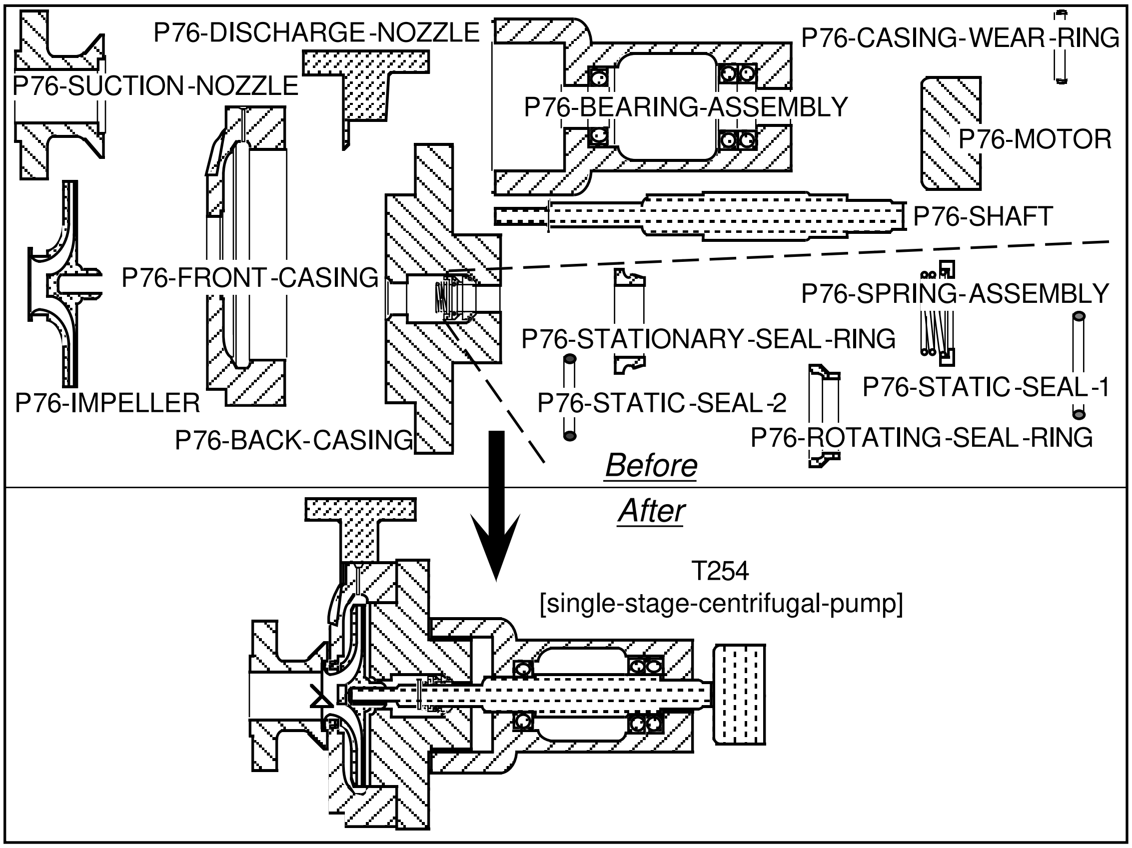
OUTPUTS:
T

```
((any-common-portion?
   m3-inlet-flow-rate     time                   m3-inlet-flow-rate     time-type
   m3-inlet-flow-rate     time-interval-type     m3-inlet-flow-rate     time-precision
   m3-inlet-flow-rate     time-precision-type    m3-water-density       time
   m3-water-density       time-type              m3-water-density       time-interval-type
   m3-water-density       time-precision         m3-water-density       time-precision-type)

 (any-common-portion?
   m3-inlet-flow-rate     time                   m3-inlet-flow-rate     time-type
   m3-inlet-flow-rate     time-interval-type     m3-inlet-flow-rate     time-precision
   m3-inlet-flow-rate     time-precision-type    m3-npsh                time
   m3-npsh                time-type              m3-npsh                time-interval-type
   m3-npsh                time-precision         m3-npsh                time-precision-type)

 (m3-hw-pump::vector-values-dont-overlap?
   m3-outlet-surface-normal  value            m3-outlet-surface-normal  value-type
   m3-outlet-surface-normal  interval-type    m3-outlet-surface-normal  value-dimension
   m3-outlet-surface-normal  value-precision  m3-outlet-surface-normal  value-precision-type
   nil                       (0 0 0)          nil                       real
   nil                       point            nil                       3
   nil                       0                nil                       absolute)

 (m3-hw-pump::vector-values-dont-overlap?
   m3-inlet-surface-normal  value            m3-inlet-surface-normal  value-type
   m3-inlet-surface-normal  interval-type    m3-inlet-surface-normal  value-dimension
   m3-inlet-surface-normal  value-precision  m3-inlet-surface-normal  value-precision-type
   nil                      (0 0 0)          nil                      real
   nil                      point            nil                      3
   nil                      0                nil                      absolute)

 (m3-hw-pump::time-type-real-or-integer? m3-npsh time-type)

 (m3-hw-pump::time-type-real-or-integer? m3-water-density time-type)

 (equal m3-inlet-flow-rate time-units m3-npsh time-units)

 (equal m3-inlet-flow-rate time-units m3-water-density time-units))
```

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (second vector-list))

OUTPUTS:
nil

SIDE EFFECTS:



D.12.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm3-internal-water 'm3-water-temperature 'p76-liquid 'water-temperature)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m3-water-temperature	req-p	water-temperature	water-temperature
1	m3-internal-water	p-obj	t267	t267
2	m3-port-1b-inlet	p-obj	t268	t270
3	m3-port-3a-outlet	p-obj	t270	t268
4	m3-port-2a	p-obj	t258	t258
5	m3-stream-1-inlet	p-obj	stream-liquid-182liq	stream-liquid-183liq
6	m3-inlet-pressure	req-p	pump-inlet-pressure	pump-outlet-pressure
7	m3-inlet-flow-rate	req-p	water-flow	pump-output-flow
8	m3-edge-1b	p-obj	t269	t271
9	m3-stream-3-outlet	p-obj	stream-liquid-183liq	stream-liquid-182liq
10	m3-edge-3a	p-obj	t271	t269
11	m3-stream-2	p-obj	t259	t259
12	m3-port-2b	p-obj	t255	t255
13	m3-centrifugal-pump	p-obj	t254	t254
14	m3-s-s-impeller	p-obj	p76-impeller	p76-impeller
15	m3-pump-speed	req-p	pump-rpm	pump-rpm
16	m3-water-density	opt-p	nil	nil
17	m3-npsh	opt-p	nil	nil
18	m3-inlet-surface-normal	opt-p	nil	nil
19	m3-outlet-surface-normal	opt-p	nil	nil
20	m3-int-water-neg-port	neg-1	nil	nil
21	m3-pump-neg-port-a	neg-2	nil	nil
22	m3-pump-neg-stream-a	neg-2	nil	nil
23	m3-adjacent-water-a	neg-2	nil	nil
24	m3-adj-water-a-neg-port-1	neg-2	nil	nil
25	m3-adj-water-a-neg-port-2	neg-2	nil	nil
26	m3-adj-water-a-neg-stream	neg-2	nil	nil
27	m3-neg-edge-1b	neg-3	nil	nil
28	m3-pump-neg-port-b	neg-4	nil	nil
29	m3-pump-neg-stream-b	neg-4	nil	nil
30	m3-adjacent-water-b	neg-4	nil	nil
31	m3-adj-water-b-neg-port-1	neg-4	nil	nil
32	m3-adj-water-b-neg-port-2	neg-4	nil	nil
33	m3-adj-water-b-neg-stream	neg-4	nil	nil
34	m3-neg-edge-3a	neg-5	nil	nil
35	m3-neg-edge-2a	neg-6	nil	nil
36	m3-neg-endpt-1b	neg-7	nil	nil
37	m3-neg-endpt-3a	neg-8	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.12 NPSH Model and DETAILED-PUMP Equipment Description

D.12.1 Statistics on the Matching

Equipment Description:	DETAILED-PUMP
Model Description:	NPSH
Goal:	find EFFECTS
Max. partial matches:	24
Complete Matches:	2
Rematches:	1
Run time:	0:08 (hr:min)
Objects created (kept):	244 (28)
Links created (kept):	1594 (164)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.12.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'water-temperature 'value 'describes 'p76-liquid 'effects)

OUTPUTS:
(water-temperature value describes p76-liquid effects m3-water-temperature m3-internal-water)

D.11.6 Summary of Merges and Splits Tested

	Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m8-tempout-after	req-p		
1	m8-port-liq-outlet	p-obj		
2	m8-edge-outlet	p-obj		
3	m8-tempout-before	req-p		
4	m8-liquid	p-obj		
5	m8-port-nozzle-liquid	p-obj		
6	m8-port-liq-1-inlet	p-obj		
7	m8-port-liq-2-inlet	p-obj		
8	m8-port-liq-impeller	p-obj		
9	m8-height-before	req-p		
10	m8-outlet-nozzle	p-obj		
11	m8-coldflow-after	req-p		
12	m8-coldflow-before	req-p		
13	m8-hotflow-after	req-p		
14	m8-hotflow-before	req-p		
15	m8-stream-liq-impeller	p-obj		
			part/whole	(r14-tank-floor r14-outlet-nozzle r14-tank-wall r14-tank-top r14-inlet-nozzle-1 r14-inlet-nozzle-2 r14-seal r14-shaft r14-impeller-blade r14-motor)
16	m8-hot-cold-mixer	p-obj		
17	m8-port-nozzle-floor	p-obj		
18	m8-port-impeller-liquid	p-obj		
19	m8-tank-floor	p-obj		
			part/whole	(r14-seal r14-shaft r14-impeller-blade)
20	m8-impeller	p-obj		
21	m8-stream-floor-outlet	p-obj		
22	m8-port-floor-outnoz	p-obj		
23	m8-outflow-after	opt-p		
⋮	...	...		
39	m8-neg-port-liquid	neg-1		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

<u>SIDE-EFFECTS:</u> (continued) The parameter object, outlet-temperature-2771, is modified with attributes and links shown.

OUTLET-TEMPERATURE-2771 —exemplifies→ TEMPERATURE *(functional-type)*

OUTLET-TEMPERATURE-2771 —describes→ T367*

OUTLET-TEMPERATURE-2771	
time: t1	*calculated*
time-dimension: 1	*specified*
time-units: min	*carried*
time-type: qlandmark	*specified*
time-interval-type: closed	*carried*
time-precision-type: (t0 t1 inf)	*calculated*
time-precision: nil	
time-max: nil	
time-min: nil	
value: te-2	*calculated*
value-dimension: 1	*specified*
value-units: nil	*carried*
value-type: qlandmark	*specified*
value-interval-type: point	*specified*
value-precision-type: (0 t-cold te-2 t-norm t-high inf)	*calculated*
value-precison: nil	
value-max: nil	
value-min: nil	
"location" attributes are all nil.	

KEY
calculated-- attribute value calculated by model
specified-- attribute value specified in model param
carried-- attribute value copied from another parameter as specified in the "carried variables" list of the model description

* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

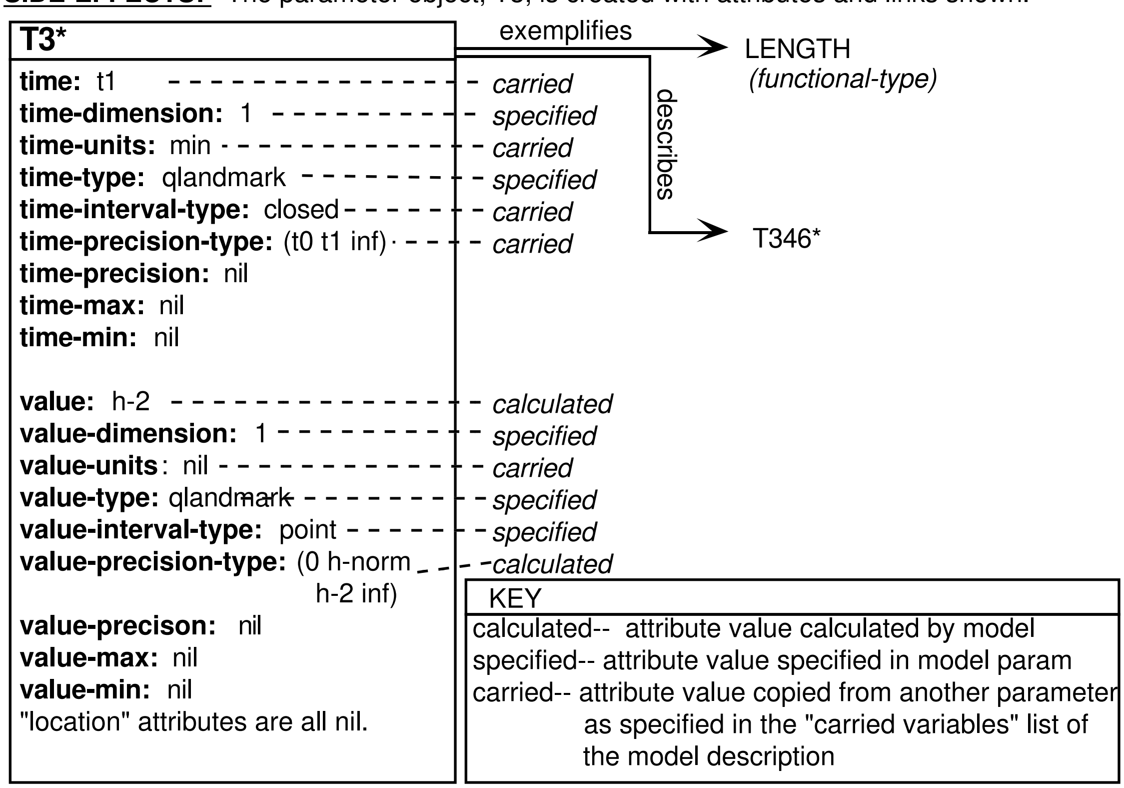
D.11.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS: [note: The list of QSIM files loaded is printed out to the screen is identical to the list shown in the CSTR-1 matching case.]

(OUTLET-TEMPERATURE-2771 T3)

SIDE-EFFECTS: The parameter object, T3, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

D.11.4 Checking Model Conditions

INPUTS: Vector #1
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS: Vector #1
T
[The list of unverified conditions returned is identical to the list returned in the CSTR-1 matching.]

INPUTS: Vector #5
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (fifth vector-list))

OUTPUTS: Vector #5
T
[The list of unverified conditions returned is identical to the list returned in the CSTR-1 matching.]

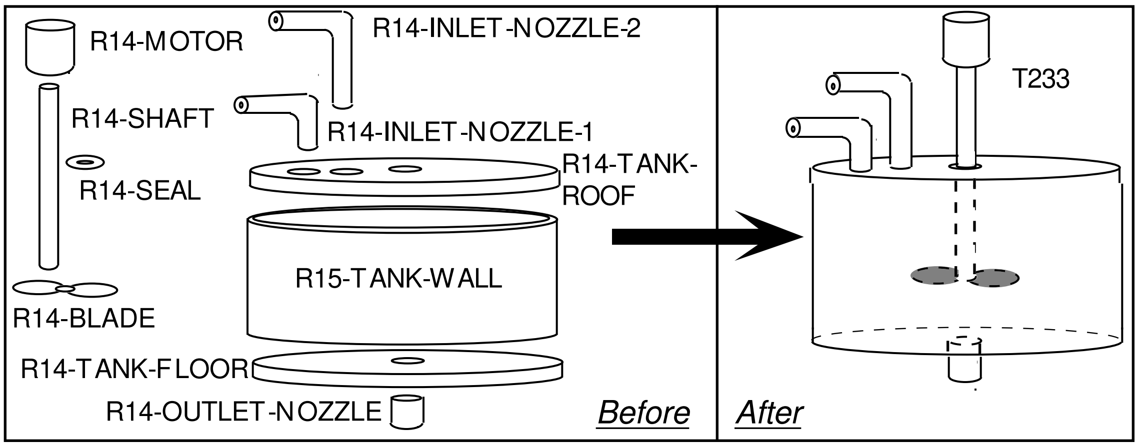
INPUTS: Vector #9
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (ninth vector-list))

OUTPUTS: Vector #9
T
[The list of unverified conditions returned is identical to the list returned in the CSTR-1 matching.]

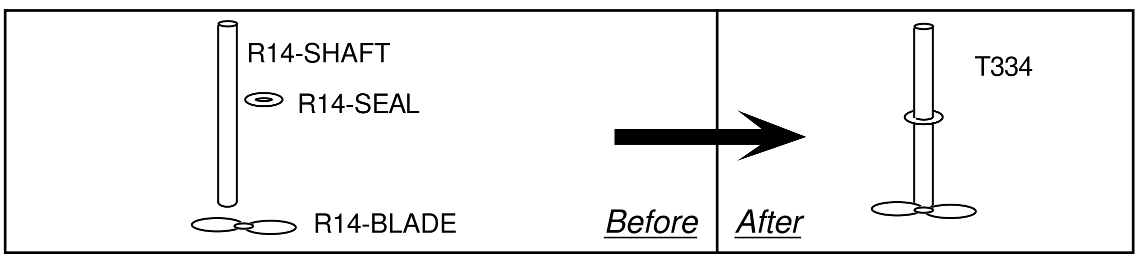
INPUTS: Vector #13
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 12 vector-list))

OUTPUTS: Vector #13
T
[The list of unverified conditions returned is identical to the list returned in the CSTR-1 matching.]

SIDE EFFECTS:



SIDE EFFECTS:--continued



OUTPUTS: --CONTINUED

Index from Assoc-list	Vector-list of matching Equipment Objects Returned	
	vector #9	vector #13
0	outlet-temperature-2771	outlet-temperature-2771
1	t367	t367
2	edge-onoz-1	edge-onoz-1
3	outlet-temperature-2766	outlet-temperature-2766
4	t346	t346
5	port-onoz-4*	port-onoz-4*
6	t358	t347*
7	t347	t358*
8	t337	t337
9	liquid-height-2766	liquid-height-2766
10	r14-outlet-nozzle	r14-outlet-nozzle
11	cold-inlet-flow-2771	hot-inlet-flow-2771*
12	cold-inlet-flow-2766	hot-inlet-flow-2766*
13	hot-inlet-flow-2771	cold-inlet-flow-2771*
14	hot-inlet-flow-2766	cold-inlet-flow-2766*
15	t338	t338
16	t233	t233
17	port-onoz-2	port-onoz-2
18	t335	t335
19	r14-tank-floor	r14-tank-floor
20	t334	t334
21	stream-tfloor-onoz	stream-tfloor-onoz
22	port-tfloor-4	port-tfloor-4
23	nil	nil
24	nil	nil
25	nil	nil
26	nil	nil
27	nil	nil
28	nil	nil
29	nil	nil
30	nil	nil
31	nil	nil
32	nil	nil
33	nil	nil
34	nil	nil
35	nil	nil
36	nil	nil
37	nil	nil
38	nil	nil
39	nil	nil

* different than vector #1.

NOTE: 16 vectors (correct matches) were returned, but only the 1st, 5th, 9th, and 13th are shown here because all the others do not meet the approximation conditions. See the note attached to the CSTR-1 Matching Case for a description of the 16 vectors.

D.11.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm8-port-liq-outlet 'm8-tempout-after 'port-liq-1 'outlet-temperature-2771)

OUTPUTS: [association-list and vector-list]

	Assoc-List of Model Objects Returned	notes	Vector-list of matching Equipment Objects Returned (See NOTE) vector #1	vector #5
0	m8-tempout-after	req-p	outlet-temperature-2771	outlet-temperature-2771
1	m8-port-liq-outlet	p-obj	t367	t367
2	m8-edge-outlet	p-obj	edge-onoz-1	edge-onoz-1
3	m8-tempout-before	req-p	outlet-temperature-2766	outlet-temperature-2766
4	m8-liquid	p-obj	t346	t346
5	m8-port-nozzle-liquid	p-obj	port-onoz-1	port-onoz-1
6	m8-port-liq-1-inlet	p-obj	t358	t347*
7	m8-port-liq-2-inlet	p-obj	t347	t358*
8	m8-port-liq-impeller	p-obj	t337	t337
9	m8-height-before	req-p	liquid-height-2766	liquid-height-2766
10	m8-outlet-nozzle	p-obj	r14-outlet-nozzle	r14-outlet-nozzle
11	m8-coldflow-after	req-p	cold-inlet-flow-2771	hot-inlet-flow-2771*
12	m8-coldflow-before	req-p	cold-inlet-flow-2766	hot-inlet-flow-2766*
13	m8-hotflow-after	req-p	hot-inlet-flow-2771	cold-inlet-flow-2771*
14	m8-hotflow-before	req-p	hot-inlet-flow-2766	cold-inlet-flow-2766*
15	m8-stream-liq-impeller	p-obj	t338	t338
16	m8-hot-cold-mixer	p-obj	t233	t233
17	m8-port-nozzle-floor	p-obj	port-onoz-2	port-onoz-2
18	m8-port-impeller-liquid	p-obj	t335	t335
19	m8-tank-floor	p-obj	r14-tank-floor	r14-tank-floor
20	m8-impeller	p-obj	t334	t334
21	m8-stream-floor-outlet	p-obj	stream-tfloor-onoz	stream-tfloor-onoz
22	m8-port-floor-outnoz	p-obj	port-tfloor-4	port-tfloor-4
23	m8-outflow-after	opt-p	nil	nil
24	m8-outflow-before	opt-p	nil	nil
25	m8-height-deriv-before	opt-p	nil	nil
26	m8-height-after	opt-p	nil	nil
27	m8-cflow-deriv-after	opt-p	nil	nil
28	m8-cflow-deriv-before	opt-p	nil	nil
29	m8-ctemp-deriv-before	opt-p	nil	nil
30	m8-ctemp-deriv-after	opt-p	nil	nil
31	m8-coldtemp-after	opt-p	nil	nil
32	m8-coldtemp-before	opt-p	nil	nil
33	m8-hflow-deriv-before	opt-p	nil	nil
34	m8-htemp-deriv-after	opt-p	nil	nil
35	m8-hflow-deriv-after	opt-p	nil	nil
36	m8-htemp-deriv-before	opt-p	nil	nil
37	m8-hottemp-after	opt-p	nil	nil
38	m8-hottemp-before	opt-p	nil	nil
39	m8-neg-port-liquid	neg-1	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.11 Dalle-Molle Model and CSTR-3 Equipment Description

D.11.1 Statistics on the Matching

Equipment Description:	CSTR-3
Model Description:	DALLE-MOLLE
Goal:	calculate VALUE
Max. partial matches:	2048
Complete Matches:	16
Rematches:	2
Run time:	2:04 (hr:min)
Objects created (kept):	? (192)
Links created (kept):	? (965)
Separately triggered reconfigurations:	2
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.11.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'outlet-temperature-2771 'value 'describes 'port-liq-1 'values)

OUTPUTS:
(
(outlet-temperature-2771 value describes port-liq-1 values m8-tempout-after m8-port-liq-outlet)
)

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
15 m8-hot-cold-mixer	p-obj		
16 m8-port-nozzle-floor	p-obj		
17 m8-port-impeller-liquid	p-obj		
18 m8-tank-floor	p-obj		
19 m8-impeller	p-obj		
20 m8-stream-floor-outlet	p-obj		
21 m8-port-floor-outnoz	p-obj		
22 m8-outflow-after	opt-p		
23 m8-outflow-before	opt-p		
24 m8-height-before	opt-p		
25 m8-height-deriv-before	opt-p		
26 m8-height-after	opt-p		
27 m8-cflow-deriv-after	opt-p		
28 m8-cflow-deriv-before	opt-p		
29 m8-ctemp-deriv-before	opt-p		
30 m8-ctemp-deriv-after	opt-p		
31 m8-coldtemp-after	opt-p		
32 m8-coldtemp-before	opt-p		
33 m8-hflow-deriv-before	opt-p		
34 m8-htemp-deriv-after	opt-p		
35 m8-hflow-deriv-after	opt-p		
36 m8-htemp-deriv-before	opt-p		
37 m8-hottemp-after	opt-p		
38 m8-hottemp-before	opt-p		
		intensive	(r15-liquid-d r15-liquid-c p183-liq) (r15-liquid-d p183-liq r15-liquid-b) (r15-liquid-d r15-liquid-b r15-liquid-a) (r15-liquid-d r15-liquid-c r15-liquid-a p181-liq) (r15-liquid-d r15-liquid-c r15-liquid-a p182-liq) (r15-liquid-d r15-liquid-b r15-liquid-a r15-liquid-c)
39 m8-neg-port-liquid	neg-1		

req-p -- required parameter
opt-p -- optional parameter
p-obj -- positive object
neg-n -- member of the nth negative set matched

```
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 1 vector-list))  <---2nd vector
T
 [The list of unverified  conditions  returned is the same as for the first vector.]

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 8 vector-list))  <---9th vector
T
 [The list of unverified  conditions  returned is the same as for the first vector.]

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 9 vector-list))  <---10th vector
T
 [The list of unverified  conditions  returned is the same as for the first vector.]
```

D.10.5 Executing the Model

This model does not get executed because the requirement that the M8-height-before model parameter (an input, required parameter) be matched was suspended, and thus no value for the height is available.

D.10.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfigur-ation type**	**Reconfigurations tried**
0	m8-tempout-after	opt-p		
1	m8-port-liq-outlet	p-obj		
2	m8-edge-outlet	p-obj		
3	m8-tempout-before	req-p		
4	m8-liquid	p-obj		
5	m8-port-nozzle-liquid	p-obj		
6	m8-port-liq-1-inlet	p-obj		
7	m8-port-liq-2-inlet	p-obj		
8	m8-port-liq-impeller	p-obj		
9	m8-outlet-nozzle	p-obj		
			intensive	(r15-liquid-d r15-liquid-c) (r15-liquid-d p183-liq) (r15-liquid-d r15-liquid-b) (r15-liquid-d r15-liquid-c r15-liquid-b) (r15-liquid-d r15-liquid-c r15-liquid-a)
10	m8-coldflow-after	req-p		
11	m8-coldflow-before	req-p		
12	m8-hotflow-after	req-p		
13	m8-hotflow-before	req-p		
14	m8-stream-liq-impeller	p-obj		
			part/whole	(r15-tank-floor r15-outlet-nozzle r15-tank-wall r15-tank-top r15-inlet-nozzle-1 r15-inlet-nozzle-2 r15-impeller r15-motor)

req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

OUTPUTS:-- continued

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-height-deriv-before time
m8-height-deriv-before time-type m8-height-deriv-before time-interval-type
m8-height-deriv-before time-precision m8-height-deriv-before time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-cflow-deriv-before time
m8-cflow-deriv-before time-type m8-cflow-deriv-before time-interval-type
m8-cflow-deriv-before time-precision m8-cflow-deriv-before time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-height-before time
m8-height-before time-type m8-height-before time-interval-type
m8-height-before time-precision m8-height-before time-precision-type)

(**possibly-before?**
m8-coldtemp-before value m8-coldtemp-before value-type
m8-coldtemp-before value-interval-type m8-coldtemp-before value-precision
m8-coldtemp-before value-precision-type m8-hottemp-before value
m8-hottemp-before value-type m8-hottemp-before value-interval-type
m8-hottemp-before value-precision m8-hottemp-before value-precision-type)

(**equal** m8-height-after time-units m8-coldflow-before time-units)

(**equal** m8-htemp-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-htemp-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-ctemp-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-ctemp-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-hflow-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-hflow-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-height-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-cflow-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-cflow-deriv-before time-units m8-coldflow-before time-units))

<u>OUTPUTS:--</u> continued

(**possibly-equal?**
m8-coldflow-after time m8-coldflow-after time-type
m8-coldflow-after time-interval-type m8-coldflow-after time-precision
m8-coldflow-after time-precision-type m8-ctemp-deriv-after time
m8-ctemp-deriv-after time-type m8-ctemp-deriv-after time-interval-type
m8-ctemp-deriv-after time-precision m8-ctemp-deriv-after time-precision-type)

(**possibly-equal?**
m8-coldflow-after time m8-coldflow-after time-type
m8-coldflow-after time-interval-type m8-coldflow-after time-precision
m8-coldflow-after time-precision-type m8-hflow-deriv-after time
m8-hflow-deriv-after time-type m8-hflow-deriv-after time-interval-type
m8-hflow-deriv-after time-precision m8-hflow-deriv-after time-precision-type)

(**possibly-equal?**
m8-coldflow-after time m8-coldflow-after time-type
m8-coldflow-after time-interval-type m8-coldflow-after time-precision
m8-coldflow-after time-precision-type m8-cflow-deriv-after time
m8-cflow-deriv-after time-type m8-cflow-deriv-after time-interval-type
m8-cflow-deriv-after time-precision m8-cflow-deriv-after time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-outflow-before time
m8-outflow-before time-type m8-outflow-before time-interval-type
m8-outflow-before time-precision m8-outflow-before time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-htemp-deriv-before time
m8-htemp-deriv-before time-type m8-htemp-deriv-before time-interval-type
m8-htemp-deriv-before time-precision m8-htemp-deriv-before time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-ctemp-deriv-before time
m8-ctemp-deriv-before time-type m8-ctemp-deriv-before time-interval-type
m8-ctemp-deriv-before time-precision m8-ctemp-deriv-before time-precision-type)

(**possibly-equal?**
m8-coldflow-before time m8-coldflow-before time-type
m8-coldflow-before time-interval-type m8-coldflow-before time-precision
m8-coldflow-before time-precision-type m8-hflow-deriv-before time
m8-hflow-deriv-before time-type m8-hflow-deriv-before time-interval-type
m8-hflow-deriv-before time-precision m8-hflow-deriv-before time-precision-type)

D.10.4 Checking Model Conditions

INPUTS:

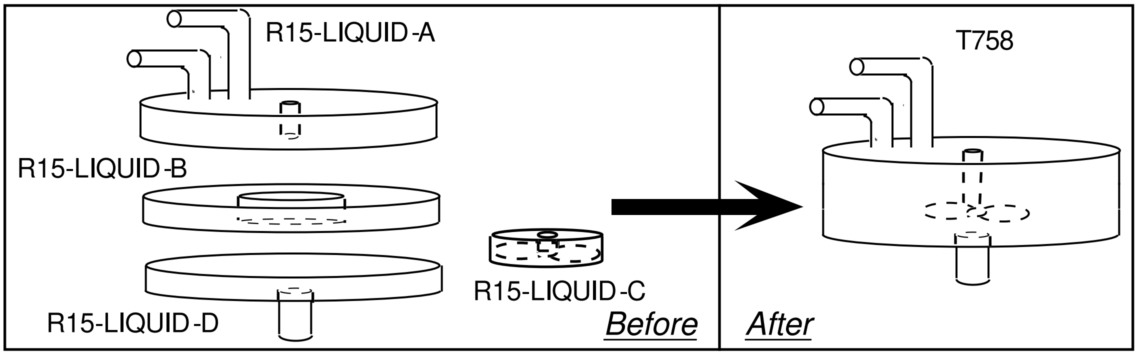
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 0 vector-list))

OUTPUTS:

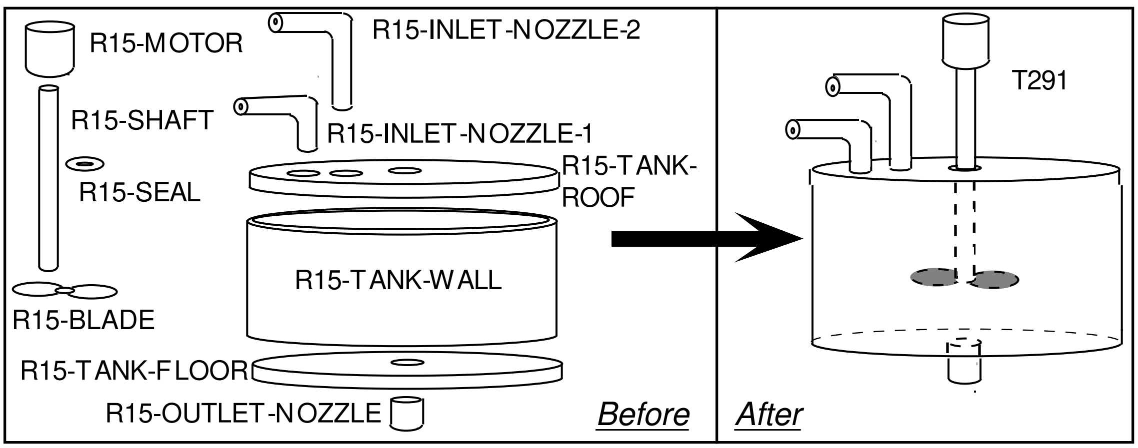
```
T
((m8-dalle-molle::flow-proportional-to-height?
   m8-outflow-before  value                  m8-outflow-after  value
   m8-outflow-after   value-precision-type   m8-height-before  value
   m8-height-after    value                  m8-height-after   value-precision-type)
 (equal m8-hottemp-before value m8-hottemp-after value)
 (equal m8-coldtemp-before value m8-coldtemp-after value)
 (equal m8-htemp-deriv-after value nil 0)
 (equal m8-htemp-deriv-before value nil 0)
 (equal m8-ctemp-deriv-after value nil 0)
 (equal m8-ctemp-deriv-before value nil 0)
 (equal m8-hflow-deriv-after value nil 0)
 (equal m8-hflow-deriv-before value nil 0)
 (equal m8-height-deriv-before value nil 0)
 (equal m8-cflow-deriv-after value nil 0)
 (equal m8-cflow-deriv-before value nil 0)
 (m8-dalle-molle::qspaces-have-common-element?
    m8-coldtemp-before   value-precision-type   nil  2
    m8-tempout-before    value-precision-type   nil  2)
 (m8-dalle-molle::qspaces-have-common-element?
    m8-hottemp-before   value-precision-type   nil  2
    m8-tempout-before   value-precision-type   nil  4)
 (m8-dalle-molle::qspace-ok? m8-height-before value-precision-type nil 3)
 (possibly-equal?
    m8-coldflow-after   time                   m8-coldflow-after   time-type
    m8-coldflow-after   time-interval-type     m8-coldflow-after   time-precision
    m8-coldflow-after   time-precision-type    m8-outflow-after    time
    m8-outflow-after    time-type              m8-outflow-after    time-interval-type
    m8-outflow-after    time-precision         m8-outflow-after    time-precision-type)
 (possibly-equal?
    m8-coldflow-after       time                   m8-coldflow-after       time-type
    m8-coldflow-after       time-interval-type     m8-coldflow-after       time-precision
    m8-coldflow-after       time-precision-type    m8-htemp-deriv-after    time
    m8-htemp-deriv-after    time-type              m8-htemp-deriv-after    time-interval-type
    m8-htemp-deriv-after    time-precision         m8-htemp-deriv-after    time-precision-type)
```

NOTE: 16 vectors (correct matches) were returned, but only the 1st, 2nd, 9th, and 10th are shown here because all the others do not meet the approximation conditions. See the note attached to the CSTR-1 Matching Case for a description of the 16 vectors.

SIDE EFFECTS:



SIDE EFFECTS:--continued



OUTPUTS: --CONTINUED

Index from Assoc-list	Vector-list of matching Equipment Objects Returned	
	vector #9	vector #10
0	outlet-temperature-2771	outlet-temperature-2771
1	t770	t770
2	t769	t769
3	outlet-temperature-2766	outlet-temperature-2766
4	t758	t758
5	port-onoz-1	port-onoz-4*
6	t775*	t775*
7	t771*	t771*
8	t759	t759
9	r15-outlet-nozzle	r15-outlet-nozzle
10	hot-inlet-flow-2771*	hot-inlet-flow-2771*
11	hot-inlet-flow-2766*	hot-inlet-flow-2766*
12	cold-inlet-flow-2771*	cold-inlet-flow-2771*
13	cold-inlet-flow-2766*	cold-inlet-flow-2766*
14	t762	t762
15	t291	t291
16	port-onoz-2	port-onoz-2
17	t761	t761
18	r15-tank-floor	r15-tank-floor
19	r15-impeller	r15-impeller
20	stream-tfloor-onoz	stream-tfloor-onoz
21	port-tfloor-4	port-tfloor-4
22	nil	nil
23	nil	nil
24	nil	nil
25	nil	nil
26	nil	nil
27	nil	nil
28	nil	nil
29	nil	nil
30	nil	nil
31	nil	nil
32	nil	nil
33	nil	nil
34	nil	nil
35	nil	nil
36	nil	nil
37	nil	nil
38	nil	nil
39	nil	nil

* different than vector #1.

NOTE: 16 vectors (correct matches) were returned, but only the 1st, 2nd, 9th, and 10th are shown here because all the others do not meet the approximation conditions. See the note attached to the CSTR-1 Matching Case for a description of the 16 vectors.

D.10.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm8-port-liq-outlet 'm8-tempout-after 'port-liq-d-6 'outlet-temperature-2771)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned (See NOTE)	
			vector #1	vector #2
0	m8-tempout-after	opt-p	outlet-temperature-2771	outlet-temperature-2771
1	m8-port-liq-outlet	p-obj	t770	t770
2	m8-edge-outlet	p-obj	t769	t769
3	m8-tempout-before	req-p	outlet-temperature-2766	outlet-temperature-2766
4	m8-liquid	p-obj	t758	t758
5	m8-port-nozzle-liquid	p-obj	port-onoz-1	port-onoz-4*
6	m8-port-liq-1-inlet	p-obj	t771	t771
7	m8-port-liq-2-inlet	p-obj	t775	t775
8	m8-port-liq-impeller	p-obj	t759	t759
9	m8-outlet-nozzle	p-obj	r15-outlet-nozzle	r15-outlet-nozzle
10	m8-coldflow-after	req-p	cold-inlet-flow-2771	cold-inlet-flow-2771
11	m8-coldflow-before	req-p	cold-inlet-flow-2766	cold-inlet-flow-2766
12	m8-hotflow-after	req-p	hot-inlet-flow-2771	hot-inlet-flow-2771
13	m8-hotflow-before	req-p	hot-inlet-flow-2766	hot-inlet-flow-2766
14	m8-stream-liq-impeller	p-obj	t762	t762
15	m8-hot-cold-mixer	p-obj	t291	t291
16	m8-port-nozzle-floor	p-obj	port-onoz-2	port-onoz-2
17	m8-port-impeller-liquid	p-obj	t761	t761
18	m8-tank-floor	p-obj	r15-tank-floor	r15-tank-floor
19	m8-impeller	p-obj	r15-impeller	r15-impeller
20	m8-stream-floor-outlet	p-obj	stream-tfloor-onoz	stream-tfloor-onoz
21	m8-port-floor-outnoz	p-obj	port-tfloor-4	port-tfloor-4
22	m8-outflow-after	opt-p	nil	nil
23	m8-outflow-before	opt-p	nil	nil
24	m8-height-before	opt-p	nil	nil
25	m8-height-deriv-before	opt-p	nil	nil
26	m8-height-after	opt-p	nil	nil
27	m8-cflow-deriv-after	opt-p	nil	nil
28	m8-cflow-deriv-before	opt-p	nil	nil
29	m8-ctemp-deriv-before	opt-p	nil	nil
30	m8-ctemp-deriv-after	opt-p	nil	nil
31	m8-coldtemp-after	opt-p	nil	nil
32	m8-coldtemp-before	opt-p	nil	nil
33	m8-hflow-deriv-before	opt-p	nil	nil
34	m8-htemp-deriv-after	opt-p	nil	nil
35	m8-hflow-deriv-after	opt-p	nil	nil
36	m8-htemp-deriv-before	opt-p	nil	nil
37	m8-hottemp-after	opt-p	nil	nil
38	m8-hottemp-before	opt-p	nil	nil
39	m8-neg-port-liquid	neg-1	nil	nil

continued →

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.9.6 Summary of Merges and Splits Tested

D.10 Dalle-Molle Model and CSTR-2 Equipment Description

D.10.1 Statistics on the Matching

Equipment Description:	CSTR-2
Model Description:	DALLE-MOLLE
Goal:	calculate VALUES
Max. partial matches:	1840
Complete Matches:	16
Rematches:	33
Run time:	4:46 (hr:min)
Objects created (kept):	? (77)
Links created (kept):	? (334)
Separately triggered reconfigurations:	3
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	Yes
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	Yes
Negative set triggered reconfiguration?	Yes

D.10.2 Generating the Potential Match Set

INPUTS:
```
(SELECT-MODELS 'outlet-temperature-2771 'value 'describes 'port-liq-d-6 'values)
```

OUTPUTS:
```
(
 (outlet-temperature-2771 value describes port-liq-d-6 values m8-tempout-after
      m8-port-liq-outlet)
 )
```

SIDE-EFFECTS: --continued

The parameter object, OUTLET-TEMPERATURE-2771, is created with attributes and links shown.

OUTLET-TEMPERATURE-2771 —exemplifies→ TEMPERATURE *(functional-type)*

OUTLET-TEMPERATURE-2771 —describes→ PORT-LIQUID-7

Attribute	Source
time: t1	*calculated*
time-dimension: 1	*specified*
time-units: min	*carried*
time-type: qlandmark	*specified*
time-interval-type: closed	*carried*
time-precision-type: (t0 t1 inf)	*calculated*
time-precision: nil	
time-max: nil	
time-min: nil	
value: te-5	*calculated*
value-dimension: 1	*specified*
value-units: nil	*carried*
value-type: qlandmark	*specified*
value-interval-type: point	*specified*
value-precision-type: (0 t-cold t-norm te-5 t-high inf)	*calculated*
value-precison: nil	
value-max: nil	
value-min: nil	

"location" attributes are all nil.

KEY

calculated-- attribute value calculated by model

specified-- attribute value specified in model param

carried-- attribute value copied from another parameter as specified in the "carried variables" list of the model description

* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

OUTPUTS:

*** Type <system>-q to use the QSIM User Interface ***
Making NQ-92: The New QSIM patchable
Experimental NQ-92 version 4. loaded
; Loading KSL-EXP-23: MURDOCK.MODELS; DALLE-MOLLE-INTERFACE.LISP#>
into package QSIM

Run time: 0.283 seconds to initialize a state.
Run time: 0.083 seconds to initialize a state.
Run time: 0.800 seconds to simulate 5 states.
Final states are: (S-11)

(OUTLET-TEMPERATURE-2771 T4) [This is the list returned by executing the model]

SIDE-EFFECTS: The parameter object, T4, is created with attributes and links shown.

T4* —exemplifies→ LENGTH *(functional-type)*

T4* —describes→ R12-LIQUID

T4*	
time: t1	*carried*
time-dimension: 1	*specified*
time-units: min	*carried*
time-type: qlandmark	*specified*
time-interval-type: closed	*carried*
time-precision-type: (t0 t1 inf)	*carried*
time-precision: nil	
time-max: nil	
time-min: nil	
value: h-7	*calculated*
value-dimension: 1	*specified*
value-units: nil	*carried*
value-type: qlandmark	*specified*
value-interval-type: point	*specified*
value-precision-type: (0 h-norm h-7 inf)	*calculated*
value-precison: nil	
value-max: nil	
value-min: nil	
"location" attributes are all nil.	

KEY
calculated-- attribute value calculated by model
specified-- attribute value specified in model param
carried-- attribute value copied from another parameter as specified in the "carried variables" list of the model description

* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

OUTPUTS:

```
Loading NQ-92: NQ-BIN; ALLOCATIONS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; TIME-PLOT.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NUMERIC-PLOT.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; EXAMINE.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; SPECIAL-DISPLAYS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; VERSIPLOT.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; TABLES.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; FRONT-END.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; SYNTAX.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; HQ-PLOT.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; VIEWERS-CLISP-2.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; VIEWERS-MISC.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; VIEWERS-CLISP.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; EQUATIONS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; RK45S.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; TIME-SCALE.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; HOD-DERIVATION.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NCURVATURE.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QMATCH.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; HOD-SYSTEM-PROPERTY.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; ARITHMETIC.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; Q2.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NIC.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; ENERGY-GF.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; COMPARISON.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; OCC-BRANCH-ELIMINATION.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; FOCUSING-TECHNIQUES.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NUMERIC-EQN-GEN.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NSIM-INTERVAL.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; NSIM-INIT.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; XEDGE.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; ENVISION-GUIDE-FILTER.XLD#> into package QSIM
Reading NQ-92: CATALOGS; EXAMPLES.LISP#> into package QSIM
Reading NQ-92: CATALOGS; TSA.LISP#> into package QSIM
Reading NQ-92: CATALOGS; EXTRAS.LISP#> into package QSIM
Reading NQ-92: CATALOGS; CHATTER4.LISP#> into package QSIM
Reading NQ-92: CATALOGS; CHATTER3.LISP#> into package QSIM
Reading NQ-92: CATALOGS; CHATTER2.LISP#> into package QSIM
Reading NQ-92: CATALOGS; CHATTER1.LISP#> into package QSIM
Reading NQ-92: CATALOGS; Q-FEATURES.LISP#> into package QSIM
Reading NQ-92: CC; CC-CATALOG.LISP#> into package QSIM
Loading NQ-92: NQ-BIN; DEFINITIONS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; FUNCTIONS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QSIM-EXPORTS.XLD#> into package QSIM
Reading NQ-92: CC; HYDRAULIC.LISP#> into package QSIM
Reading NQ-92: CC; ELECTRICAL.LISP#> into package QSIM
Reading NQ-92: NQ-92; TI-USER-INTERFACE.LISP#> into package QSIM
```

OUTPUTS:

Files to be Read:

NQ-92: CATALOGS; EXAMPLES.LISP#>
NQ-92: CATALOGS; TSA.LISP#>
NQ-92: CATALOGS; EXTRAS.LISP#>
NQ-92: CATALOGS; CHATTER4.LISP#>
NQ-92: CATALOGS; CHATTER3.LISP#>
NQ-92: CATALOGS; CHATTER2.LISP#>
NQ-92: CATALOGS; CHATTER1.LISP#>
NQ-92: CATALOGS; Q-FEATURES.LISP#>
NQ-92: CC; CC-CATALOG.LISP#>

Files to be loaded:

NQ-92: NQ-BIN; DEFINITIONS.XLD#>
NQ-92: NQ-BIN; FUNCTIONS.XLD#>
NQ-92: NQ-BIN; QSIM-EXPORTS.XLD#>

Files to be Read:

NQ-92: CC; HYDRAULIC.LISP#>
NQ-92: CC; ELECTRICAL.LISP#>
NQ-92: NQ-92; TI-USER-INTERFACE.LISP#>

Going to Make NQ-92: The New QSIM patchable

Load or Read all sixty-four of them and Make NQ-92: The New QSIM
patchable? (S, Y, or N) Yes.
Loading NQ-92: NQ-BIN; QDEFS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; PLOTTING-CONSTANTS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QUTILS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; DYNAMIC-SLOTS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; STRUCTURES.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; DEFOTHERS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; PROLOG.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; DEFINE-QDE.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; CFILTER.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; CONSTRAINTS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QSPACE-HIERARCHY.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; PROPAGATION.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; STATES.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; GLOBAL-FILTERS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; TRANSITIONS.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QSIM-TO-QGRAPH.XLD#> into package QSIM
Loading NQ-92: NQ-BIN; QPLOT.XLD#> into package QSIM

OUTPUTS:

NQ-92: NQ-BIN; DEFINE-QDE.XLD#>
NQ-92: NQ-BIN; CFILTER.XLD#>
NQ-92: NQ-BIN; CONSTRAINTS.XLD#>
NQ-92: NQ-BIN; QSPACE-HIERARCHY.XLD#>
NQ-92: NQ-BIN; PROPAGATION.XLD#>
NQ-92: NQ-BIN; STATES.XLD#>
NQ-92: NQ-BIN; GLOBAL-FILTERS.XLD#>
NQ-92: NQ-BIN; TRANSITIONS.XLD#>
NQ-92: NQ-BIN; QSIM-TO-QGRAPH.XLD#>
NQ-92: NQ-BIN; QPLOT.XLD#>
NQ-92: NQ-BIN; ALLOCATIONS.XLD#>
NQ-92: NQ-BIN; TIME-PLOT.XLD#>
NQ-92: NQ-BIN; NUMERIC-PLOT.XLD#>
NQ-92: NQ-BIN; EXAMINE.XLD#>
NQ-92: NQ-BIN; SPECIAL-DISPLAYS.XLD#>
NQ-92: NQ-BIN; VERSIPLOT.XLD#>
NQ-92: NQ-BIN; TABLES.XLD#>
NQ-92: NQ-BIN; FRONT-END.XLD#>
NQ-92: NQ-BIN; SYNTAX.XLD#>
NQ-92: NQ-BIN; HQ-PLOT.XLD#>
NQ-92: NQ-BIN; VIEWERS-CLISP-2.XLD#>
NQ-92: NQ-BIN; VIEWERS-MISC.XLD#>
NQ-92: NQ-BIN; VIEWERS-CLISP.XLD#>
NQ-92: NQ-BIN; EQUATIONS.XLD#>
NQ-92: NQ-BIN; RK45S.XLD#>
NQ-92: NQ-BIN; TIME-SCALE.XLD#>
NQ-92: NQ-BIN; HOD-DERIVATION.XLD#>
NQ-92: NQ-BIN; NCURVATURE.XLD#>
NQ-92: NQ-BIN; QMATCH.XLD#>
NQ-92: NQ-BIN; HOD-SYSTEM-PROPERTY.XLD#>
NQ-92: NQ-BIN; ARITHMETIC.XLD#>
NQ-92: NQ-BIN; Q2.XLD#>
NQ-92: NQ-BIN; NIC.XLD#>
NQ-92: NQ-BIN; ENERGY-GF.XLD#>
NQ-92: NQ-BIN; COMPARISON.XLD#>
NQ-92: NQ-BIN; OCC-BRANCH-ELIMINATION.XLD#>
NQ-92: NQ-BIN; FOCUSING-TECHNIQUES.XLD#>
NQ-92: NQ-BIN; NUMERIC-EQN-GEN.XLD#>
NQ-92: NQ-BIN; NSIM-INTERVAL.XLD#>
NQ-92: NQ-BIN; NSIM-INIT.XLD#>
NQ-92: NQ-BIN; XEDGE.XLD#>
NQ-92: NQ-BIN; ENVISION-GUIDE-FILTER.XLD#>

D.9.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
· [NOTE: All of the following is printed to the screen while the
QSIM system is being loaded.]

; Loading SYS: SITE; NQ-92.TRANSLATIONS#> into package FILE-SYSTEM
; Loading SYS: SITE; NQ-92.SYSTEM#> into package USER
; Loading SYS: SITE; NQ-92.TRANSLATIONS#> into package FILE-SYSTEM
Current login name is MURDOCK for host HPP.Stanford.EDU.
Type either password or loginname<space>password: XXXXXXXmurdock XXXXXXXX

File to Read

NQ-92: NQ-92; TI-NQ-SYSTEM.LISP#>

Read it? (Y or N) Yes.
; Reading NQ-92: NQ-92; TI-NQ-SYSTEM.LISP#>
; Loading SYS: SITE; XPOS-92.TRANSLATIONS#> into package FILE-SYSTEM
; Loading SYS: SITE; XPOS-92.SYSTEM#> into package USER
; Loading SYS: SITE; XPOS-92.TRANSLATIONS#> into package FILE-SYSTEM

; Reading Xpos-92: XPOS-92; XPOS-SYSTEM.LISP#>
Loading Xpos-92: XPOS-BIN; POSTSCRIPTER.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; QPLOT-PRIMITIVES.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; QPLOT-ADVANCED.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; PPLOT.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; TI-MACHINE-PARAMS.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; TI-POS-USRINTFACE.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; QGRAPH.XLD#> into package POS
Loading Xpos-92: XPOS-BIN; QGRAPH-IO.XLD#> into package POS
Making XPOS-92 patchable
Experimental XPOS version 1. loaded

If you use the :defaulted-batch option, compiler warnings for
system NQ-92 will be written to the file nq-2:nq-92;cwarns.lisp.

Files to be loaded:

NQ-92: NQ-BIN; QDEFS.XLD#>
NQ-92: NQ-BIN; PLOTTING-CONSTANTS.XLD#>
NQ-92: NQ-BIN; QUTILS.XLD#>
NQ-92: NQ-BIN; DYNAMIC-SLOTS.XLD#>
NQ-92: NQ-BIN; STRUCTURES.XLD#>
NQ-92: NQ-BIN; DEFOTHERS.XLD#>
NQ-92: NQ-BIN; PROLOG.XLD#>

OUTPUTS:-- continued

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-height-deriv-before	time
m8-height-deriv-before	time-type	m8-height-deriv-before	time-interval-type
m8-height-deriv-before	time-precision	m8-height-deriv-before	time-precision-type)

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-cflow-deriv-before	time
m8-cflow-deriv-before	time-type	m8-cflow-deriv-before	time-interval-type
m8-cflow-deriv-before	time-precision	m8-cflow-deriv-before	time-precision-type)

(**possibly-before?**

m8-coldtemp-before	value	m8-coldtemp-before	value-type
m8-coldtemp-before	value-interval-type	m8-coldtemp-before	value-precision
m8-coldtemp-before	value-precision-type	m8-hottemp-before	value
m8-hottemp-before	value-type	m8-hottemp-before	value-interval-type
m8-hottemp-before	value-precision	m8-hottemp-before	value-precision-type)

(**equal** m8-height-after time-units m8-coldflow-before time-units)

(**equal** m8-htemp-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-htemp-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-ctemp-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-ctemp-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-hflow-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-hflow-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-height-deriv-before time-units m8-coldflow-before time-units)

(**equal** m8-cflow-deriv-after time-units m8-coldflow-before time-units)

(**equal** m8-cflow-deriv-before time-units m8-coldflow-before time-units))

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 4 vector-list)) ⟵ 5th vector
T
[The list of unverified conditions returned is the same as for the first vector.]

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 8 vector-list)) ⟵ 9th vector
T
[The list of unverified conditions returned is the same as for the first vector.]

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 12 vector-list)) ⟵ 13th vector
T
[The list of unverified conditions returned is the same as for the first vector.]

OUTPUTS:-- continued

(**possibly-equal?**

m8-coldflow-after	time	m8-coldflow-after	time-type
m8-coldflow-after	time-interval-type	m8-coldflow-after	time-precision
m8-coldflow-after	time-precision-type	m8-ctemp-deriv-after	time
m8-ctemp-deriv-after	time-type	m8-ctemp-deriv-after	time-interval-type
m8-ctemp-deriv-after	time-precision	m8-ctemp-deriv-after	time-precision-type)

(**possibly-equal?**

m8-coldflow-after	time	m8-coldflow-after	time-type
m8-coldflow-after	time-interval-type	m8-coldflow-after	time-precision
m8-coldflow-after	time-precision-type	m8-hflow-deriv-after	time
m8-hflow-deriv-after	time-type	m8-hflow-deriv-after	time-interval-type
m8-hflow-deriv-after	time-precision	m8-hflow-deriv-after	time-precision-type)

(**possibly-equal?**

m8-coldflow-after	time	m8-coldflow-after	time-type
m8-coldflow-after	time-interval-type	m8-coldflow-after	time-precision
m8-coldflow-after	time-precision-type	m8-cflow-deriv-after	time
m8-cflow-deriv-after	time-type	m8-cflow-deriv-after	time-interval-type
m8-cflow-deriv-after	time-precision	m8-cflow-deriv-after	time-precision-type)

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-outflow-before	time
m8-outflow-before	time-type	m8-outflow-before	time-interval-type
m8-outflow-before	time-precision	m8-outflow-before	time-precision-type)

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-htemp-deriv-before	time
m8-htemp-deriv-before	time-type	m8-htemp-deriv-before	time-interval-type
m8-htemp-deriv-before	time-precision	m8-htemp-deriv-before	time-precision-type)

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-ctemp-deriv-before	time
m8-ctemp-deriv-before	time-type	m8-ctemp-deriv-before	time-interval-type
m8-ctemp-deriv-before	time-precision	m8-ctemp-deriv-before	time-precision-type)

(**possibly-equal?**

m8-coldflow-before	time	m8-coldflow-before	time-type
m8-coldflow-before	time-interval-type	m8-coldflow-before	time-precision
m8-coldflow-before	time-precision-type	m8-hflow-deriv-before	time
m8-hflow-deriv-before	time-type	m8-hflow-deriv-before	time-interval-type
m8-hflow-deriv-before	time-precision	m8-hflow-deriv-before	time-precision-type)

D.9.4 Checking Model Conditions

INPUTS:

```
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (nth 0 vector-list))
```

OUTPUTS:

```
T
((m8-dalle-molle::flow-proportional-to-height?
   m8-outflow-before   value                  m8-outflow-after    value
   m8-outflow-after    value-precision-type   m8-height-before    value
   m8-height-after     value                  m8-height-after     value-precision-type)
 (equal m8-hottemp-before value m8-hottemp-after value)
 (equal m8-coldtemp-before value m8-coldtemp-after value)
 (equal m8-htemp-deriv-after value nil 0)
 (equal m8-htemp-deriv-before value nil 0)
 (equal m8-ctemp-deriv-after value nil 0)
 (equal m8-ctemp-deriv-before value nil 0)
 (equal m8-hflow-deriv-after value nil 0)
 (equal m8-hflow-deriv-before value nil 0)
 (equal m8-height-deriv-before value nil 0)
 (equal m8-cflow-deriv-after value nil 0)
 (equal m8-cflow-deriv-before value nil 0)
 (m8-dalle-molle::qspaces-have-common-element?
    m8-coldtemp-before   value-precision-type   nil   2
    m8-tempout-before    value-precision-type   nil   2)
 (m8-dalle-molle::qspaces-have-common-element?
    m8-hottemp-before    value-precision-type   nil   2
    m8-tempout-before    value-precision-type   nil   4)
 (possibly-equal?
    m8-coldflow-after   time                   m8-coldflow-after   time-type
    m8-coldflow-after   time-interval-type     m8-coldflow-after   time-precision
    m8-coldflow-after   time-precision-type    m8-outflow-after    time
    m8-outflow-after    time-type              m8-outflow-after    time-interval-type
    m8-outflow-after    time-precision         m8-outflow-after    time-precision-type)
 (possibly-equal?
    m8-coldflow-after       time                   m8-coldflow-after       time-type
    m8-coldflow-after       time-interval-type     m8-coldflow-after       time-precision
    m8-coldflow-after       time-precision-type    m8-htemp-deriv-after    time
    m8-htemp-deriv-after    time-type              m8-htemp-deriv-after    time-interval-type
    m8-htemp-deriv-after    time-precision         m8-htemp-deriv-after    time-precision-type)
```

NOTE: 16 vectors (correct matches) were returned, but only the 1st, 5th, 9th, and 13th are shown here because all the others do not meet the approximation conditions. (The function CHECK-MODEL-CONDITIONS returns NIL when called on those other vectors.) These 16 different vectors result from multiple ways in which some objects and parameters can be matched. At the matching stage, the relative times of parameters are not considered, so M8-COLDFLOW-BEFORE (a model parameter) can be matched to either the initial cold flow rate or the later cold flow rate (2 possibilities). Similarly M8-HOTFLOW-BEFORE can be matched to either of the two available flow rates (2 possibiliites). The matching stage also does not distinguish between relative values of parameters, so it can match M8-PORT-LIQ-1-INLET, the inlet for the cold flow, to either the cold inlet or the hot inlet (2 possibilities). This model description also requires that the outlet port of the liquid share an edge with the outlet nozzle, and the outlet nozzle actually has two ports that have such an edge (2 possibilities). There are 16 (2 X 2 X 2 X 2) combinations.

OUTPUTS: --CONTINUED

Index from Assoc-list	Vector-list of matching Equipment Objects Returned	
	vector #9	vector #13
0	outlet-temperature-2771	outlet-temperature-2771
1	port-liquid-7	port-liquid-7
2	edge-liquid-9	edge-liquid-9
3	outlet-temperature-2766	outlet-temperature-2766
4	r12-liquid	r12-liquid
5	port-onoz-1*	port-onoz-1*
6	port-liquid-3	port-liquid-1*
7	port-liquid-1	port-liquid-3*
8	port-liquid-8	port-liquid-8
9	liquid-height-2766	liquid-height-2766
10	r12-outlet-nozzle	r12-outlet-nozzle
11	hot-inlet-flow-2771	cold-inlet-flow-2771*
12	hot-inlet-flow-2766	cold-inlet-flow-2766*
13	cold-inlet-flow-2771	hot-inlet-flow-2771*
14	cold-inlet-flow-2766	hot-inlet-flow-2766*
15	stream-imp-liq	stream-imp-liq
16	r12-cstr	r12-cstr
17	port-onoz-2	port-onoz-2
18	port-imp-1	port-imp-1
19	r12-tank-floor	r12-tank-floor
20	r12-impeller	r12-impeller
21	stream-tfloor-onoz	stream-tfloor-onoz
22	port-tfloor-4	port-tfloor-4
23	nil	nil
24	nil	nil
25	nil	nil
26	nil	nil
27	nil	nil
28	nil	nil
29	nil	nil
30	nil	nil
31	nil	nil
32	nil	nil
33	nil	nil
34	nil	nil
35	nil	nil
36	nil	nil
37	nil	nil
38	nil	nil
39	nil	nil

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.9.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH
'm8-port-liq-outlet 'm8-tempout-after 'port-liquid-7 'outlet-temperature-2771)
OUTPUTS: (association-list and vector-list)

	Assoc-List of Model Objects Returned	notes	Vector-list of matching Equipment Objects Returned (See NOTE)	
			vector #1	vector #5
0	m8-tempout-after	req-p	outlet-temperature-2771	outlet-temperature-2771
1	m8-port-liq-outlet	p-obj	port-liquid-7	port-liquid-7
2	m8-edge-outlet	p-obj	edge-liquid-9	edge-liquid-9
3	m8-tempout-before	req-p	outlet-temperature-2766	outlet-temperature-2766
4	m8-liquid	p-obj	r12-liquid	r12-liquid
5	m8-port-nozzle-liquid	p-obj	port-onoz-4	port-onoz-4
6	m8-port-liq-1-inlet	p-obj	port-liquid-3	port-liquid-1*
7	m8-port-liq-2-inlet	p-obj	port-liquid-1	port-liquid-3*
8	m8-port-liq-impeller	p-obj	port-liquid-8	port-liquid-8
9	m8-height-before	req-p	liquid-height-2766	liquid-height-2766
10	m8-outlet-nozzle	p-obj	r12-outlet-nozzle	r12-outlet-nozzle
11	m8-coldflow-after	req-p	hot-inlet-flow-2771	cold-inlet-flow-2771*
12	m8-coldflow-before	req-p	hot-inlet-flow-2766	cold-inlet-flow-2766*
13	m8-hotflow-after	req-p	cold-inlet-flow-2771	hot-inlet-flow-2771*
14	m8-hotflow-before	req-p	cold-inlet-flow-2766	hot-inlet-flow-2766*
15	m8-stream-liq-impeller	p-obj	stream-imp-liq	stream-imp-liq
16	m8-hot-cold-mixer	p-obj	r12-cstr	r12-cstr
17	m8-port-nozzle-floor	p-obj	port-onoz-2	port-onoz-2
18	m8-port-impeller-liquid	p-obj	port-imp-1	port-imp-1
19	m8-tank-floor	p-obj	r12-tank-floor	r12-tank-floor
20	m8-impeller	p-obj	r12-impeller	r12-impeller
21	m8-stream-floor-outlet	p-obj	stream-tfloor-onoz	stream-tfloor-onoz
22	m8-port-floor-outnoz	p-obj	port-tfloor-4	port-tfloor-4
23	m8-outflow-after	opt-p	nil	nil
24	m8-outflow-before	opt-p	nil	nil
25	m8-height-deriv-before	opt-p	nil	nil
26	m8-height-after	opt-p	nil	nil
27	m8-cflow-deriv-after	opt-p	nil	nil
28	m8-cflow-deriv-before	opt-p	nil	nil
29	m8-ctemp-deriv-before	opt-p	nil	nil
30	m8-ctemp-deriv-after	opt-p	nil	nil
31	m8-coldtemp-after	opt-p	nil	nil
32	m8-coldtemp-before	opt-p	nil	nil
33	m8-hflow-deriv-before	opt-p	nil	nil
34	m8-htemp-deriv-after	opt-p	nil	nil
35	m8-hflow-deriv-after	opt-p	nil	nil
36	m8-htemp-deriv-before	opt-p	nil	nil
37	m8-hottemp-after	opt-p	nil	nil
38	m8-hottemp-before	opt-p	nil	nil
39	m8-neg-port-liquid	neg-1	nil	nil

D.9 Dalle-Molle Model and CSTR-1 Equipment Description

D.9.1 Statistics on the Matching

Equipment Description:	CSTR-1
Model Description:	DALLE-MOLLE
Goal:	calculate VALUES
Max. partial matches:	240
Complete Matches:	16
Rematches:	0
Run time:	0:13 (hr:min)
Objects created (kept):	1 (1)
Links created (kept):	2 (2)
Separately triggered reconfigurations:	0
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.9.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS 'outlet-temperature-2771 'value 'describes 'port-liquid-7 'values)
```

OUTPUTS:

```
(
 (outlet-temperature-2771  value  describes  port-liquid-7  values m8-tempout-after
                           m8-port-liq-outlet)
 )
```

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
13 m2-stream-2	p-obj	intensive--Note #1	(region-b region-a) (b-4) --Note #2 (region-a b-1 b-2 b-3) (b-3) (region-a b-1 b-2) (region-a b-1) (b-2) (b-2 b-3) (b-3 b-4) (b-2 b-3 b-4) (region-b region-c) (b-1 b-2 b-3) (b-4 region-c) (b-1 b-2) (b-3 b-4 region-c) (b-2 b-3 b-4 region-c) (b-1)
14 m2-port-3b	p-obj		
		intensive	*termination* --Note #3
15 m2-pipe	p-obj		
req-p -- required parameter p-obj -- positive object	opt-p -- optional parameter neg-n -- member of the nth negative set matched		

Note #1: This intensive reconfiguration is not triggered by inability to find an equipment object to match M2-PORT-3B, but by the attempt to find a "has-edge" link from objects matching M2-PORT-2A (4) to objects matching M2-EDGE-2A (10). At this point the matcher was finding all objects linked to M2-PORT-2A. It successfully found and matched M2-STREAM-2, but could not correctly match the "has-edge" link to the objects matching M2-EDGE-2A, which was already found as edge of the port M2-PORT-3A. The cylindrical surface of the liquid region must share edges with the ends of the region, and this condition is not met in the current liquid region, Region-C.

Note #2: This case does both merges and splits. The split regions are referred to as "B-1", "B-2", "B-3", and "B-4" where B is the letter naming the original region. Region-B can be split along the boundary of Elbow-1, Pipe-2, Pipe-3, and Pipe-4, where the pieces are numbered from left to right, with B-1 corresponding to the liquid region inside Elbow-1.

Note #3: No further merges or splits were available, because all regions had previously been tested and failed to rematch the objects matched so far. Thus the intensive reconfiguration heuristic prevents any further merges or splits from being tested.

D.8.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

No conditions checked because no matches found.

D.8.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
No model executed because no matches found.

SIDE-EFFECTS: No side effects..

D.8.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m2-thermal-conductivity	req-p		
1	m2-fluid-in-pipe	p-obj		
2	m2-port-3a	p-obj		
3	m2-port-1b	p-obj		
4	m2-port-2a	p-obj		
5	m2-heat-capacity	req-p		
6	m2-density	req-p		
7	m2-viscosity	req-p		
8	m2-stream-3	p-obj		
9	m2-edge-1b	p-obj		
10	m2-edge-2a	p-obj		
11	m2-stream-1	p-obj		
12	m2-mass-flow-rate	req-p		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

D.8 Dittus-Boelter Model and PIPES-6 Equipment Description

D.8.1 Statistics on the Matching

Equipment Description:	PIPES-6
Model Description:	DITTUS-BOELTER
Goal:	find EFFECTS
Max. partial matches:	40
Complete Matches:	0
Rematches:	20
Run time:	0:15 (hr:min)
Objects created (kept):	183 (0)
Links created (kept):	1230 (0)
Separately triggered reconfigurations:	2
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.8.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS 'thermal-cond-b 'value 'describes 'region-b 'effects)
```

OUTPUTS:

```
(
 (thermal-cond-b value describes region-b effects m2-thermal-conductivity m2-fluid-in-pipe)
)
```

D.8.3 Match/Reconfigure

INPUTS:

```
(DO-SIMPLE-MATCH 'm2-fluid-in-pipe 'm2-thermal-conductivity 'region-B 'thermal-cond-b)
```

OUTPUTS: (association-list and vector-list)

```
nil
```

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
23 m2-pipe-length	opt-p		
24 m2-pipe-axial-direction	opt-p		
25 m2-neg-port	neg-1		
26 m2-neg-edge-3a	neg-2		
27 m2-neg-edge-1b	neg-3		
28 m2-neg-s-normal-1b	neg-4		
29 m2-neg-edge-2a	neg-5		
30 m2-neg-s-normal-2a	neg-6		
31 m2-neg-endpt-1b	neg-7		
32 m2-neg-endpt-2a	neg-8		
req-p -- required parameter p-obj -- positive object	opt-p -- optional parameter neg-n -- member of the nth negative set matched		

Note #1: This intensive reconfiguration is not triggered by inability to find an equipment object to match M2-PORT-3B, but by the attempt to find a "has-edge" link from objects matching M2-PORT-2A (4) to objects matching M2-EDGE-2A (10). At this point the matcher was finding all objects linked to M2-PORT-2A. It successfully found and matched M2-STREAM-2, but could not correctly match the "has-edge" link to the objects matching M2-EDGE-2A, which was already found as edge of the port M2-PORT-3A. The cylindrical surface of the liquid region must share edges with the ends of the region, and this condition is not met in the current liquid region, Region-C.

Note #2: This case does both merges and splits. The split regions are referred to as "x-left", x-middle", "x-right", where x is the letter naming the original region. Region-B can be split along the boundary of Pipe-1 and Pipe-2, with the two halves called "B-left" (portion in Pipe-1) and "B-right" for the portion in Pipe-2. Region-C can be split into 3 portions corresponding to the liquid in Pipe-2 ("C-left"), Pipe-3 ("C-middle"), and Pipe-4 ("C-right").

D.7.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
0 m2-heat-capacity	req-p		
1 m2-fluid-in-pipe	p-obj		
2 m2-port-3a	p-obj		
3 m2-port-1b	p-obj		
4 m2-port-2a	p-obj		
5 m2-density	req-p		
6 m2-thermal-conductivity	req-p		
7 m2-viscosity	req-p		
8 m2-stream-3	p-obj		
9 m2-edge-1b	p-obj		
10 m2-edge-2a	p-obj		
11 m2-stream-1	p-obj		
12 m2-mass-flow-rate	req-p		
13 m2-stream-2	p-obj		
		intensive--Note #1	(region-b region-c) (c-right)--Note #2 (region-b c-left c-middle) (c-middle) (region-b c-left) (b-left) (b-right c-left) (b-right c-left c-middle) (c-middle c-right) (b-right region-c) (region-c region-d) (c-left c-middle) (c-left)
14 m2-port-3b	p-obj		
15 m2-pipe	p-obj		
16 m2-pipe-inside-diameter	req-p		
17 m2-inlet-temperature	opt-p		
18 m2-surface-normal-1b	opt-p		
19 m2-outlet-temperature	opt-p		
20 m2-surface-normal-2a	opt-p		
21 m2-pipe-wall-temperature	opt-p		
22 m2-heat-transfer-coefficient	opt-p		

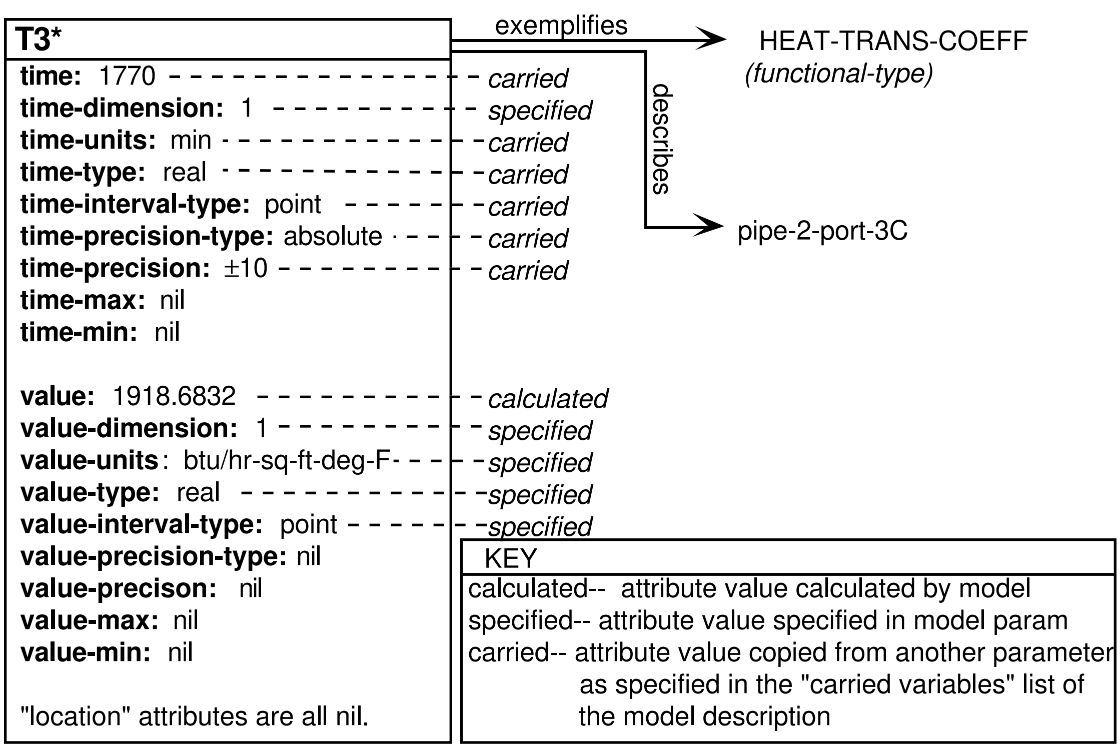
req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

D.7.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
(T3)

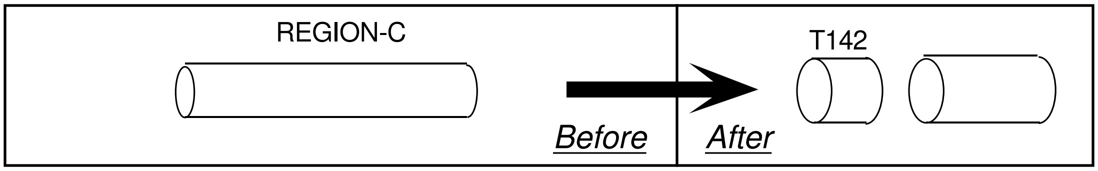
SIDE-EFFECTS: The parameter object, T3, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

```
(m2-dittus-boelter:l-over-d-ok?
   m2-pipe-length       value       m2-pipe-inside-diameter    value)

(any-common-portion?
   m2-viscosity            time                   m2-viscosity            time-type
   m2-viscosity            time-interval-type     m2-viscosity            time-precision
   m2-viscosity            time-precision-type    m2-outlet-temperature   time
   m2-outlet-temperature   time-type              m2-outlet-temperature   time-interval-type
   m2-outlet-temperature   time-precision         m2-outlet-temperature   time-precision-type)

(any-common-portion?
 m2-viscosity               time                  m2-viscosity               time-type
 m2-viscosity               time-interval-type    m2-viscosity               time-precision
 m2-viscosity               time-precision-type   m2-pipe-wall-temperature   time
 m2-pipe-wall-temperature   time-type             m2-pipe-wall-temperature   time-interval-type
 m2-pipe-wall-temperature   time-precision        m2-pipe-wall-temperature   time-precision-type)

 (equal   m2-viscosity  time-type    m2-pipe-wall-temperature time-type)

 (equal   m2-viscosity  time-type    m2-outlet-temperature  time-type)

 (equal   m2-viscosity  time-dimension    m2-pipe-wall-temperature  time-dimension)

 (equal   m2-viscosity  time-dimension    m2-outlet-temperature  time-dimension)

 (equal   m2-viscosity  time-units    m2-pipe-wall-temperature  time-units)

 (equal   m2-viscosity  time-units    m2-outlet-temperature  time-units))
```

SIDE EFFECTS:



D.7.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

((**vectors-parallel?**

m2-pipe-axial-direction	value	m2-pipe-axial-direction	value-type
m2-pipe-axial-direction	value-interval-type	m2-pipe-axial-direction	value-dimension
m2-pipe-axial-direction	value-precision	m2-pipe-axial-direction	value-precision-type
m2-pipe-axial-direction	time	m2-pipe-axial-direction	time-type
m2-pipe-axial-direction	time-interval-type	m2-pipe-axial-direction	time-dimension
m2-pipe-axial-direction	time-precision	m2-pipe-axial-direction	time-precision-type
m2-surface-normal-1b	value	m2-surface-normal-1b	value-type
m2-surface-normal-1b	value-interval-type	m2-surface-normal-1b	value-dimension
m2-surface-normal-1b	value-precision	m2-surface-normal-1b	value-precision-type
m2-surface-normal-1b	time	m2-surface-normal-1b	time-type
m2-surface-normal-1b	time-interval-type	m2-surface-normal-1b	time-dimension
m2-surface-normal-1b	time-precision	m2-surface-normal-1b	time-precision-type)

(**vectors-parallel?**

m2-pipe-axial-direction	value	m2-pipe-axial-direction	value-type
m2-pipe-axial-direction	value-interval-type	m2-pipe-axial-direction	value-dimension
m2-pipe-axial-direction	value-precision	m2-pipe-axial-direction	value-precision-type
m2-pipe-axial-direction	time	m2-pipe-axial-direction	time-type
m2-pipe-axial-direction	time-interval-type	m2-pipe-axial-direction	time-dimension
m2-pipe-axial-direction	time-precision	m2-pipe-axial-direction	time-precision-type
m2-surface-normal-2a	value	m2-surface-normal-2a	value-type
m2-surface-normal-2a	value-interval-type	m2-surface-normal-2a	value-dimension
m2-surface-normal-2a	value-precision	m2-surface-normal-2a	value-precision-type
m2-surface-normal-2a	time	m2-surface-normal-2a	time-type
m2-surface-normal-2a	time-interval-type	m2-surface-normal-2a	time-dimension
m2-surface-normal-2a	time-precision	m2-surface-normal-2a	time-precision-type)

(**m2-dittus-boelter:pipe-temperatures-ok?**

m2-inlet-temperature	value
m2-outlet-temperature	value
m2-pipe-wall-temperature	value)

D.7.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-fluid-in-pipe 'm2-heat-capacity 'region-c 'heat-capacity-c)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m2-heat-capacity	req-p	heat-capacity-c	
1	m2-fluid-in-pipe	p-obj	t142	
2	m2-port-3 a	p-obj	t152	
3	m2-port-1 b	p-obj	t150	
4	m2-port-2 a	p-obj	t138	
5	m2-density	req-p	rho-c	
6	m2-thermal-conductivity	req-p	thermal-cond-c	
7	m2-viscosity	req-p	mu-c	
8	m2-stream-3	p-obj	stream-pipe-2-c3	
9	m2-edge-1 b	p-obj	t151	
10	m2-edge-2 a	p-obj	t140	
11	m2-stream-1	p-obj	stream-bc	
12	m2-mass-flow-rate	req-p	inlet-flow-c	
13	m2-stream-2	p-obj	t139	
14	m2-port-3 b	p-obj	pipe-2-port-3 c	
15	m2-pipe	p-obj	pipe-2	
16	m2-pipe-inside-diameter	req-p	pipe-2-id	
17	m2-inlet-temperature	opt-p	fluid-temp-port-c1	
18	m2-surface-normal-1 b	opt-p	nil	
19	m2-outlet-temperature	opt-p	nil	
20	m2-surface-normal-2 a	opt-p	nil	
21	m2-pipe-wall-temperature	opt-p	nil	
22	m2-heat-transfer-coefficient	opt-p	wall-h-trans-coeff-c	
23	m2-pipe-length	opt-p	nil	
24	m2-pipe-axial-direction	opt-p	pipe-2-axial-direction	
25	m2-neg-port	neg-1	nil	
26	m2-neg-edge-3 a	neg-2	nil	
27	m2-neg-edge-1 b	neg-3	nil	
28	m2-neg-s-normal-1 b	neg-4	nil	
29	m2-neg-edge-2 a	neg-5	nil	
30	m2-neg-s-normal-2 a	neg-6	nil	
31	m2-neg-endpt-1 b	neg-7	nil	
32	m2-neg-endpt-2 a	neg-8	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.7 Dittus-Boelter Model and PIPES-5 Equipment Description

D.7.1 Statistics on the Matching

Equipment Description:	PIPES-5
Model Description:	DITTUS-BOELTER
Goal:	find EFFECTS
Max. partial matches:	24
Complete Matches:	1
Rematches:	18
Run time:	0:14 (hr:min)
Objects created (kept):	147 (8)
Links created (kept):	1268 (29)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	Yes
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.7.2 Generating the Potential Match Set

INPUTS:

(SELECT-MODELS 'heat-capacity-c 'value 'describes 'region-c 'effects)

OUTPUTS:

```
(
 (heat-capacity-c value describes region-c effects m2-heat-capacity m2-fluid-in-pipe)
)
```

D.6.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

No correct matches, so no conditions checked.

D.6.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
No correct matches, so no execution of model.

SIDE-EFFECTS: No sideeffects.

D.6.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		notes	Reconfiguration type	Reconfigurations tried
0	m2-heat-capacity	req-p		
1	m2-fluid-in-pipe	p-obj		
2	m2-port-3 a	p-obj		
3	m2-port-2 a	p-obj		
4	m2-port-1 b	p-obj		
5	m2-density	req-p		
6	m2-thermal-conductivity	req-p		
7	m2-viscosity	req-p		
8	m2-stream-3	p-obj		
9	m2-edge-1 b	p-obj		
10	m2-edge-2 a	p-obj		
11	m2-stream-2	p-obj		
12	m2-stream-1	p-obj	intensive	(region-b region-a) (region-b region-c) (region-b region-c region-d) *termination*
13	m2-mass-flow-rate	req-p		

req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

D.6 Dittus-Boelter Model and PIPES-4 Equipment Description

D.6.1 Statistics on the Matching

Equipment Description:	PIPES-4
Model Description:	DITTUS-BOELTER
Goal:	find EFFECTS
Max. partial matches:	8
Complete Matches:	0
Rematches:	3
Run time:	0:01 (hr:min)
Objects created (kept):	24 (0)
Links created (kept):	110 (0)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	Yes
Negative set triggered reconfiguration?	No

D.6.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'heat-capacity-b 'value 'describes 'region-b 'effects)

OUTPUTS:
(
(heat-capacity-b value describes region-b effects m2-heat-capacity m2-fluid-in-pipe)
)

D.6.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-fluid-in-pipe 'm2-heat-capacity 'region-b 'heat-capacitiy-b)

OUTPUTS: (association-list and vector-list)

nil

D.5.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m2-heat-transfer-coefficient	opt-p		
1	m2-port-3b	p-obj		
2	m2-stream-3	p-obj		
3	m2-pipe	p-obj		
4	m2-edge-2a	p-obj		
5	m2-edge-1b	p-obj		
6	m2-port-3a	p-obj		
7	m2-pipe-inside-diameter	req-p		
8	m2-port-2a	p-obj		
9	m2-port-1b	p-obj		
10	m2-fluid-in-pipe	p-obj		
11	m2-stream-2	p-obj		
12	m2-stream-1	p-obj		
13	m2-mass-flow-rate	req-p		
14	m2-heat-capacity	req-p		
15	m2-density	req-p		
16	m2-thermal-conductivity	req-p		
17	m2-viscosity	req-p		
18	m2-pipe-wall-temperature	opt-p		
19	m2-pipe-length	opt-p		
20	m2-pipe-axial-direction	opt-p		
21	m2-outlet-temperature	opt-p		
22	m2-surface-normal-2a	opt-p		
23	m2-inlet-temperature	opt-p		
24	m2-surface-normal-1b	opt-p		
			intensive	(region-wc region-wd) (region-wc region-wb) (region-wc region-wd region-wb) (region-wc region-wd region-we) *termination*
25	m2-neg-endpt-2a	neg-1		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

D.5.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-port-3b 'm2-heat-transfer-coefficient 'pipe-3-port-3wc
'wall-h-trans-coeff-c)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned	notes	Vector-list of matching Equipment Objects Returned	
		vector #1	vector #2
		nil	
* different than vector #1. req-p -- required parameter	p-obj -- positive object opt-p -- optional parameter	neg-n -- member of the nth negative set matched	

D.5.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

No conditions checked because no models matched.

D.5.5 Executing the Model

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

No conditions checked because no models matched.

D.5 Dittus-Boelter Model and PIPES-3 Equipment Description

D.5.1 Statistics on the Matching

Equipment Description:	PIPES-3
Model Description:	DITTUS-BOELTER
Goal:	calculate VALUES
Max. partial matches:	156
Complete Matches:	0
Rematches:	4
Run time:	0:11 (hr:min)
Objects created (kept):	104 (0)
Links created (kept):	557 (0)
Separately triggered reconfigurations:	1
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	Yes

D.5.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS ''wall-h-trans-coeff-c 'value 'describes  'pipe-3-port-3wc 'values)
```

OUTPUTS:

```
(
 (wall-h-trans-coeff-c value describes pipe-3-port-3wc values m2-heat-transfer-coefficient
          m2-port-3b)
)
```

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
29 m2-neg-s-normal-2a	neg-5	intensive	(region-a region-b) (region-a region-c region-b) (region-a region-c region-d) (region-a region-c region-e) (region-a region-d region-e) (region-a region-d region-b) (region-a region-e region-b)
30 m2-neg-edge-1b	neg-6		
31 m2-neg-s-normal-1b	neg-7	intensive	(region-a region-c region-b region-d) (region-a region-c region-b region-e) (region-a region-c region-d region-e) (region-a region-d region-e region-b) (region-a region-c region-b region-d region-e)
32 m2-neg-port	neg-8		

req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

D.4.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m2-heat-transfer-coefficient	opt-p		
1	m2-port-3 b	p-obj		
2	m2-stream-3	p-obj		
3	m2-pipe	p-obj		
4	m2-edge-2 a	p-obj		
5	m2-edge-1 b	p-obj		
6	m2-port-3 a	p-obj		
7	m2-pipe-inside-diameter	req-p		
8	m2-port-2 a	p-obj		
9	m2-port-1 b	p-obj		
10	m2-fluid-in-pipe	p-obj		
11	m2-stream-2	p-obj		
12	m2-stream-1	p-obj		
13	m2-mass-flow-rate	req-p		
14	m2-heat-capacity	req-p		
15	m2-density	req-p		
16	m2-thermal-conductivity	req-p		
17	m2-viscosity	req-p		
18	m2-pipe-wall-temperature	opt-p		
19	m2-pipe-length	opt-p		
20	m2-pipe-axial-direction	opt-p		
21	m2-outlet-temperature	opt-p		
22	m2-surface-normal-2 a	opt-p		
23	m2-inlet-temperature	opt-p		
24	m2-surface-normal-1 b	opt-p		
25	m2-neg-endpt-2 a	neg-1		
26	m2-neg-endpt-1 b	neg-2		
27	m2-neg-edge-3 a	neg-3		
			intensive	(region-a region-y) (region-a region-x) (region-a region-c) (region-a region-d) (region-a region-e)
28	m2-neg-edge-2 a	neg-4		

req-p -- required parameter opt-p -- optional parameter
p-obj -- positive object neg-n -- member of the nth negative set matched

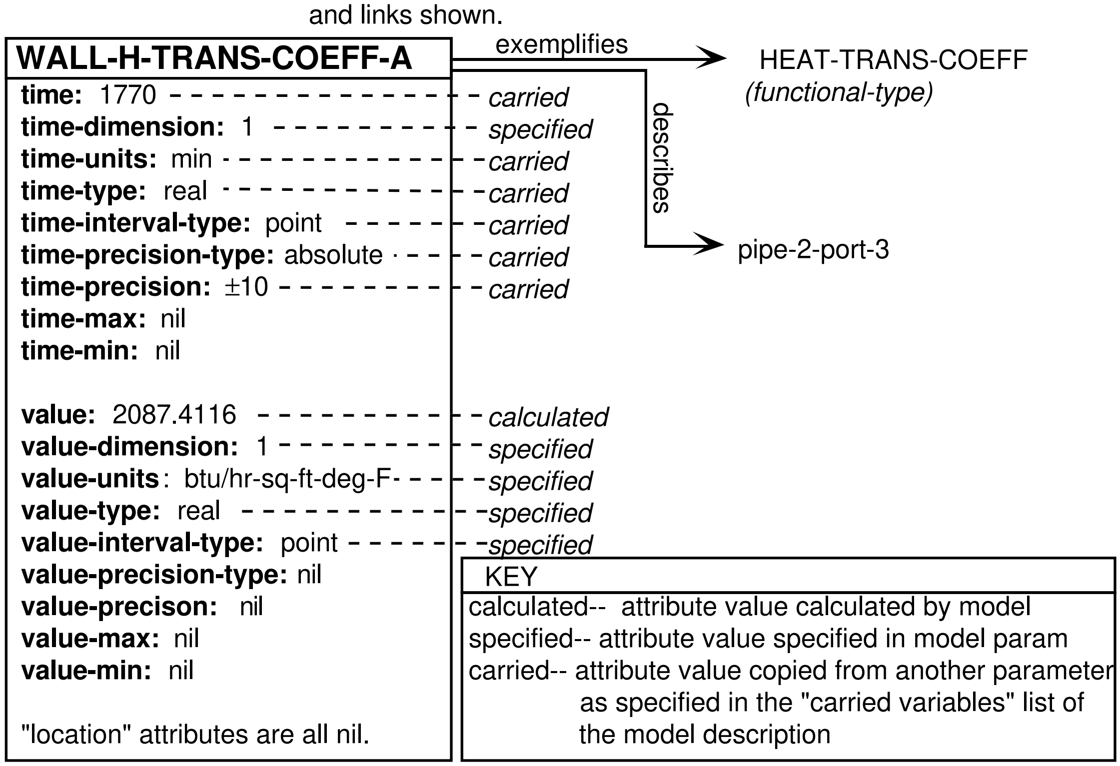
(**equal** m2-viscosity time-dimension m2-pipe-wall-temperature time-dimension)
(**equal** m2-viscosity time-dimension m2-outlet-temperature time-dimension)
(**equal** m2-viscosity time-dimension m2-inlet-temperature time-dimension)
(**equal** m2-viscosity time-units m2-pipe-wall-temperature time-units)
(**equal** m2-viscosity time-units m2-outlet-temperature time-units)
(**equal** m2-viscosity time-units m2-inlet-temperature time-units))

D.4.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

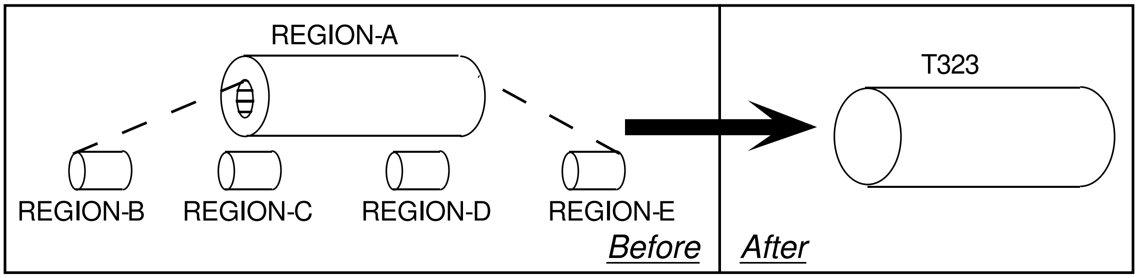
OUTPUTS:
(WALL-H-TRANS-COEFF-A)

SIDE-EFFECTS: The parameter object, HEAT-TRANS-COEFF-A, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

SIDE EFFECTS:



D.4.4 Checking Model Conditions

INPUTS:

(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:

T

((**m2-dittus-boelter:pipe-temperatures-ok?**
m2-inlet-temperature value
m2-outlet-temperature value
m2-pipe-wall-temperature value)

(**m2-dittus-boelter:l-over-d-ok?**
m2-pipe-length value m2-pipe-inside-diameter value)

(**any-common-portion?**
m2-viscosity time m2-viscosity time-type
m2-viscosity time-interval-type m2-viscosity time-precision
m2-viscosity time-precision-type m2-inlet-temperature time
m2-inlet-temperature time-type m2-inlet-temperature time-interval-type
m2-inlet-temperature time-precision m2-inlet-temperature time-precision-type)

(**any-common-portion?**
m2-viscosity time m2-viscosity time-type
m2-viscosity time-interval-type m2-viscosity time-precision
m2-viscosity time-precision-type m2-outlet-temperature time
m2-outlet-temperature time-type m2-outlet-temperature time-interval-type
m2-outlet-temperature time-precision m2-outlet-temperature time-precision-type)

(**any-common-portion?**
m2-viscosity time m2-viscosity time-type
m2-viscosity time-interval-type m2-viscosity time-precision
m2-viscosity time-precision-type m2-pipe-wall-temperature time
m2-pipe-wall-temperature time-type m2-pipe-wall-temperature time-interval-type
m2-pipe-wall-temperature time-precision m2-pipe-wall-temperature time-precision-type)

(**equal** m2-viscosity time-type m2-pipe-wall-temperature time-type)

(**equal** m2-viscosity time-type m2-outlet-temperature time-type)

(**equal** m2-viscosity time-type m2-inlet-temperature time-type)

D.4.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-port-3b 'm2-heat-transfer-coefficient 'pipe-2-port-3 'wall-h-trans-coeff-a)
OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m2-heat-transfer-coefficient	opt-p	wall-h-trans-coeff-a	
1	m2-port-3b	p-obj	pipe-2-port-3	
2	m2-stream-3	p-obj	stream-pipe-2-a3	
3	m2-pipe	p-obj	pipe-2	
4	m2-edge-2a	p-obj	t385	
5	m2-edge-1b	p-obj	t389	
6	m2-port-3a	p-obj	t390	
7	m2-pipe-inside-diameter	req-p	pipe-2-id	
8	m2-port-2a	p-obj	t384	
9	m2-port-1b	p-obj	t388	
10	m2-fluid-in-pipe	p-obj	t383	
11	m2-stream-2	p-obj	t387	
12	m2-stream-1	p-obj	t267	
13	m2-mass-flow-rate	req-p	t265	
14	m2-heat-capacity	req-p	heat-capacity-a	
15	m2-density	req-p	rho-a	
16	m2-thermal-conductivity	req-p	thermal-cond-a	
17	m2-viscosity	req-p	mu-a	
18	m2-pipe-wall-temperature	opt-p	nil	
19	m2-pipe-length	opt-p	nil	
20	m2-pipe-axial-direction	opt-p	pipe-2-axial-direction	
21	m2-outlet-temperature	opt-p	nil	
22	m2-surface-normal-2a	opt-p	port-normal-a2	
23	m2-inlet-temperature	opt-p	nil	
24	m2-surface-normal-1b	opt-p	port-normal-a1	
25	m2-neg-endpt-2a	neg-1	nil	
26	m2-neg-endpt-1b	neg-2	nil	
27	m2-neg-edge-3a	neg-3	nil	
28	m2-neg-edge-2a	neg-4	nil	
29	m2-neg-s-normal-2a	neg-5	nil	
30	m2-neg-edge-1b	neg-6	nil	
31	m2-neg-s-normal-1b	neg-7	nil	
32	m2-neg-port	neg-8	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.4 Dittus-Boelter Model and PIPES-2 Equipment Description

D.4.1 Statistics on the Matching

Equipment Description:	PIPES-2
Model Description:	DITTUS-BOELTER
Goal:	calculate VALUES
Max. partial matches:	120
Complete Matches:	1
Rematches:	35
Run time:	1:25 (hr:min)
Objects created (kept):	338 (12)
Links created (kept):	3373 (56)
Separately triggered reconfigurations:	3
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	Yes

D.4.2 Generating the Potential Match Set

INPUTS:
(SELECT-MODELS 'wall-h-trans-coeff-a 'value 'describes 'pipe-2-port-3 'values

OUTPUTS:
(
(wall-h-trans-coeff-a value describes pipe-2-port-3 values m2-heat-transfer-coefficient
m2-port-3b)
)

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
24 m2-surface-normal-1 b	opt-p	intensive	(region-c region-d region-e) (region-c region-y region-d region-b) (region-c region-y region-d region-e) (region-c region-y region-e region-b) (region-c region-y region-b region-a) (region-c region-b region-d region-e) (region-c region-b region-d region-a) (region-c region-b region-e region-a) (region-c region-b region-a region-x) (region-c region-y region-d region-b region-e) (region-c region-y region-d region-b region-a) (region-c region-y region-e region-b region-a) (region-c region-y region-b region-a region-x) (region-c region-b region-d region-e region-a)
25 m2-neg-endpt-2 a	neg-1		
26 m2-neg-endpt-1 b	neg-2		
27 m2-neg-edge-3 a	neg-3		
28 m2-neg-edge-2 a	neg-4		
29 m2-neg-s-normal-2 a	neg-5		
30 m2-neg-edge-1 b	neg-6		
31 m2-neg-s-normal-1 b	neg-7		
32 m2-neg-port	neg-8		

req-p -- required parameter   opt-p -- optional parameter
p-obj -- positive object   neg-n -- member of the nth negative set matched

D.3.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)		**notes**	**Reconfiguration type**	**Reconfigurations tried**
0	m2-heat-transfer-coefficient	req-p		
1	m2-port-3b	p-obj		
2	m2-stream-3	p-obj		
3	m2-pipe	p-obj		
4	m2-edge-2a	p-obj		
5	m2-edge-1b	p-obj		
6	m2-port-3a	p-obj		
7	m2-pipe-inside-diameter	req-p		
8	m2-port-2a	p-obj		
9	m2-port-1b	p-obj		
10	m2-fluid-in-pipe	p-ojb		
11	m2-stream-2	p-obj		
12	m2-stream-1	p-obj		
			intensive	(region-c region-y)
				(region-c region-b)
				(region-c region-d)
				(region-c region-e)
				(region-c region-y region-d)
				(region-c region-y region-e)
				(region-c region-y region-b)
				(region-c region-b region-d)
				(region-c region-b region-e)
				(region-c region-b region-a)
13	m2-mass-flow-rate	req-p		
14	m2-heat-capacity	req-p		
15	m2-density	req-p		
16	m2-thermal-conductivity	req-p		
17	m2-viscosity	req-p		
18	m2-pipe-wall-temperature	opt-p		
19	m2-pipe-length	opt-p		
20	m2-pipe-axial-direction	opt-p		
21	m2-outlet-temperature	opt-p		
22	m2-surface-normal-2a	opt-p		
23	m2-inlet-temperature	opt-p		

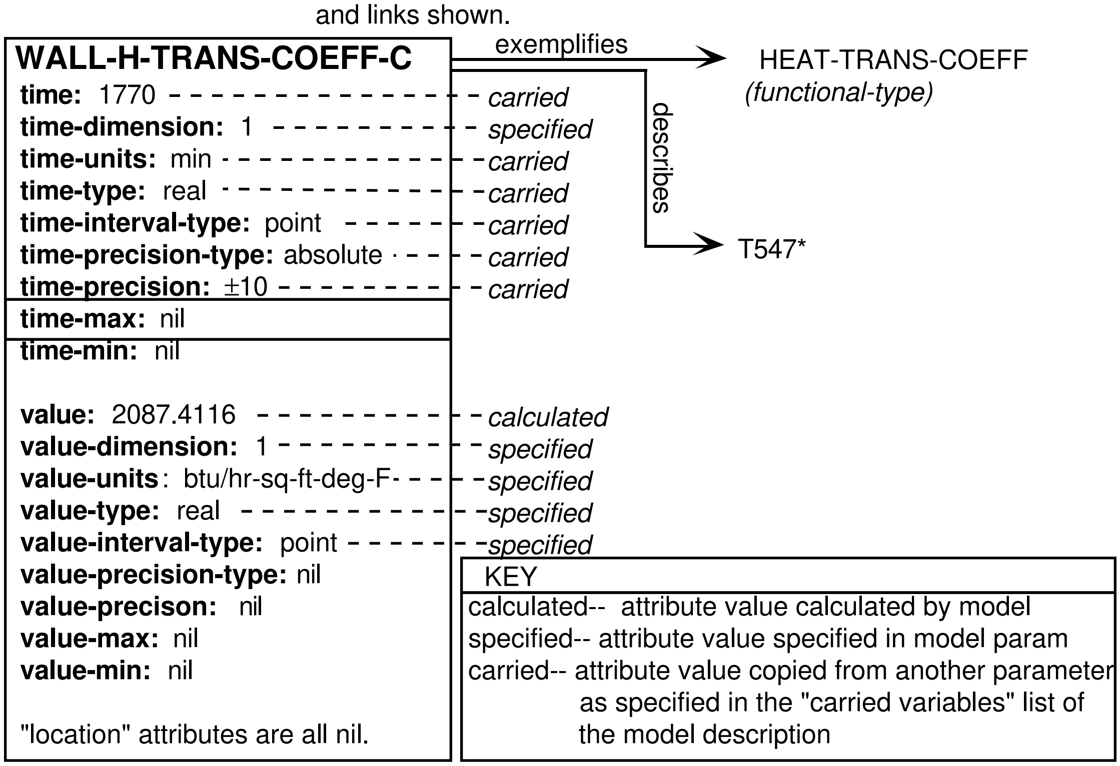
req-p -- required parameter　　opt-p -- optional parameter
p-obj -- positive object　　neg-n -- member of the nth negative set matched

D.3.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

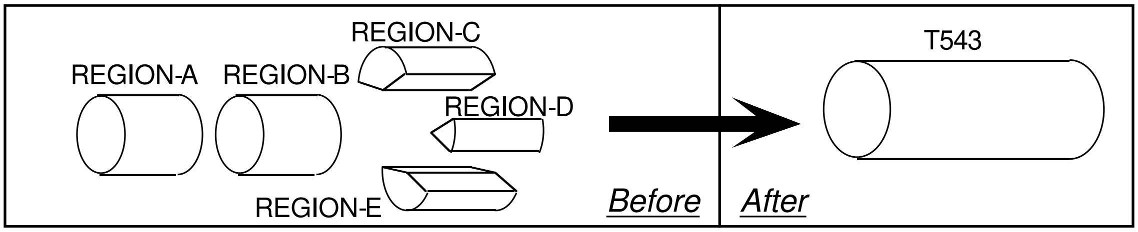
OUTPUTS:
(WALL-H-TRANS-COEFF-C)

SIDE-EFFECTS: The Parameter Object, HEAT-TRANS-COEFF, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

SIDE EFFECTS:



D.3.4 Checking Model Conditions

INPUTS:
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))

OUTPUTS:
T

((**m2-dittus-boelter:pipe-temperatures-ok?**
m2-inlet-temperature value
m2-outlet-temperature value
m2-pipe-wall-temperature value)

(**m2-dittus-boelter:l-over-d-ok?**
m2-pipe-length value
m2-pipe-inside-diameter value)

(**any-common-portion?**
m2-viscosity time m2-viscosity time-type
m2-viscosity time-interval-type m2-viscosity time-precision
m2-viscosity time-precision-type m2-outlet-temperature time
m2-outlet-temperature time-type time-precision m2-outlet-temperature time-interval-type
m2-outlet-temperature m2-outlet-temperature time-precision-type)

(**any-common-portion?**
m2-viscosity time m2-viscosity time-type
m2-viscosity time-interval-type m2-viscosity time-precision
m2-viscosity time-precision-type m2-pipe-wall-temperature time
m2-pipe-wall-temperature time-type m2-pipe-wall-temperature time-interval-type
m2-pipe-wall-temperature time-precision m2-pipe-wall-temperature time-precision-type)

(**equal** m2-viscosity time-type m2-pipe-wall-temperature time-type)

(**equal** m2-viscosity time-type m2-outlet-temperature time-type)

(**equal** m2-viscosity time-dimension m2-pipe-wall-temperature time-dimension)

(**equal** m2-viscosity time-dimension m2-outlet-temperature time-dimension)

(**equal** m2-viscosity time-units m2-pipe-wall-temperature time-units)

(**equal** m2-viscosity time-units m2-outlet-temperature time-units))

D.3.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-port-3b 'm2-heat-transfer-coefficient 'pipe-2-port-3c
'wall-h-trans-coeff-c)
OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned	notes	Vector-list of matching Equipment Objects Returned	
		vector #1	vector #2
0 m2-heat-transfer-coefficient	opt-p	wall-h-trans-coeff-c	
1 m2-port-3b	p-obj	t547	
2 m2-stream-3	p-obj	t548	
3 m2-pipe	p-obj	pipe-2	
4 m2-edge-2a	p-obj	t545	
5 m2-edge-1b	p-obj	t546	
6 m2-port-3a	p-obj	t544	
7 m2-pipe-inside-diameter	req-p	pipe-2-id	
8 m2-port-2a	p-obj	t549	
9 m2-port-1b	p-obj	t550	
10 m2-fluid-in-pipe	p-ojb	t543	
11 m2-stream-2	p-obj	t425	
12 m2-stream-1	p-obj	stream-x2-a1	
13 m2-mass-flow-rate	req-p	pipe-2-inlet-flow	
14 m2-heat-capacity	req-p	heat-capacity-c	
15 m2-density	req-p	rho-c	
16 m2-thermal-conductivity	req-p	thermal-cond-c	
17 m2-viscosity	req-p	mu-c	
18 m2-pipe-wall-temperature	opt-p	nil	
19 m2-pipe-length	opt-p	nil	
20 m2-pipe-axial-direction	opt-p	pipe-2-axial-direction	
21 m2-outlet-temperature	opt-p	nil	
22 m2-surface-normal-2a	opt-p	port-normal-c2	
23 m2-inlet-temperature	opt-p	fluid-temp-port-a1	
24 m2-surface-normal-1b	opt-p	port-normal-a1	
25 m2-neg-endpt-2a	neg-1	nil	
26 m2-neg-endpt-1b	neg-2	nil	
27 m2-neg-edge-3a	neg-3	nil	
28 m2-neg-edge-2a	neg-4	nil	
29 m2-neg-s-normal-2a	neg-5	nil	
30 m2-neg-edge-1b	neg-6	nil	
31 m2-neg-s-normal-1b	neg-7	nil	
32 m2-neg-port	neg-8	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.3 Dittus-Boelter Model and PIPES-1 Equipment Description

D.3.1 Statistics on the Matching

Equipment Description:	PIPES-1
Model Description:	DITTUS-BOELTER
Goal:	calculate VALUES
Max. partial matches:	312
Complete Matches:	1
Rematches:	23
Run time:	1:29 (hr:min)
Objects created (kept):	633 (11)
Links created (kept):	3998 (67)
Separately triggered reconfigurations:	2
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	Yes
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	Yes
Negative set triggered reconfiguration?	Yes

D.3.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS 'wall-h-trans-coeff-c 'value 'describes 'pipe-2-port-3c 'values)
```

OUTPUTS:

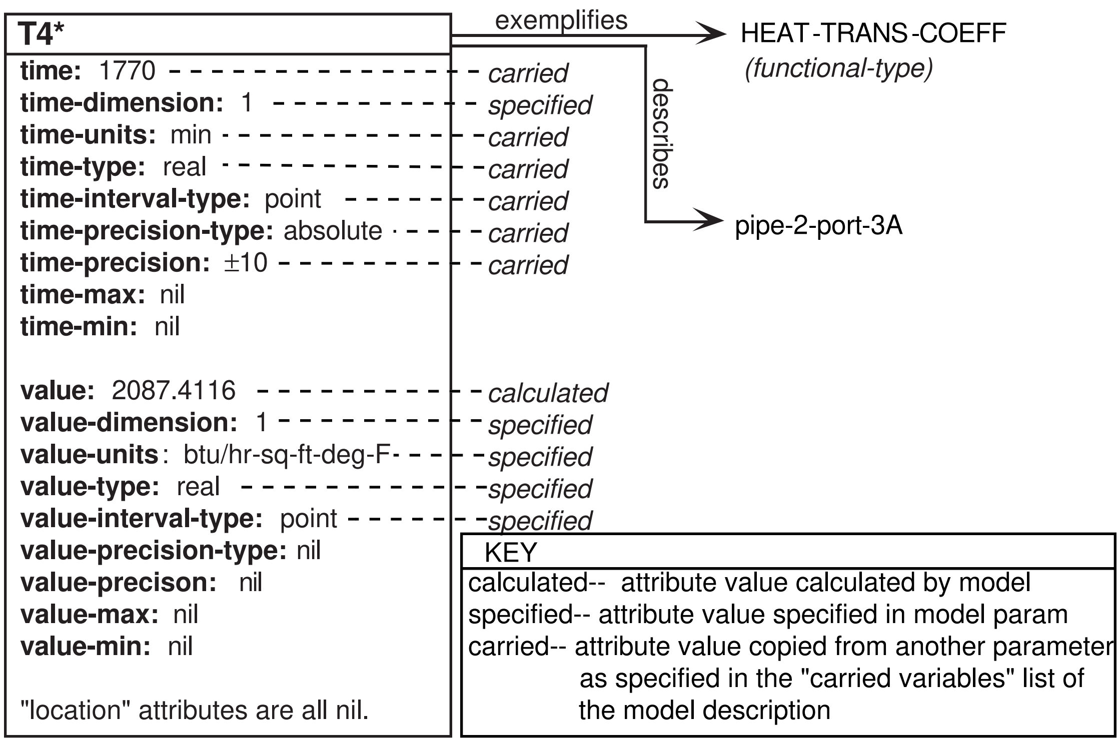
```
(
 (wall-h-trans-coeff-c value describes pipe-2-port-3c values m2-heat-transfer-coefficient
          m2-port-3b)
)
```

D.2.5 Executing the Model

INPUTS:
(EXECUTE-MODEL assoc-list (first vector-list))

OUTPUTS:
(T4)

SIDE-EFFECTS: The parameter object, T4, is created with attributes and links shown.



* name of parameter object is created by the GENTEMP function which generates unique names, all beginning with "T" followed by an integer.

D.2.6 Summary of Merges and Splits Tested

Assoc-list of model objects (indicates order of matching)	**notes**	**Reconfiguration type**	**Reconfigurations tried**
No reconfigurations made.			
req-p -- required parameter p-obj -- positive object	opt-p -- optional parameter neg-n -- member of the nth negative set matched		

D.2.4 Checking Model Conditions

INPUTS:

```
(CHECK-MODEL-CONDITIONS 'approx-conditions assoc-list (first vector-list))
```

OUTPUTS:

```
T
((m2-dittus-boelter:pipe-temperatures-ok?
  m2-inlet-temperature        value
  m2-outlet-temperature       value
  m2-pipe-wall-temperature    value)

 (m2-dittus-boelter:l-over-d-ok?
  m2-pipe-length   value   m2-pipe-inside-diameter   value)

 (any-common-portion?
  m2-viscosity               time                   m2-viscosity               time-type
  m2-viscosity               time-interval-type     m2-viscosity               time-precision
  m2-viscosity               time-precision-type    m2-outlet-temperature      time
  m2-outlet-temperature      time-type              m2-outlet-temperature      time-interval-type
  m2-outlet-temperature      time-precision         m2-outlet-temperature      time-precision-type)

 (any-common-portion?
  m2-viscosity               time                   m2-viscosity               time-type
  m2-viscosity               time-interval-type     m2-viscosity               time-precision
  m2-viscosity               time-precision-type    m2-pipe-wall-temperature   time
  m2-pipe-wall-temperature   time-type              m2-pipe-wall-temperature   time-interval-type
  m2-pipe-wall-temperature   time-precision         m2-pipe-wall-temperature   time-precision-type)

 (equal   m2-viscosity   time-type   m2-pipe-wall-temperature   time-type)

 (equal   m2-viscosity   time-type   m2-outlet-temperature   time-type)

 (equal   m2-viscosity   time-dimension   m2-pipe-wall-temperature   time-dimension)

 (equal   m2-viscosity   time-dimension   m2-outlet-temperature   time-dimension)

 (equal   m2-viscosity   time-units   m2-pipe-wall-temperature   time-units)

 (equal   m2-viscosity   time-units   m2-outlet-temperature   time-units))
```

D.2.3 Match/Reconfigure

INPUTS:
(DO-SIMPLE-MATCH 'm2-fluid-in-pipe 'm2-heat-capacity 'region-A 'heat-capacity-A)

OUTPUTS: (association-list and vector-list)

Assoc-List of Model Objects Returned		notes	Vector-list of matching Equipment Objects Returned	
			vector #1	vector #2
0	m2-heat-capacity	req-p	heat-capacity-a	
1	m2-fluid-in-pipe	p-obj	region-a	
2	m2-port-3a	p-obj	port-a3	
3	m2-port-2a	p-obj	port-a2	
4	m2-port-1b	p-obj	port-a1	
5	m2-density	req-p	rho-a	
6	m2-thermal-conductivity	req-p	thermal-cond-a	
7	m2-viscosity	req-p	mu-a	
8	m2-stream-3	p-obj	stream-pipe-2-a3	
9	m2-edge-1b	p-obj	pipe-1-edge-2-1	
10	m2-edge-2a	p-obj	pipe-2-edge-2-1	
11	m2-stream-2	p-obj	stream-a2-y1	
12	m2-stream-1	p-obj	stream-x2-a1	
13	m2-mass-flow-rate	req-p	pipe-2-inlet-flow	
14	m2-port-3b	p-obj	pipe-2-port-3a	
15	m2-pipe	p-obj	pipe-2	
16	m2-pipe-inside-diameter	req-p	pipe-2-id	
17	m2-outlet-temperature	opt-p	nil	
18	m2-surface-normal-2a	opt-p	port-normal-a2	
19	m2-inlet-temperature	opt-p	fluid-temp-port-a1	
20	m2-surface-normal-1b	opt-p	port-normal-a1	
21	m2-pipe-wall-temperature	opt-p	nil	
22	m2-heat-transfer-coefficient	opt-p	wall-h-trans-coeff-a	
23	m2-pipe-length	opt-p	nil	
24	m2-pipe-axial-direction	opt-p	pipe-2-axial-direction	
25	m2-neg-port	neg-1	nil	
26	m2-neg-edge-3a	neg-2	nil	
27	m2-neg-edge-2a	neg-3	nil	
28	m2-neg-s-normal-2a	neg-4	nil	
29	m2-neg-edge-1b	neg-5	nil	
30	m2-neg-s-normal-1b	neg-6	nil	
31	m2-neg-endpt-1b	neg-7	nil	
32	m2-neg-endpt-2a	neg-8	nil	

* different than vector #1. p-obj -- positive object neg-n -- member of the nth negative set matched
req-p -- required parameter opt-p -- optional parameter

D.2 Dittus-Boelter Model and PIPES-0 Equipment Description

D.2.1 Statistics on the Matching

Equipment Description:	PIPES-0
Model Description:	DITTUS-BOELTER
Goal:	find EFFECTS
Max. partial matches:	4
Complete matches:	1
Rematches:	0
Run time:	0:01 (hr:min)
Objects created (kept):	1 (1)
Links created (kept):	2 (2)
Separately triggered reconfigurations:	0
Used simple matching?	Yes
Used negative matching?	Yes
Used intensive reconfiguration?	No
Used part-whole reconfiguration?	No
Positive object triggered reconfiguration?	No
Parameter triggered reconfiguration?	No
Negative set triggered reconfiguration?	No

D.2.2 Generating the Potential Match Set

INPUTS:

```
(SELECT-MODELS 'heat-capacity-a 'value 'describes
               'region-a 'effects)
```

OUTPUTS:

```
(
 (heat-capacity-a-value describes region-a effects m2-heat-capacity m2-fluid-in-pipe)
)
```

Appendix D

Implemented Matching/Reconfiguration Cases

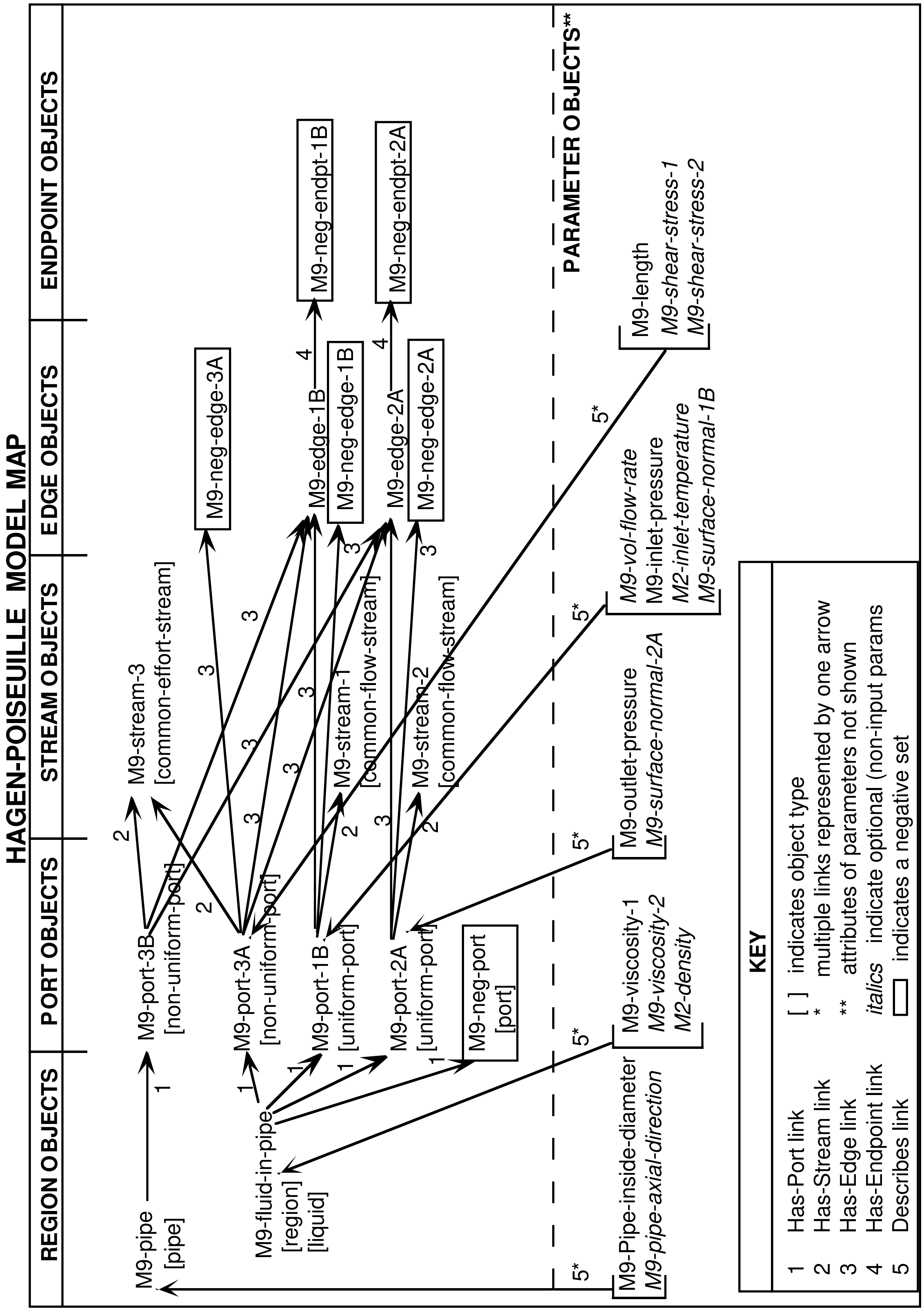
D.1 Introduction

This appendix contains the inputs and outputs for the 16 matchings made as tests of the methods developed in this thesis as well as some statistics that were collected on the runs. Inputs and outputs from each of the four steps, potential match set generation (PMSG), match-reconfigure (M-R), check model conditions (CMC), and execute the model (EXEC-M) are listed here. The side effects, if any, are also listed. A model matching system would normally be called by a diagnosis system, and besides returning the values calculated by the models after matching, the model matcher would also return the relations required by the model. The list of relations is not shown here because it is a straight forward instantiation of the model relations with equipment objects that were matched to model objects.

THE HAGEN-POISEUILLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
19. **equal** (function name)	m9-shear-stress-1 m9-shear-stress-2	value-units value-units
20. **m9-hagen:: fluid-is-newtonian?**	m9-shear-stress-1 m9-viscosity-1 m9-shear-stress-2 m9-viscosity-2	value value value value
21. **m9-hagen:: flow-is-laminar?**	m9-pipe-inside-diameter m9-vol-flow-rate m9-density m9-viscosity-1	value value value value
22. **possibly-equal?**	m9-pressure-deriv nil	value, value-type, value-interval-type value-precision, value-precision-type 0, real, point, nil, nil
34. **vectors-parallel?**	m9-pipe-axial-direction m9-surface-normal-2a	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type
35. **vectors-parallel?**	m9-pipe-axial-direction m9-surfoace-normal-1b	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type

THE HAGEN-POISEUILLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
10. **any-common-portion?** (function name)	m9-inlet-pressure m9-outlet-pressure	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
11. **any-common-portion?**	m9-inlet-pressure m9-viscosity-1	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
12. **any-common-portion?**	m9-inlet-pressure m9-pipe-inside-diameter	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
13. **any-common-portion?**	m9-shear-stress-1 m9-viscosity-1	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
14. **any-common-portion?**	m9-inlet-pressure m9-length	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
15. **any-common-portion?**	m9-inlet-pressure m9-density	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
16. **any-common-portion?**	m9-inlet-pressure m9-vol-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
17. **any-common-portion?**	m9-shear-stress-2 m9viscosity-2	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
18. **not-any-common-portion?**	m9-shear-stress-1 m9-shear-stress-2	value, value-type, value-interval-type value-precision, value-precision-type value, value-type, value-interval-type value-precision, value-precision-type

THE HAGEN-POISEUILLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Carried Variables (continued)	From: m9-inlet-pressure To: m9-vol-flow-rate From: m9-inlet-pressure To: m9-vol-flow-rate	time-precision time-precision time time
Call Form **m9-hagen:: calculate-hagen-flow** (function name)	m9-viscosity-1 m9-pipe-inside-diameter m9-length m9-inlet-pressure m9-outlet-pressure	value value value value value
Return Form	m9-vol-flow-rate	value
Enabling Conditions	nil	
Approximation Conditions		
1. **equal** (function name)	m9-viscosity-1 m9-inlet-pressure	time-units time-units
2. **equal**	m9-viscosity-1 m9-outlet-pressure	time-units time-units
3. **equal**	m9-viscosity-1 m9-pressure-deriv	time-units time-units
4. **equal**	m9-viscosity-1 m9-shear-stress-1	time-units time-units
5. **equal**	m9-viscosity m9-shear-stress-2	time-units time-units
6. **equal**	m9-viscosity-1 m9-viscosity-2	time-units tme-units
7. **equal**	m9-viscosity-1 m9-density	time-units time-units
8. **equal**	m9-viscosity-1 m9-length	time-units time-units
9. **equal**	m9-viscosity-1 m9-pipe-inside-diameter	time-units time-units



HAGEN-POISEUILLE MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
ENDPOINT OBJECTS
PARAMETER OBJECTS**
M9-pipe
[pipe]
M9-fluid-in-pipe
[region]
[liquid]
M9-port-3B
[non-uniform-port]
M9-port-3A
[non-uniform-port]
M9-port-1B
[uniform-port]
M9-port-2A
[uniform-port]
M9-neg-port
[port]
M9-stream-3
[common-effort-stream]
M9-stream-1
[common-flow-stream]
M9-stream-2
[common-flow-stream]
M9-neg-edge-3A
M9-edge-1B
M9-neg-edge-1B
M9-edge-2A
M9-neg-edge-2A
M9-neg-endpt-1B
M9-neg-endpt-2A
M9-Pipe-inside-diameter
M9-pipe-axial-direction
M9-viscosity-1
M9-viscosity-2
M2-density
M9-outlet-pressure
M9-surface-normal-2A
M9-vol-flow-rate
M9-inlet-pressure
M2-inlet-temperature
M9-surface-normal-1B
M9-length
M9-shear-stress-1
M9-shear-stress-2
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Endpoint link
5 Describes link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set

21 and 22 ensure that flow is laminar and is steady state, respectively. Since the fluid section must be a cylinder with its ends perpendicular to the pipe axis, *Conditions* 23 and 24 specify that surface normals on both ends of the cylinder must be parallel to the pipe axial direction. The other conditions (1-17) ensure that the parameters all have the appropriate time relationships to each other.

C.8.2 Statistics

STATISTICS FOR HAGEN-POISEUILLE MODEL AND DESCRIPTION	
model size:	1 K
model internal method:	mathematical equation
model author:	J. Murdock
model description size:	47 K
individuals in model description:	2
model map representation:	33 objects, 70 links

C.8.3 The Implemented Model Description

THE HAGEN-POISEUILLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m9-viscosity-1 m9-pipe-inside-diameter m9-length m9-inlet-pressure m9-outlet-pressure	value value value value value
Output Variables	m9-vol-flow-rate	value
Causal Variables	m9-viscosity-1 m9-pipe-inside-diameter m9-length m9-inlet-pressure m9-outlet-pressure	value value value value value
Affected Variables	m9-vol-flow-rate	value
Carried Variables	From: m9-inlet-pressure To: m9-vol-flow-rate	time-units time-units
	From: m9-inlet-pressure To: m9-vol-flow-rate	time-type time-type
(continued)	From: m9-inlet-pressure To: m9-vol-flow-rate	time-precision-type time-precision-type

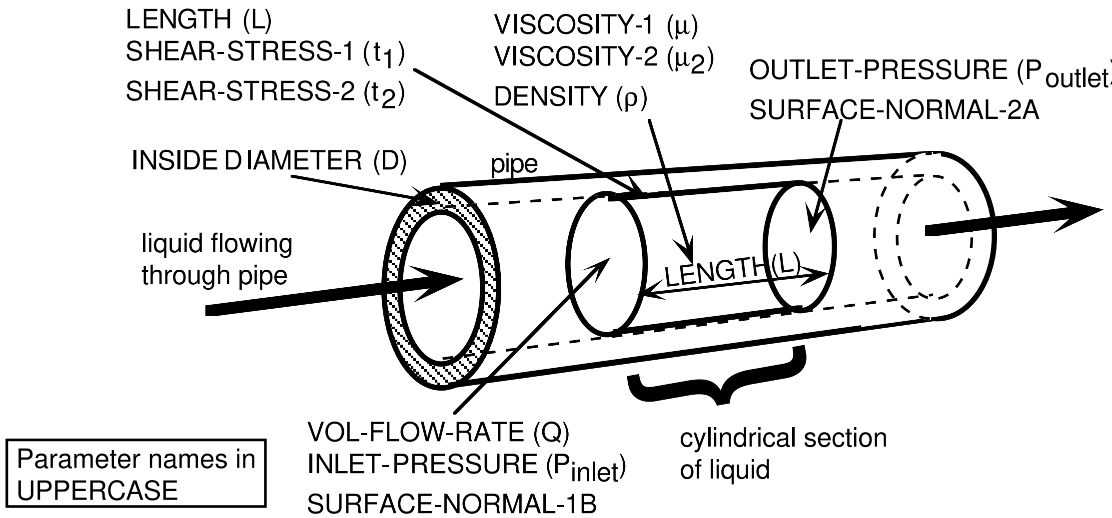


Figure C.4: The individuals and parameters required by the Hagen-Poiseuille model.

C.8 The Hagen-Poiseuille Model Description

C.8.1 The Model and its Requirements

The Hagen-Poiseuille model is a computer implementation of the Hagen-Poiseuille flow law which can be found in fluid flow textbooks such as [33]. The model applies to laminar steady-state flow of imcompressible newtonian fluids in circular pipes. This model requires identification of a cylindrical cross-section of fluid as an individual as shown in Figure C.4. The variables in the Hagen-Poiseuille law,

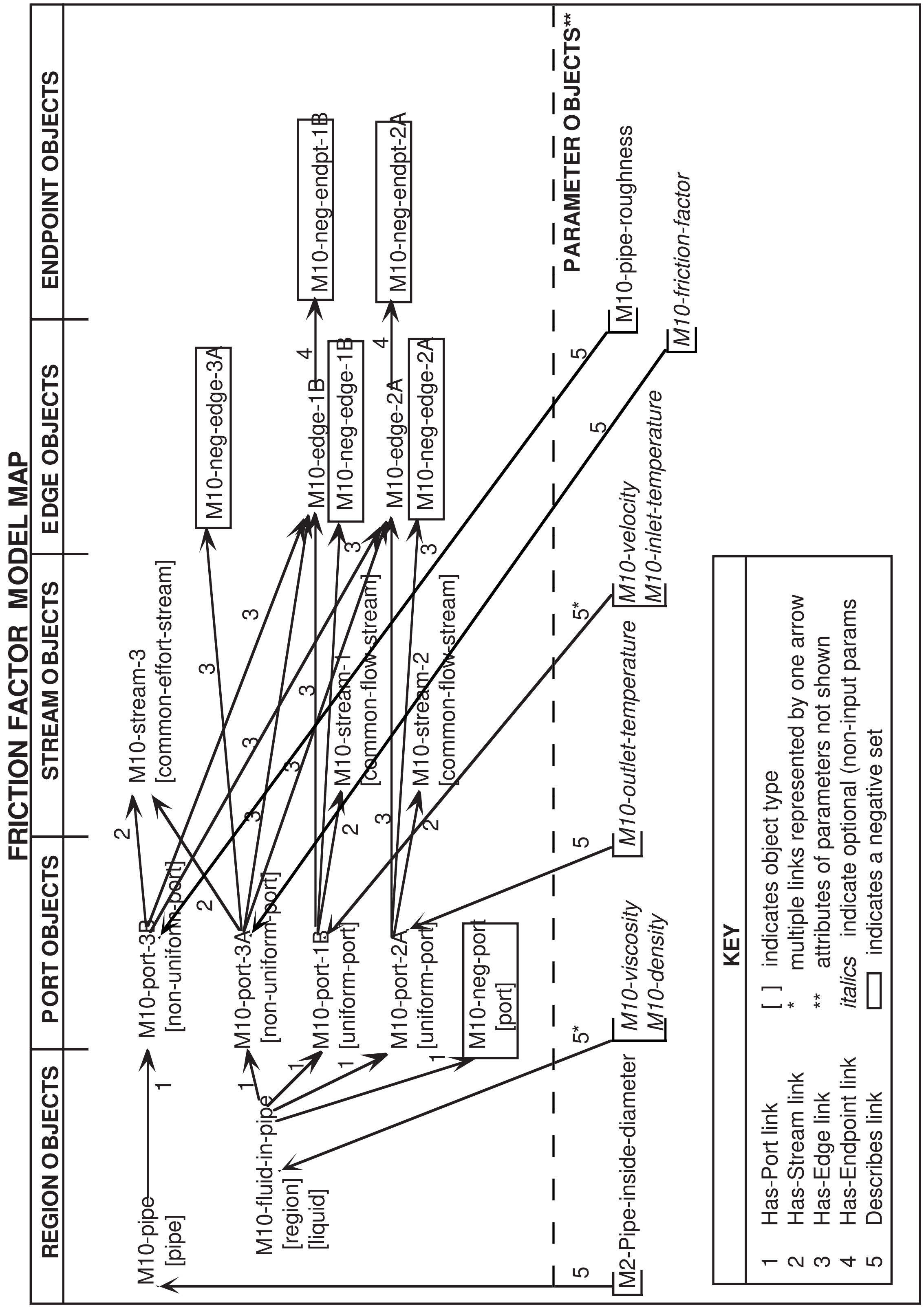
$$Q = \frac{-\pi R^4 \delta P}{8 \mu L},$$

where R is the pipe radius ($D/2$), δP is $P_{outlet} - P_{inlet}$, and the rest of the variables are defined in Figure C.4. Each variable has a particular spatial extent within the fluid cylinder or the pipe. For example, P_{outlet} describes the right end of the fluid cylinder, and μ (viscosity) describes the entire volume of the cylinder. The Hagen-Poiseuille law is applicable to fluid cylinders that cross boundaries of connected pipes. However, to specify this physical situation, one needs to specify a pattern, any number of pipes connected end to end. This kind of specification and the associated matching algorithms have not been implemented for this thesis. Thus the model is treated as if it is valid for fluid cylinders residing in one pipe.

This model requires a geometric situation very similar to the Dittus-Boelter model (Section C.2). We include negative sets in the model map to ensure that the liquid cylinder completely fills the pipe cross-section, that the cylinder has only 3 adjacencies (one with the pipe and two with adjacent fluid regions), and that nothing is "hiding" inside the cylinder. The map specifies that the fluid be a liquid so that the incompressibility condition is met. *Approximation Conditions* 18–20 check that the fluid is newtonian (the viscosity is constant with respect to shear stress). *Conditions*

THE FRICTION FACTOR MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
10. **any-common-portion?** (function name)	m10-density m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
11. **any-common-portion?**	m10-viscosity m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
12. **any-common-portion?**	m10-inlet-temperature m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
13. **any-common-portion?**	m10-outlet-temperature m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
14. **equal**	m10-inlet-temperature m10-outlet-temperature	value-units value-units
15. **possibly-equal?**	m10-outlet-temperature m10-inlet-temperature	value, value-type, value-interval-type, value-precision, value-precision-type value, value-type, value-interval-type, value-precision, value-precision-type
16. **m10-friction::Re-ok?**	m10-pipe-inside-diameter m10-velocity m10-density m10-viscosity	value value value value
17. **m10-friction::kD-ok?**	m10-pipe-roughness m10-pipe-inside-diameter	value value
18. **m10-friction:: Re-and-kD-ok?**	m10-pipe-inside-diameter m10-velocity m10-density m10-viscosity m10-pipe-roughness	value value value value value

THE FRICTION FACTOR MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Call Form **m10-friction::** **fanning-friction-factor** (function name)	m10-pipe-inside-diameter m10-pipe-roughness	value value
Return Form	m10-friction-factor	value
Enabling Conditions	nil	
Approximation Conditions		
1. **equal** (function name)	m10-pipe-roughness m10-pipe-inside-diameter	time-units time-units
2. **equal**	m10-pipe-roughness m10-velocity	time-units time-units
3. **equal**	m10-pipe-roughness m10-density	time-units time-units
4. **equal**	m10-pipe-roughness m10-viscosity	time-units time-units
5. **equal**	m10-pipe-roughness m10-inlet-temperature	time-units time-units
6. **equal**	m10-pipe-roughness m10-outlet-temperature	time-units tme-units
7. **any-common-portion?**	m10-pipe-roughness m10-pipe-inside-diameter	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
8. **any-common-portion?**	m10-pipe-roughness m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
9. **any-common-portion?**	m10-pipe-inside-diameter m10-velocity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type


FRICTION FACTOR MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
ENDPOINT OBJECTS
M10-pipe
[pipe]
M10-fluid-in-pipe
[region]
[liquid]
M10-port-3B
[non-uniform-port]
M10-port-3A
[non-uniform-port]
M10-port-1B
[uniform-port]
M10-port-2A
[uniform-port]
M10-neg-port
[port]
M10-stream-3
[common-effort-stream]
M10-stream-1
[common-flow-stream]
M10-stream-2
[common-flow-stream]
M10-neg-edge-3A
M10-edge-1B
M10-neg-edge-1B
M10-edge-2A
M10-neg-edge-2A
M10-neg-endpt-1B
M10-neg-endpt-2A
PARAMETER OBJECTS**
M2-Pipe-inside-diameter
M10-viscosity
M10-density
M10-outlet-temperature
M10-velocity
M10-inlet-temperature
M10-pipe-roughness
M10-friction-factor
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Endpoint link
5 Describes link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set

of this model to cases where the inlet and outlet temperatures are the same to within their known precisions. This model is an empirical correlation of data collected at particular Reynolds numbers (Re) and pipe roughnesses (k), and so *Conditions* 16–18 check that these values are within bounds.

C.7.2 Statistics

STATISTICS FOR FRICTION FACTOR MODEL AND DESCRIPTION	
model size:	1 K
model internal method:	mathematical equation
model author:	J. Murdock
model description size:	30 K
individuals in model description:	2
model map representation:	25 objects, 54 links

C.7.3 The Implemented Model Description

THE FRICTION FACTOR MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m10-pipe-inside-diameter m10-pipe-roughness	value value
Output Variables	m10-friction-factor	value
Causal Variables	m10-pipe-inside-diameter m10-pipe-roughness m10-velocity m10-density m10-viscosity	value value value value value
Affected Variables	m10-friction-factor	value
Carried Variables	From: m10-velocity To: m10-friction-factor	time time
	From: m10-velocity To: m10-friction-factor	time-interval-type time-interval-type
	From: m10-velocity To: m10-friction-factor	time-units time-units
	From: m10-velocity To: m10-friction-factor	time-precision-type time-precision-type
	From: m10-velocity To: m10-friction-factor	time-precision time-precision

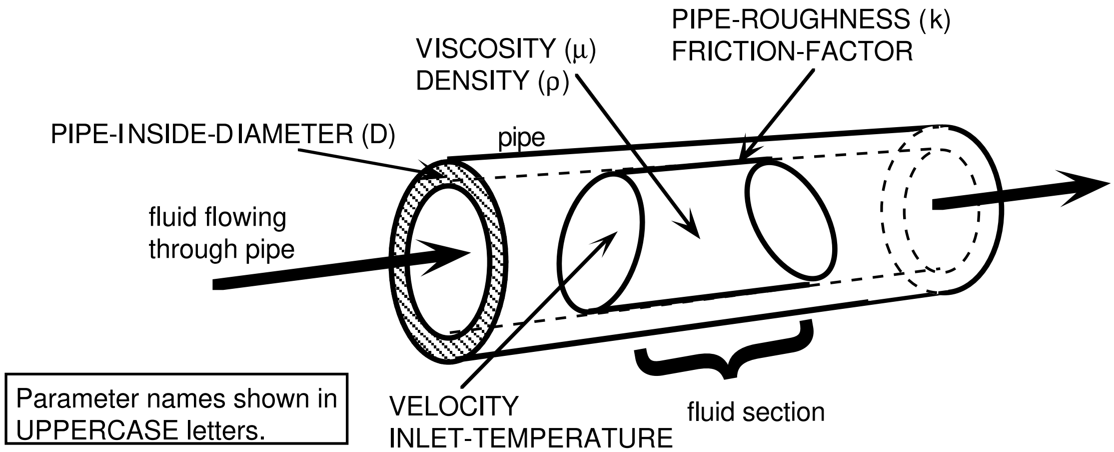


Figure C.3: The individuals and parameters required by the Friction Factor model.

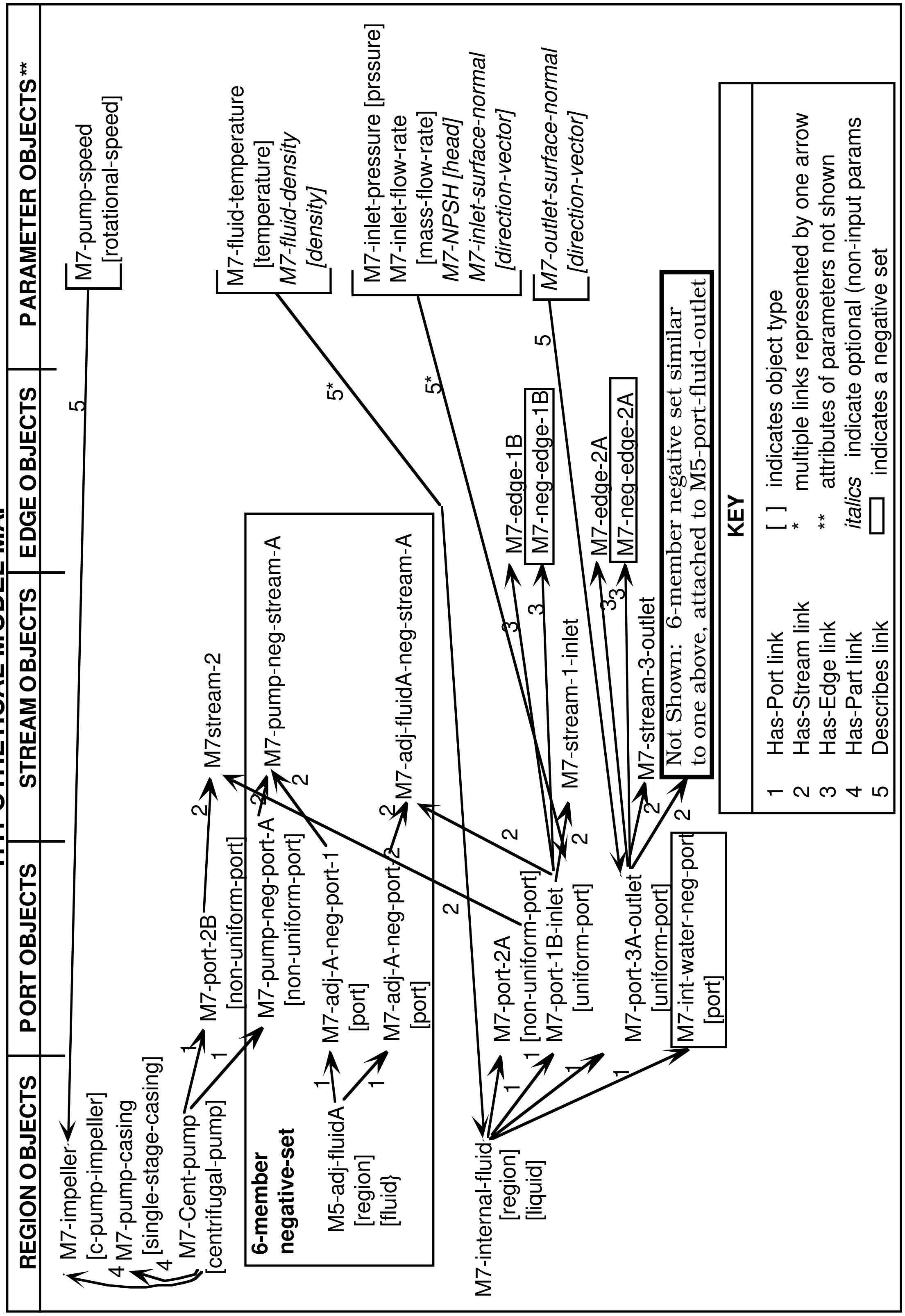
C.7 The Friction Factor Model Description

C.7.1 The Model and its Requirements

Friction factors are a form of drag coefficient used to calculate pressure loss of fluid flowing in round pipes due to viscous drag. The model we have incorporated is an empirical correlation of friction factor with pipe roughness and diameter given by [33]. This model requires that the pipe be full of the flowing fluid and that we identify a section of the fluid as shown in Figure C.3. Unlike the Dittus-Boelter model (Section C.2), this model does not require that the fluid section be a cylinder with parallel ends. The section is not allowed to cross pipe boundaries, though, because the model requires the pipe roughness variable to describe the whole pipe surface adjacent to the liquid. It would be acceptable to calculate a friction factor for many pipes connected in series as long as they had the same friction factor. However, this would require describing a pattern (the series of pipes), and the associated matching method was not implemented during this work. Thus the model description is restricted to a single pipe.

The model map specifies that the fluid section fill the cross-section of the pipe and that it not cross pipe boundaries by using negative sets. The singleton negative set containing `M10-neg-port` (shown in Section C.7.3) ensures that the fluid section can have no other ports beyond its inlet, outlet, and surface with one pipe. This negative port also guarantees that nothing else can be "hiding" inside the liquid section. The single port to the pipe is guaranteed to be a cylindrical surface because it must have exactly two edges (guaranteed by a negative edge `M10-neg-edge-3A`) and the edges are guaranteed to be closed curves. They can have exactly 0 endpoints, as specified by a negative endpoint (`M10-neg-endpt-1B, M10-neg-endpt-2A`) attached to each edge.

The approximation conditions in the model map specify the conditions that must be satisfied by paramter attribute values for the Friction Factor model to be valid. Besides requiring that the parameters occur at the same time (*Conditions* 1–13), The model requires that heat is not being transferred to or from the fluid as it flows through the pipe. *Conditions* 14 and 15 restrict application


HYPOTHETICAL MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
PARAMETER OBJECTS**
M7-impeller
[c-pump-impeller]
M7-pump-casing
[single-stage-casing]
M7-Cent-pump
[centrifugal-pump]
6-member negative-set
M5-adj-fluidA
[region]
[fluid}
M7-internal-fluid
[region]
[liquid]
M7-port-2B
[non-uniform-port]
M7-pump-neg-port-A
[non-uniform-port]
M7-adj-A-neg-port-1
[port]
M7-adj-A-neg-port-2
[port]
M7-port-2A
[non-uniform-port]
M7-port-1B-inlet
[uniform-port]
M7-port-3A-outlet
[uniform-port]
M7-int-water-neg-port
[port]
M7stream-2
M7-pump-neg-stream-A
M7-adj-fluidA-neg-stream-A
M7-stream-1-inlet
M7-stream-3-outlet
M7-edge-1B
M7-neg-edge-1B
M7-edge-2A
M7-neg-edge-2A
M7-pump-speed
[rotational-speed]
M7-fluid-temperature
[temperature]
M7-fluid-density
[density]
M7-inlet-pressure [prssure]
M7-inlet-flow-rate
[mass-flow-rate]
M7-NPSH [head]
M7-inlet-surface-normal
[direction-vector]
M7-outlet-surface-normal
[direction-vector]
Not Shown: 6-member negative set similar
to one above, attached to M5-port-fluid-outlet
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Part link
5 Describes link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set

C.6.2 Statistics

STATISTICS FOR HYPOTHETICAL MODEL AND DESCRIPTION	
model size:	0 K
model internal method:	none
model author:	none
model description size:	34 K
individuals in model description:	4
model map representation:	39 objects, 85 links

C.6.3 The Implemented Model Description

THE HYPOTHETICAL MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m7-inlet-flow-rate m7-inlet-pressure m7-fluid-temperature m7-pump-speed	value value value value
Output Variables	m7-npsh	value
Causal Variables	m7-inlet-flow-rate m7-fluid-temperature m7-pump-speed	value value value
Affected Variables	m7-npsh	value
Carried Variables	nil	
Call Form	nil	
Return Form	nil	
Enabling Conditions	nil	
Approximation Conditions	nil	

THE HYDRAULIC HORSEPOWER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
7. **any-common-portion?** (function name)	m6-suction-pressure	time, time-type, time-interval-type time-precision, time-precision-type
	m6-suction-nozzle-id	time, time-type, time-interval-type time-precision, time-precision-type
8. **any-common-portion?**	m6-suction-pressure	time, time-type, time-interval-type time-precision, time-precision-type
	m6-discharge-nozzle-id	time, time-type, time-interval-type time-precision, time-precision-type
9. **any-common-portion?**	m6-suction-pressure	time, time-type, time-interval-type time-precision, time-precision-type
	m6-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
10. **any-common-portion?**	m6-suction-pressure	time, time-type, time-interval-type time-precision, time-precision-type
	m6-specific-gravity	time, time-type, time-interval-type time-precision, time-precision-type
11. >	m6-discharge-pressure	value
	m6-suction-pressure	value

C.6 The Hypothetical Model Description

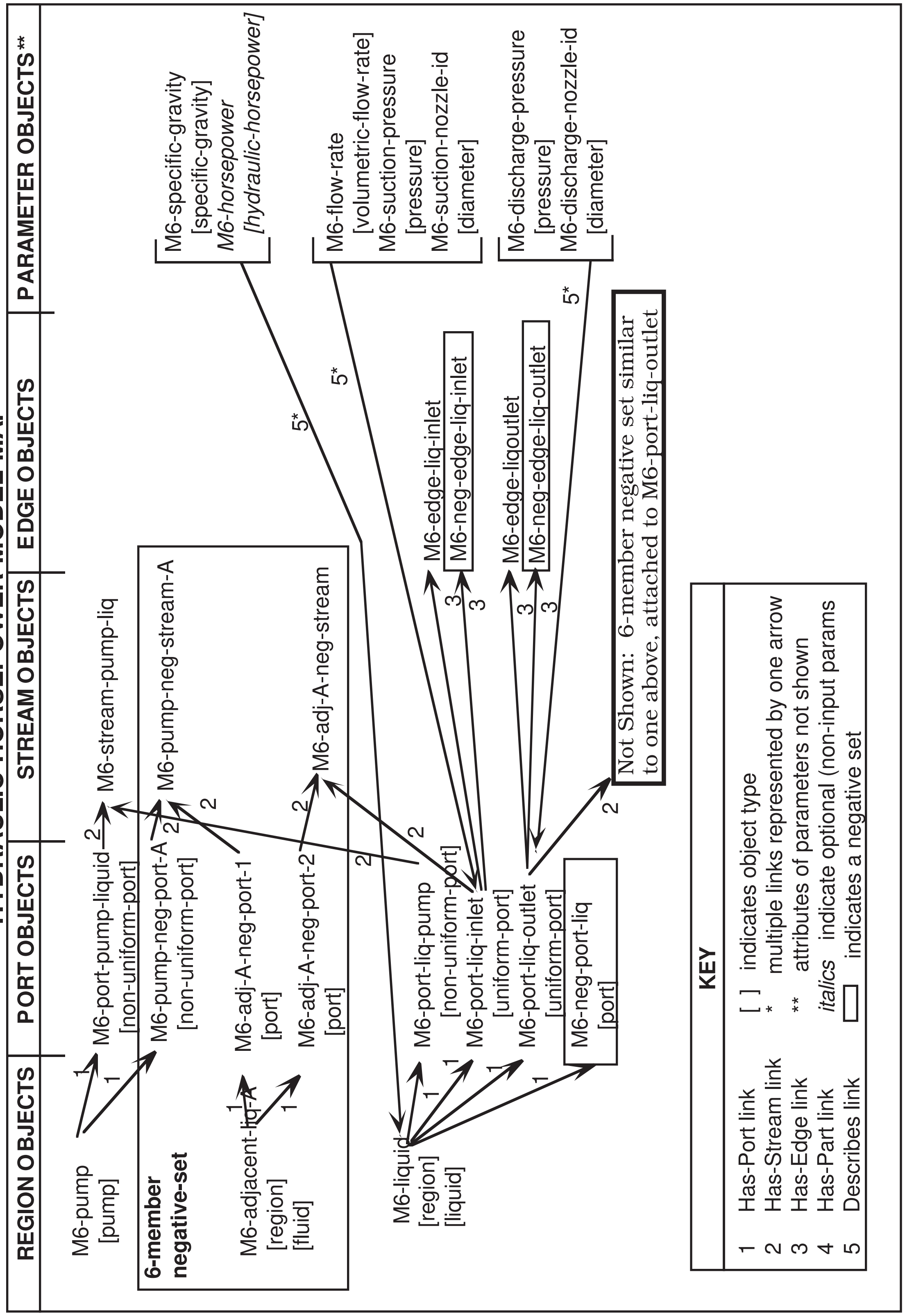
C.6.1 The Model and its Requirements

The Hypothetical model is simply a model description that does not correspond to any real model. It was built by modifying the model description for the NPSH model to test reconfiguration capabilities that did not get tested by the real pump models. The purpose of the Hypothetical model is to force the matching system to reconfigure parts of the pump at two different levels of detail within the same matching. The Hypothetical model requires a pump individual that has a `has-part` relation with a pump casing. In the equipment description used to match the pump models, subparts of the casing and the pump are represented, but neither the pump nor the casing appear as individuals. To find the match for the Hypothetical model in this equipment description, the reconfiguration algorithms do two separate reconfigurations to generate the pump and the casing individuals.

Since the Hypothetical model is not a real model, its description does not contain any *approximation conditions, enabling conditions, call-form,* or *return-form*. It does have *input, output, causal,* and *affected variables*, (taken from the NPSH model description) so that the model may be identified as a potential model to match when given a parameter and a goal.

THE HYDRAULIC HORSEPOWER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Carried Variables (continued)	From: m6-flow-rate To: m6-horsepower	time-units time-units
	From: m6-flow-rate To: m6-horsepower	time-interval-type time-interval-type
	From: m6-flow-rate To: m6-horsepower	time-precision-type time-precision-type
	From: m6-flow-rate To: m6-horsepower	time-precision time-precision
Call Form **m6-hydraulic-hp:: calc-hydraulic-horsepower** (function name)	m6-suction-pressure m6-discharge-pressure m6-suction-nozzle-id m6-discharge-nozzle-id m6-flow-rate m6-specific-gravity	value value value value value value
Return Form	m6-horsepower	value
Enabling Conditions	nil	
Approximation Conditions		
1. **equal** (function name)	m6-suction-pressure m6-discharge-pressure	time-units time-units
2. **equal**	m6-suction-pressure m6-suction-nozzle-id	time-units time-units
3. **equal**	m6-suction-pressure m6-discharge-nozzle-id	time-units time-units
4. **equal**	m6-suction-pressure m6-flow-rate	time-units time-units
5. **equal**	m6-suction-pressure m6-specific-gravity	time-units time-units
6. **any-common-portion?**	m6-suction-pressure m6-discharge-pressure	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type

HYDRAULIC HORSEPOWER MODEL MAP



requirement is specified through negative sets similarly to the NPSH and Wear Ring model maps. Two negative sets of 6 objects each specify that anything beyond the inlet and outlet surface of the liquid must be outside the pump. (See the Hydraulic Horsepower Model Map in Section C.5.3.) A singleton negative set specifies that the liquid region can have no other ports and thus nothing can "hide" inside the liquid itself. The only conditions on parameter values are that the values of all the parameters occur at the same time (*Approximation Conditions 1–10*) and that the outlet pressure be greater than the inlet pressure (*Approximation Condtion 11*).

C.5.2 Statistics

STATISTICS FOR MODEL AND DESCRIPTION	
model size:	2 K
model internal method:	mathematical equations
model author:	J. Murdock
model description size:	32 K
individuals in model description:	2
model map representation:	31 objects, 71 links

C.5.3 The Implemented Model Description

THE HYDRAULIC HORSEPOWER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m6-suction-pressure m6-discharge-pressure m6-suction-nozzle-id m6-discharge-nozzle-id m6-flow-rate m6-specific-gravity	value value value value value value
Output Variables	m6-horsepower	value
Causal Variables	m6-suction-pressure m6-suction-nozzle-id m6-discharge-nozzle-id m6-specific-gravity m6-horsepower	value value value value value
Affected Variables	m6-discharge-pressure m6-flow-rate	value value
Carried Variables (continued)	From: m6-flow-rate To: m6-horsepower From: m6-flow-rate To: m6-horsepower	time time time-type time-type

THE WEAR RING MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
12. **any-common-portion?** (function name)	m5-power-2	time, time-type, time-interval-type time-precision, time-precision-type
	m5-mu-2	time, time-type, time-interval-type time-precision, time-precision-type
13. **any-common-portion?**	m5-power-2	time, time-type, time-interval-type time-precision, time-precision-type
	m5-rho-2	time, time-type, time-interval-type time-precision, time-precision-type
14. **any-common-portion?**	m5-power-2	time, time-type, time-interval-type time-precision, time-precision-type
	m5-rpm-2	time, time-type, time-interval-type time-precision, time-precision-type
15. **equal**	m5-power-1	value-units
	m5-power-2	value-units
16. **equal**	m5-i-wear-ring-od	value-units
	m5-c-wear-ring-id	value-units
17. **m5-wear-ring:: power-delta-small?**	m5-power-2	value
	m5-power-1	value
18. >	m5-c-wear-ring-id	value
	m5-i-wear-ring-od	value
19. >	m5-power-2	value
	m5-power-1	value

C.5 The Hyraulic Horsepower Model Description

C.5.1 The Model and its Requirements

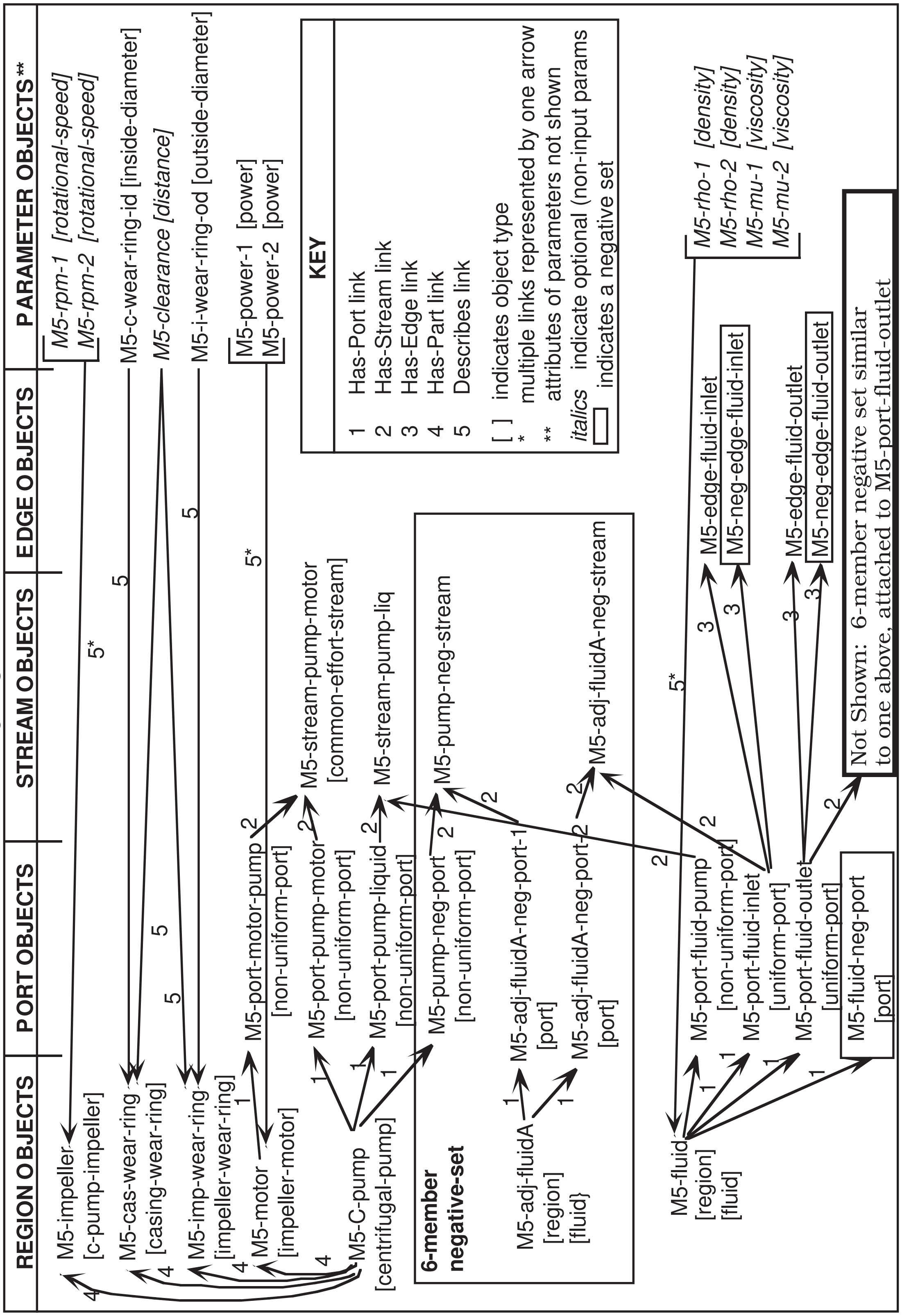
The Hydraulic Horsepower model calculates the theoretical minimum horsepower required by a pump to generate the pressure and velocity of a fluid at its outlet from the fluid conditions at the pump inlet [55]. The model takes the inlet and outlet pressures, the inlet and outlet nozzle diameters, the volumetric flow rate, and the specific gravity of the fluid as input. This model, unlike the NPSH and Wear Ring models, applies to any kind of pump, not just centrifugal pumps.

The Hydraulic Horsepower model does require that the liquid region inside the pump occupy the entire pump, because the parameters are assumed to describe certain portions of that space. The specific gravity is assumed to describe the whole region. The inlet and outlet pressures are assumed to describe a surface of the liquid exactly the size of the inlet and outlet nozzles. This

THE WEAR RING MODEL DESCRIPTION

	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
4. **any-common-portion?** (function name)	m5-mu-1	value, value-type, value-interval-type, value-precision, value-precision-type
	m5-mu-2	value, value-type, value-interval-type, value-precision, value-precision-type
5. **any-common-portion?**	m5-rho-1	value, value-type, value-interval-type, value-precision, value-precision-type
	m5-rho-2	value, value-type, value-interval-type, value-precision, value-precision-type
6. **any-common-portion?**	m5-rpm-1	value, value-type, value-interval-type, value-precision, value-precision-type
	m5-rpm-2	value, value-type, value-interval-type, value-precision, value-precision-type
7. **any-common-portion?**	m5-power-1	time, time-type, time-interval-type time-precision, time-precision-type
	m5-i-wear-ring-od	time, time-type, time-interval-type time-precision, time-precision-type
8. **any-common-portion?**	m5-power-1	time, time-type, time-interval-type time-precision, time-precision-type
	m5-c-wear-ring-id	time, time-type, time-interval-type time-precision, time-precision-type
9. **any-common-portion?**	m5-power-1	time, time-type, time-interval-type time-precision, time-precision-type
	m5-mu-1	time, time-type, time-interval-type time-precision, time-precision-type
10. **any-common-portion?**	m5-power-1	time, time-type, time-interval-type time-precision, time-precision-type
	m5-rho--1	time, time-type, time-interval-type time-precision, time-precision-type
11. **any-common-portion?**	m5-power-1	time, time-type, time-interval-type time-precision, time-precision-type
	m5-rpm-1	time, time-type, time-interval-type time-precision, time-precision-type

THE WEAR RING MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Carried Variables	From: m5-power-2 To: m5-clearance	time time
	From: m5-power-2 To: m5-clearance	time-type time-type
	From: m5-power-2 To: m5-clearance	time-interval-type time-interval-type
	From: m5-power-2 To: m5-clearance	time-units time-units
	From: m5-power-2 To: m5-clearance	time-precision-type time-precision-type
	From: m5-power-2 To: m5-clearance	time-precision time-precision
	From: m5-power-2 To: m5-clearance	time-dimension time-dimension
	From: m5-c-wear-ring-id To: m5-clearance	value-units value-units
Call Form **m5-wear-ring:: wear-ring-power-consumption** (function name)	m5-power-1 m5-power-2 m5-c-wear-ring-id m5-i-wear-ring-od	value value value value
Return Form	m5-clearance	value
Enabling Conditions	NIL	
Approximation Conditions		
1. **equal** (function name)	m5-rho-1 m5-rho-2	value-units value-units
2. **equal**	m5-mu-1 m5-mu-2	value-units value-units
3. **equal**	m5-rpm-1 m5-rpm-2	value-units value-units


WEAR RING MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
PARAMETER OBJECTS**
M5-impeller
[c-pump-impeller]
M5-cas-wear-ring
[casing-wear-ring]
M5-imp-wear-ring
[impeller-wear-ring]
M5-motor
[impeller-motor]
M5-C-pump
[centrifugal-pump]
M5-port-motor-pump
[non-uniform-port]
M5-port-pump-motor
[non-uniform-port]
M5-port-pump-liquid
[non-uniform-port]
M5-stream-pump-motor
[common-effort-stream]
M5-stream-pump-liq
M5-rpm-1 [rotational-speed]
M5-rpm-2 [rotational-speed]
M5-c-wear-ring-id [inside-diameter]
M5-clearance [distance]
M5-i-wear-ring-od [outside-diameter]
M5-power-1 [power]
M5-power-2 [power]
6-member
negative-set
M5-pump-neg-port
[non-uniform-port]
M5-pump-neg-stream
M5-adj-fluidA
[region]
[fluid}
M5-adj-fluidA-neg-port-1
[port]
M5-adj-fluidA-neg-port-2
[port]
M5-adj-fluidA-neg-stream
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Part link
5 Describes link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set
M5-fluid
[region]
[fluid]
M5-port-fluid-pump
[non-uniform-port]
M5-port-fluid-inlet
[uniform-port]
M5-port-fluid-outlet
[uniform-port]
M5-fluid-neg-port
[port]
M5-edge-fluid-inlet
M5-neg-edge-fluid-inlet
M5-edge-fluid-outlet
M5-neg-edge-fluid-outlet
M5-rho-1 [density]
M5-rho-2 [density]
M5-mu-1 [viscosity]
M5-mu-2 [viscosity]
Not Shown: 6-member negative set similar
to one above, attached to M5-port-fluid-outlet

to allow fluid to move through the clearance. As these parts wear, the clearance between the rings increases. The increased clearance allows some fluid leakage and makes the pump less efficient, thereby requiring more power to produce the same flow. The Wear Ring Model is calculates the change in clearance given the change in power consumption observed.

The Wear Ring Model Map specifies physical situation for this model. The model requires a centrifugal pump with an impeller wear ring and a casing wear ring. The pump must be completely full of the fluid being pumped. We specify this condition using a singleton negative set and two 6-member negative sets similar to those used in the NPSH model map. The result of the model is valid only if the viscosity, density, and pump speed are the same for the initial observation and the observation of increased power. *Conditions* 1–6 in the *Approximation Conditions* guarantee that those values are constant. *Conditions* 7–14 specify the time relationships amongst the model parameters. The model does not require any particular units for power, but does require the initial power and final power to have the same units (*Condition* 15), and similarly that the two wear ring diamaters given as input both have the same units (*Condition* 16). The model is only valid if the change in power consumption is less than 9%, which we specify in *Condition* 17.

C.4.2 Statistics

STATISTICS FOR WEAR RING MODEL AND DESCRIPTION	
model size:	2 K
model internal method:	mathematical equation
model author:	J. Murdock
model description size:	42 K
individuals in model description:	6
model map representation:	32 objects, 93 links

C.4.3 The Implemented Model Description

THE WEAR RING MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m5-i-wear-ring-od m5-c-wear-ring-id m5-power-1 m5-power-2	value value value value
Output Variables	m5-clearance	value
Causal Variables	m5-i-wear-ring-od m5-c-wear-ring-id m5-power-1 m5-clearance	value value value value
Affected Variables	m5-power-2	value

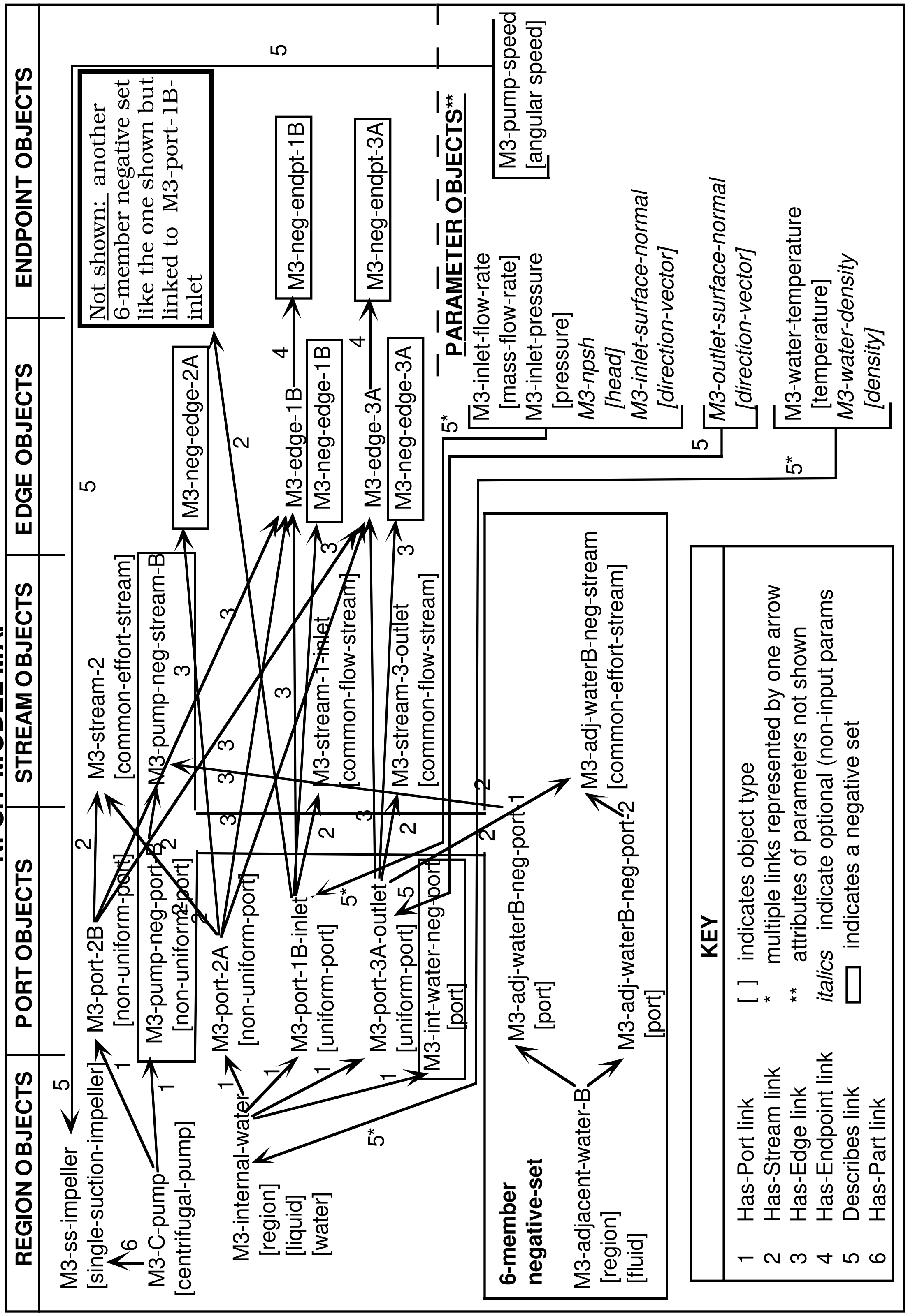
THE NPSH MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
14. **m3-hw-pump:: time-type-real-or-integer?** (function name)	m3-water-temperature	time-type
15. **m3-hw-pump:: time-type-real-or-integer?**	m3-pump-speed	time-type
16. **m3-hw-pump:: time-type-real-or-integer?**	m3-water-density	time-type
17. **m3-hw-pump:: time-type-real-or-integer?**	m3-npsh	time-type
18. **any-common-portion?**	m3-inlet-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
	m3-npsh	time, time-type, time-interval-type time-precision, time-precision-type
19. **any-common-portion?**	m3-inlet-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
	m3-water-density	time, time-type, time-interval-type time-precision, time-precision-type
20. **any-common-portion?**	m3-inlet-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
	m3-pump-speed	time, time-type, time-interval-type time-precision, time-precision-type
21. **any-common-portion?**	m3-inlet-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
	m3-water-temperature	time, time-type, time-interval-type time-precision, time-precision-type
22. **any-common-portion?**	m3-inlet-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type
	m3-inlet-pressure	time, time-type, time-interval-type time-precision, time-precision-type

C.4 The Wear Ring Model Description

C.4.1 The Model and its Requirements

The wear ring model is a correlation of increased pump power consumption with the wear of internal parts [22]. Centrifugal pumps have impellers that turn inside casings. The impeller is fitted with a wear ring, as is the casing, at the point where the impeller and casing are in very close proximity. The two rings should have enough clearance to allow the impeller and its wear ring to turn but not

THE NPSH MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Enabling Conditions	NIL	
Approximation Conditions		
1. < (function name)	m3-water-temperature nil	value 400
2. >	m3-inlet-flow-rate nil	value 0
3. **m3-hw-pump:: pump-rpm-ranges-satisfied?**	m3-pump-speed	value
4. **m3-hw-pump:: pump-gpm-ranges-satisfied?**	m3-inlet-flow-rate	value
5. **m3-hw-pump:: vector-values-dont-overlap?**	m3-inlet-surface-normal nil	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type (0, 0, 0), real, point, 3, 0, absolute
6. **m3-hw-pump:: vector-values-dont-overlap?**	m3-outlet-surface-norma nil]	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type (0, 0, 0), real, point, 3, 0, absolute
7. **equal**	m3-inlet-flow-rate m3-inlet-pressure	time-units time-units
8. **equal**	m3-inlet-flow-rate m3-water-temperature	time-units time-units
9. **equal**	m3-inlet-flow-rate m3-pump-speed	time-units time-units
10. **equal**	m3-inlet-flow-rate m3-water-density	time-units time-units
11. **equal**	m3-inlet-flow-rate m3-npsh	time-units time-units
12. **m3-hw-pump:: time-type-real-or-integer?**	m3-inlet-flow-rate	time-type
13. **m3-hw-pump:: time-type-real-or-integer?**	m3-inlet-pressure	time-type


NPSH MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
ENDPOINT OBJECTS
M3-ss-impeller
[single-suction-impeller]
M3-C-pump
[centrifugal-pump]
M3-internal-water
[region]
[liquid]
[water]
M3-port-2B
[non-uniform-port]
M3-pump-neg-port-B
[non-uniform-port]
M3-port-2A
[non-uniform-port]
M3-port-1B-inlet
[uniform-port]
M3-port-3A-outlet
[uniform-port]
M3-int-water-neg-port
[port]
M3-stream-2
[common-effort-stream]
M3-pump-neg-stream-B
M3-stream-1-inlet
[common-flow-stream]
M3-stream-3-outlet
[common-flow-stream]
M3-neg-edge-2A
M3-edge-1B
M3-neg-edge-1B
M3-edge-3A
M3-neg-edge-3A
Not shown: another 6-member negative set like the one shown but linked to M3-port-1B-inlet
M3-neg-endpt-1B
M3-neg-endpt-3A
PARAMETER OBJECTS**
M3-pump-speed
[angular speed]
M3-inlet-flow-rate
[mass-flow-rate]
M3-inlet-pressure
[pressure]
M3-npsh
[head]
M3-inlet-surface-normal
[direction-vector]
M3-outlet-surface-normal
[direction-vector]
M3-water-temperature
[temperature]
M3-water-density
[density]
6-member negative-set
M3-adjacent-water-B
[region]
[fluid]
M3-adj-waterB-neg-port
[port]
M3-adj-waterB-neg-port-1
M3-adj-waterB-neg-port-2
M3-adj-waterB-neg-stream
[common-effort-stream]
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Endpoint link
5 Describes link
6 Has-Part link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set

C.3.2 Statistics

STATISTICS FOR NPSH MODEL AND DESCRIPTION	
model size:	7 K
model internal method:	interpolation on data tables
model author:	J. Murdock
model description size:	36 K
individuals in model description:	3
model map representation:	38 objects, 84 links

C.3.3 The Implemented Model Description

THE NPSH MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m3-inlet-flow-rate m3-inlet-pressure m3-water-temperature m3-pump-speed	value value value value
Output Variables	m3-npsh	value
Causal Variables	m3-inlet-flow-rate m3-water-temperature m3-pump-speed	value value value
Affected Variables	m3-npsh	value
Carried Variables	From: m3-inlet-pressure To: m3-npsh	time time
	From: m3-inlet-pressure To: m3-npsh	time-interval-type time-interval-type
	From: m3-inlet-pressure To: m3-npsh	time-precision-type time-precision-type
	From: m3-inlet-pressure To: m3-npsh	time-precision time-precision
	From: m3-inlet-pressure To: m3-npsh	time-type time-type
	From: m3-inlet-pressure To: m3-npsh	time-units time-units
Call Form **m3-nw-pump-::** **pump-npsh** (function name)	m3-pump-speed m3-inlet-flow-rate m3-inlet-pressure m3-water-temperature	value value value value
Return Form	m3-npsh	value

THE DITTUS-BOELTER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
35. **vectors-parallel?**	m2-pipe-axial-direction	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type
	m2-surfoace-normal-1b	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type

C.3 The NPSH Model Description

C.3.1 The Model and its Requirements

The NPSH model is a computer program that calculates the minimum net positive suction head (NPSH) required at the inlet of a single stage centrifugal pump for operation without cavitation. The NPSH can be thought of approximately as the pressure (measured in feet of liquid) at the entrance to the pump. The general principles involved in calculating head are discussed in many textbooks and pump handbooks, such as [6] and [22]. The NPSH program implements some plots of correlated data found in [55]. This model requires a single stage centrifugal pump that is pumping hot water. The pump must have a single-suction impeller. The model map thus specifies three regions, a water region, a single stage pump, and an impeller that is part of that pump. The water region must be filling the pump completely. The model is not applicable if some region inside the pump is occupied by gas. The singleton negative set containing `M3-int-water-neg-port` (shown in the model map in this section) guarantees that the water region has no other adjacent regions, so that, for example, no regions surrounded by the water can exist. The two negative sets, each having 6 members, ensure that the water region matched is the whole region inside the pump by specifying that there can be nothing adjacent to the water region that also shares a surface with the pump.

The model calculates NPSH from the inlet flowrate, pressure, the water temperature and the impeller speed. The locations and spatial extents of these parameters are specified in the model map. The model requires that these parameters lie within specific ranges which we specify in the *approximation conditions*. *Conditions* 1–4 ensure that the water temperature, the flowrate, and the impeller speed are within the ranges covered by this model. *Conditions* 5 and 6 specify that the surfaces of the water at the inlet and outlet must be flat by requiring them to have non-zero surface normal vectors describing the whole inlet or outlet surface. *Conditions* 7–22 ensure that the all the parameters have the appropriate time relationships to each other.

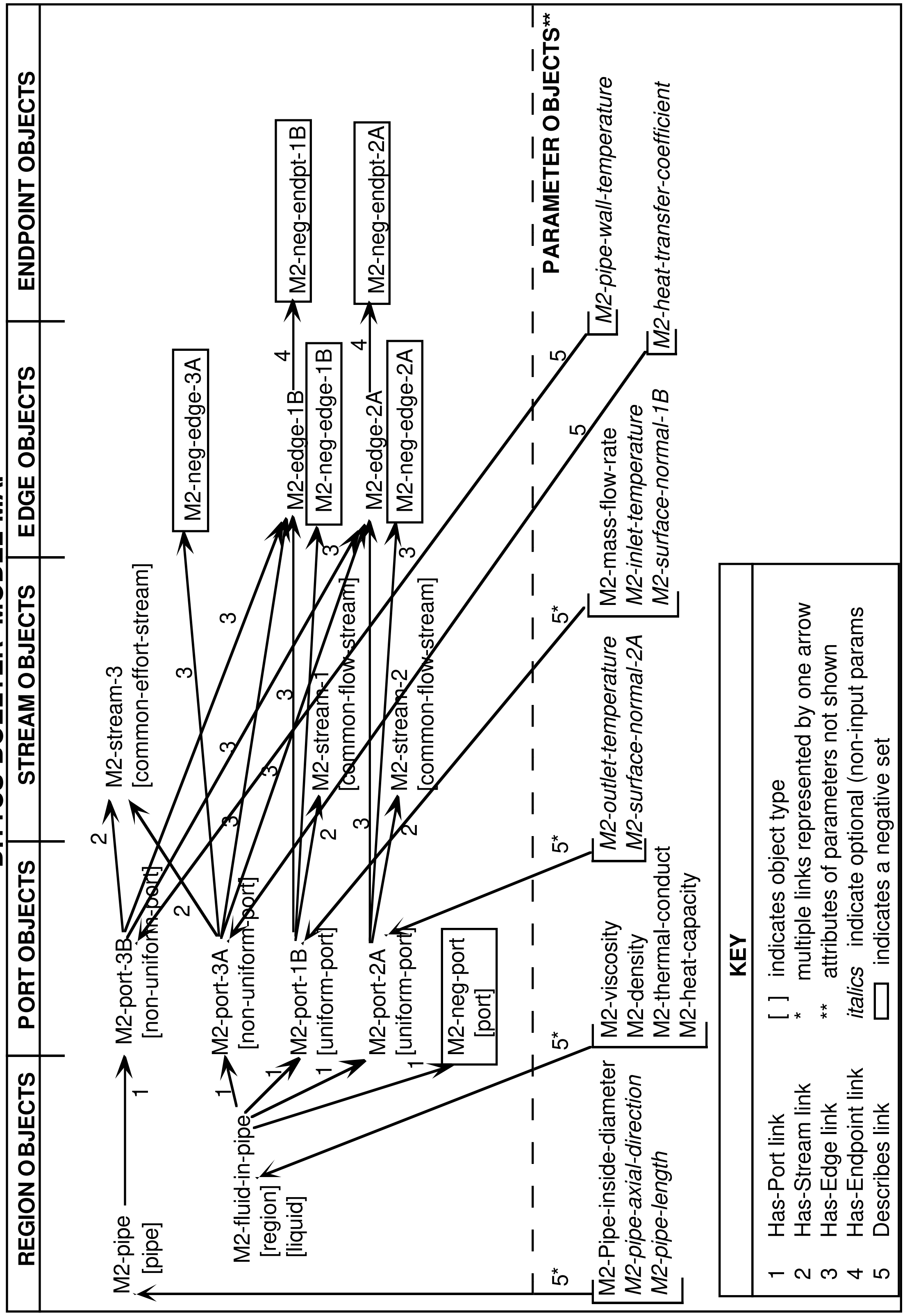
THE DITTUS-BOELTER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
26. **any-common-portion?** (function name)	m2-viscosity m2-thermal-conductivity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
27. **any-common-portion?**	m2-viscosity m2-heat-capacity	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
28. **any-common-portion?**	m2-viscosity m2-density	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
29. **m2-dittus-boelter:: re-in-range?**	m2-pipe-inside-diameter m2-mass-flow-rate m2-density m2-viscosity	value value value value
30. **m2-dittus-boelter:: pr-in-range?**	m2-heat-capacity m2-viscosity m2-thermal-conductivity	value value value
31. **m2-dittus-boelter:: l-over-d-ok?**	m2-pipe-length m2-pipe-inside-diameter	value value
32. **m2-dittus-boelter:: pipe-temperatures-ok?**	m2-inlet-temperature m2-outlet-temperature m2-pipe-wall-temperature	value value value
33. **>**	m2-mass-flow-rate nil	value 0
34. **vectors-parallel?**	m2-pipe-axial-direction m2-surface-normal-2a	value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type value, value-type, value-interval-type, value-dimension, value-precision, value-precision-type, time, time-type, time-interval-type, time-dimension, time-precision, time-precision-type

THE DITTUS-BOELTER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
15. **equal**	m2-viscosity m2-density	time-type time-type
16. **equal**	m2-viscosity m2-heat-capacity	time-type time-type
17. **equal**	m2-viscosity m2-thermal-conductivity	time-type time-type
18. **equal**	m2-viscosity m2-inlet-temperature	time-type time-type
19. **equal**	m2-viscosity m2-outlet-temperature	time-type time-type
20. **equal**	m2-viscosity m2-pipe-wall-temperature	time-type tme-type
21 **equal** (function name)	m2-viscosity m2-mass-flow-rate	time-type time-type
22. **any-common-portion?**	m2-viscosity m2-mass-flow-rate	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
23. **any-common-portion?**	m2-viscosity m2-pipe-wall-temperature	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
24. **any-common-portion?**	m2-viscosity m2-outlet-temperature	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
25. **any-common-portion?**	m2-viscosity m2-inlet-temperature	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type

THE DITTUS-BOELTER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions		
1. **equal** (function name)	m2-viscosity m2-density	time-units time-units
2. **equal**	m2-viscosity m2-heat-capacity	time-units time-units
3. **equal**	m2-viscosity m2-thermal-conductivity	time-units time-units
4. **equal**	m2-viscosity m2-inlet-temperature	time-units time-units
5. **equal**	m2-viscosity m2-outlet-temperature	time-units time-units
6. **equal**	m2-viscosity m2-pipe-wall-temperature	time-units time-units
7. **equal**	m2-viscosity m2-mass-flow-rate	time-units time-units
8. **equal**	m2-viscosity m2-density	time-dimension time-dimension
9. **equal**	m2-viscosity m2-heat-capacity	time-dimension time-dimension
10. **equal**	m2-viscosity m2-thermal-conductivity	time-dimension time-dimension
11. **equal**	m2-viscosity m2-inlet-temperature	time-dimension time-dimension
12. **equal**	m2-viscosity m2-outlet-temperature	time-dimension time-dimension
13. **equal**	m2-viscosity m2-pipe-wall-temperature	time-dimension time-dimension
14. **equal**	m2-viscosity m2-mass-flow-rate	time-dimension time-dimension

THE DITTUS-BOELTER MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m2-pipe-inside-diameter m2-mass-flow-rate m2-density m2-viscosity m2-thermal-conductivity m2-heat-capacity	value value value value value value
Output Variables	m2-heat-transfer-coefficient	value
Causal Variables	m2-pipe-inside-diameter m2-mass-flow-rate m2-density m2-viscosity m2-thermal-conductivity m2-heat-capacity	value value value value value value
Affected Variables	m2-heat-transfer-coefficient	value
Carried Variables	From: m2-mass-flow-rate To: m2-heat-transfer-coefficient From: m2-mass-flow-rate To: m2-heat-transfer-coefficient From: m2-mass-flow-rate To: m2-heat-transfer-coefficient From: m2-mass-flow-rate To: m2-heat-transfer-coefficient From: m2-mass-flow-rate To: m2-heat-transfer-coefficient From: m2-mass-flow-rate To: m2-heat-transfer-coefficient	time time time-type time-type time-interval-type time-interval-type time-precision-type time-precision-type time-precision time-precision time-units time-units
Call Form **m2-dittus-boelter::** **dittus-boelter** (function name)	m2-pipe-inside-diameter m2-mass-flow-rate m2-density m2-viscosity m2-thermal-conductivity m2-heat-capacity	value value value value value value
Return Form	m2-heat-transfer-coefficient	value
Enabling Conditions	NIL	

C.2.3 The Implemented Model Description



does not extend beyond the end of a single pipe into an adjacent pipe. The ends of the cylinder are conceptual, not physical. The fluid extends beyond the ends of the cylinder in either direction. The *model map*, shown in Section C.2.3, specifies these physical adjacencies and guarantees through its negative ports, edges, and endpoints that the cylinder fills the pipe without crossing pipe boundaries. The negative set { `M2-neg-port` } allows the cylinder to have only the 3 positive ports specified, thus preventing the fluid from being adjacent to more than one pipe and also ensuring that nothing else is "hiding" inside the cylinder. The cylinder's surface with the pipe is guaranteed to be a cylindrical shell (rather than some other shape) by the 5 singleton negative sets. These negative edges and endpoints specify that none of the ports can have any other edges than the two edges forming the boundaries at each end, and that those edges must be closed curves (because they have no endpoints).

Each of the parameters describe a particular portion of the cylinder or pipe as shown in Figure C.2. For example, the viscosity describes the entire volume of the cylinder while the heat transfer coeficient describes only the surface of that cylider. The *model map* also makes explicit the spatial requirements for parameters. We specify requirements on the parameter's attribute values in the *approximation conditions*. All the parameters must have values over the same time period as specified in *conditions* 1 through 28. *Conditions* 29 and 30 guarantee that the Reynolds number and Prandtl number are in acceptable ranges. *Condition* 31 guarantees that the aspect ratio of the pipe ($length/diameter$) required by the model is met. *Condition* 32 checks that the average bulk temperature in the cylindrical section of fluid, as calculated from the inlet and outlet temperatures does not vary more than 10 degrees F. from the wall temperature. This condition also guarantees that the liquid is being cooled as required by the exponent, $a = 0.3$, in the implemented Dittus-Boelter computer program. The final two conditions check that the surfaces of the cylinder have normal directions parallel to the axial direction of the pipe, guaranteeing that the cylindrical region matched will have the flat parallel ends required.

C.2.2 Statistics

STATISTICS FOR DITTUS-BOELTER MODEL AND DESCRIPTION	
model size:	3 K
model internal method:	mathematical equation
model author:	J. Murdock
model description size:	54 K
individuals in model description:	2
model map representation:	34 objects, 67 links

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
51. **equal** (function name)	m8-hottemp-before m8-hottemp-after	value value
52. **m8-dalle-molle:: flow-proportional-to-height?**	m8-outflow-before m8-outflow-after m8-height-before m8-height-after	value value, value-precision-type value value, value-precision-type

C.2 The Dittus-Boelter Model and Description

C.2.1 The Model and its Requirements

The Dittus-Boelter Model is an empirical equation that correlates the heat transfer coefficient inside a pipe with properties of the fluid flowing through the pipe. The Dittus-Boelter equation,

$$h = 0.023 \frac{k}{D} (\frac{Dv\rho}{\mu})^{0.8} (\frac{C_p \mu}{k})^{a},$$

is found in many engineering textbooks on heat transfer, including [33,56]. The variables used are defined in Figure C.2, except for v, which is fluid velocity, and a, which is a constant 0.4 when the fluid is being heated and 0.3 when it is being cooled.

The model description built for the Dittus-Boelter model specifies the characteristics of the physical situations in which this equation is applicable. The model applies to any cylindrical section of the fluid as shown in Figure C.2, as long as cylinder is bounded by the pipe walls and

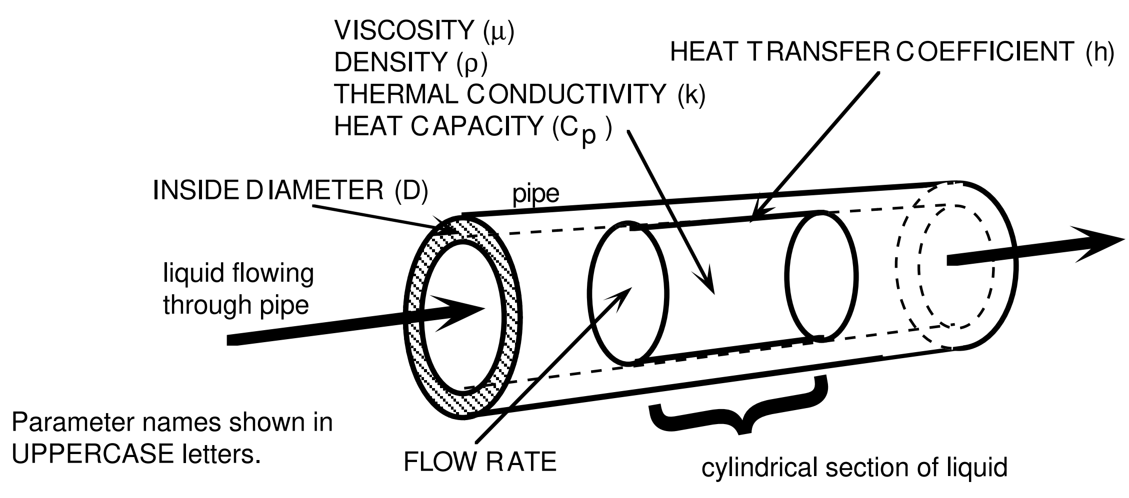


Figure C.2: The Dittus-Boelter model requires a cylindrical section of the fluid flowing inside a pipe as an explicit individual.

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
39. **m8-dalle-molle:: qspaces-have-common-element?**	m8-hottemp-before nil m8-tempout-before nil	value-precision-type 2 value-precision-type 4
40. **m8-dalle-molle:: qspaces-have-common-element?**	m8-coldtemp-before nil m8-tempout-before nil	value-precision-type 2 value-precision-type 2
41. **equal**	m8-cflow-deriv-before nil	value 0
42. **equal**	m8-cflow-deriv-after nil	value 0
43. **equal**	m8-height-deriv-before nil	value 0
44. **equal**	m8-hflow-deriv-before nil	value 0
45. **equal** (function name)	m8-hflow-deriv-after nil	value 0
46. **equal**	m8-ctemp-deriv-before nil	value 0
47. **equal**	m8-ctemp-deriv-after nil	value 0
48. **equal**	m8-htemp-deriv-before nil	value 0
49. **equal**	m8-htemp-deriv-after nil	value 0
50. **equal**	m8-coldtemp-before m8-coldtemp-after	value value

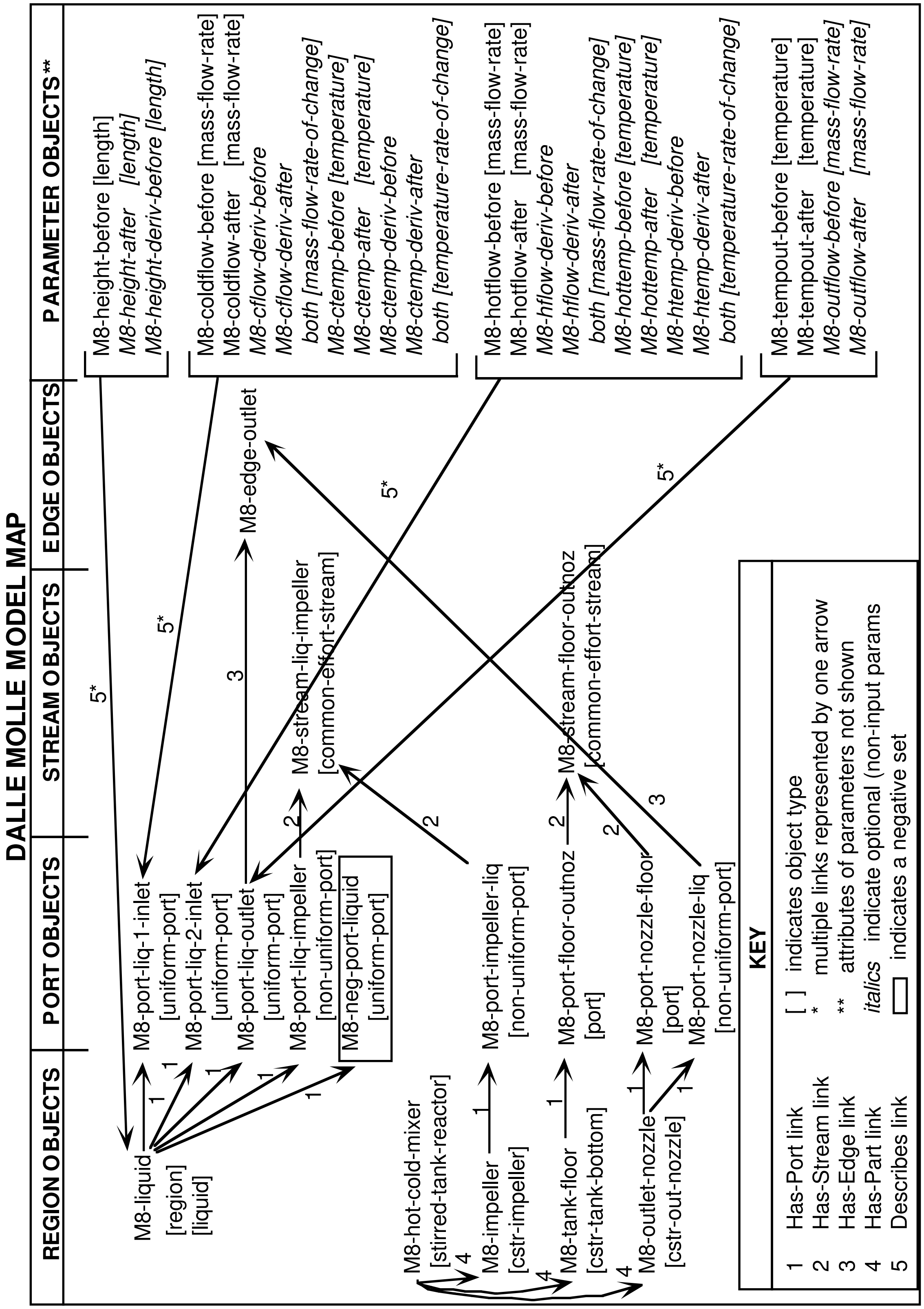
THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
29. **possibly-equal?**	m8-coldflow-after m8-cflow-deriv-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
30. **possibly-equal?**	m8-coldflow-after m8hflow-deriv-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
31. **possibly-equal?**	m8-coldflow-after m8-ctemp-deriv-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
32. **possibly-equal?**	m8-coldflow-after m8-htemp-deriv-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
33. **possibly-equal?** (function name)	m8-coldflow-after m8-outflow-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
34. **possibly-meets?**	m8-coldflow-before m8-coldflow-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
35. **m8-dalle-molle:: qspace-ok?**	m8-height-before nil	value-precision-type 3
36. **m8-dalle-molle:: qspace-ok?**	m8-coldflow-before nil	value-precision-type 5
37. **m8-dalle-molle:: qspace-ok?**	m8-hotflow-before nil	value-precision-type 5
38. **m8-dalle-molle:: qspace-ok?**	m8-tempout-before nil	value-precision-type 5

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
20. **possibly-equal?**	m8-coldflow-before m8-tempout-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
21. **possibly-equal?**	m8-coldflow-before m8-hotflow-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
22. **possibly-equal?**	m8-hotflow-before m8-cflow-deriv-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
23. **possibly-equal?**	m8-coldflow-before m8-height-deriv-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
24. **possibly-equal?** (function name)	m8-coldflow-before m8-hflow-deriv-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
25. **possibly-equal?**	m8-coldflow-before m8-ctemp-deriv-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
26. **possibly-equal?**	m8-coldflow-before m8-htemp-deriv-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
27. **possibly-equal?**	m8-coldflow-before m8-outflow-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
28. **possibly-equal?**	m8-coldflow-after m8-hotflow-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type

THE DALLE MOLLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Approximation Conditions--continued		
8. **equal**	m8-height-deriv-before m8-coldflow-before	time-units time-units
9. **equal**	m8-hflow-deriv-before m8-coldflow-before	time-units time-units
10. **equal**	m8-hflow-deriv-after m8-coldflow-before	time-units time-units
11. **equal**	m8-ctemp-deriv-before m8-coldflow-before	time-units time-units
12. **equal**	m8-ctemp-deriv-after m8-coldflow-before	time-units time-units
13. **equal**	m8-htemp-deriv-before m8-coldflow-before	time-units time-units
14. **equal** (function name)	m8-htemp-deriv-after m8-coldflow-before	time-units time-units
15. **equal**	m8-tempout-after m8-coldflow-before	time-units time-units
16. **equal**	m8-height-after m8-coldflow-before	time-units time-units
17. **possibly-before?**	m8-coldtemp-before m8-hottemp-before	value, value-type, value-interval-type, value-precision, value-precision-type value, value-type, value-interval-type, value-precision, value-precision-type
18. **possibly-before?**	m8-coldflow-before m8-coldflow-after	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type
19. **possibly-equal?**	m8-coldflow-before m8-height-before	time, time-type, time-interval-type time-precision, time-precision-type time, time-type, time-interval-type time-precision, time-precision-type

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Call Form **M8-dalle-molle:: simulate-mixing-tank** (function name)	m8-coldflow-before m8-coldflow-after m8-coldflow-before m8-hotflow-before m8-hotflow-after m8-hotflow-before m8-height-before m8-height-before m8-tempout-before m8-tempout-before	value value value-precision-type value value value-precision-type value value-precision-type value value-precision-type
Return Form	m8-height-after m8-height-after m8-tempout-after m8-tempout-after m8-tempout-after m8-tempout-after	value value-precision-type value value-precision-type time time-precision-type
Enabling Conditions	NIL	
Approximation Conditions		
1. **equal** (function name)	m8-height-before m8-coldflow-before	time-units time-units
2. **equal**	m8-tempout-before m8-coldflow-before	time-units time-units
3. **equal**	m8-coldflow-after m8-coldflow-before	time-units time-units
4. **equal**	m8-hotflow-before m8-coldflow-before	time-units time-units
5. **equal**	m8-hotflow-after m8-coldflow-before	time-units time-units
6. **equal**	m8-cflow-deriv-before m8-coldflow-before	time-units time-units
7. **equal**	m8-cflow-deriv-after m8-coldflow-before	time-units time-units

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Output Variables	m8-height-after	value
	m8-height-after	value-precision-type
	m8-height-after	time
	m8-height-after	time-precision-type
	m8-tempout-after	value
	m8-tempout-after	value-precision-type
	m8-tempout-after	time
	m8-tempout-after	time-precision-type
Causal Variables	m8-height-before	value
	m8-tempout-before	value
	m8-coldflow-before	value
	m8-coldflow-after	value
	m8-hotflow-before	value
	m8-hotflow-after	value
Affected Variables	m8-height-after	value
	m8-tempout-after	value
Carried Variables	From: m8-height-before	value-units
	To: m8-height-after	value-units
	From: m8-tempout-after	time
	To: m8-height-after	time
	From: m8-tempout-after	time-precision-type
	To: m8-height-after	time-precision-type
	From: m8-height-before	time-units
	To: m8-height-after	time-units
	From: m8-height-before	time-interval-type
	To: m8-height-after	time-interval-type
	From: m8-tempout-before	value-units
	To: m8-tempout-after	value-units
	From: m8-tempout-before	time-units
	To: m8-tempout-after	time-units
	From: m8-tempout-before	time-interval-type
	To: m8-tempout-after	time-interval-type


DALLE MOLLE MODEL MAP
REGION OBJECTS
PORT OBJECTS
STREAM OBJECTS
EDGE OBJECTS
PARAMETER OBJECTS**
M8-liquid [region] [liquid]
M8-hot-cold-mixer [stirred-tank-reactor]
M8-impeller [cstr-impeller]
M8-tank-floor [cstr-tank-bottom]
M8-outlet-nozzle [cstr-out-nozzle]
M8-port-liq-1-inlet [uniform-port]
M8-port-liq-2-inlet [uniform-port]
M8-port-liq-outlet [uniform-port]
M8-port-liq-impeller [non-uniform-port]
M8-neg-port-liquid [uniform-port]
M8-port-impeller-liq [non-uniform-port]
M8-port-floor-outnoz [port]
M8-port-nozzle-floor [port]
M8-port-nozzle-liq [non-uniform-port]
M8-stream-liq-impeller [common-effort-stream]
M8-stream-floor-outnoz [common-effort-stream]
M8-edge-outlet
M8-height-before [length]
M8-height-after [length]
M8-height-deriv-before [length]
M8-coldflow-before [mass-flow-rate]
M8-coldflow-after [mass-flow-rate]
M8-cflow-deriv-before
M8-cflow-deriv-after
both [mass-flow-rate-of-change]
M8-ctemp-before [temperature]
M8-ctemp-after [temperature]
M8-ctemp-deriv-before
M8-ctemp-deriv-after
both [temperature-rate-of-change]
M8-hotflow-before [mass-flow-rate]
M8-hotflow-after [mass-flow-rate]
M8-hflow-deriv-before
M8-hflow-deriv-after
both [mass-flow-rate-of-change]
M8-hottemp-before [temperature]
M8-hottemp-after [temperature]
M8-htemp-deriv-before
M8-htemp-deriv-after
both [temperature-rate-of-change]
M8-tempout-before [temperature]
M8-tempout-after [temperature]
M8-outflow-before [mass-flow-rate]
M8-outflow-after [mass-flow-rate]
1
2
3
4
5*
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
4 Has-Part link
5 Describes link
[ ] indicates object type
* multiple links represented by one arrow
** attributes of parameters not shown
italics indicate optional (non-input params
indicates a negative set

calculate what the HEIGHT and TEMPOUT will be when the system again reaches steady state.

The purpose of the model description is to make explicit the physical situations in which that model applies, and to make explicit the assumptions under which the model is valid. This model assumes that the CSTR has two liquid inlets, the outlet nozzle is at the bottom of the tank rather than out the side (so that flow out is proportional to height), and that enough liquid is present so that the impeller is immersed for good mixing. We have incorporated these assumptions into the spatial relations in the model map. This model also assumes that the CSTR is initially (prior to the step change in a flow rate) at steady state and returns to a steady state. We express these requirements as *Approximation Conditions* 41-49 in the model description, listed later in this appendix. The temperatures of the two inlet streams must be constant (only flow rates may change), which we express as *Conditions* 50 and 51. The flow out of the reactor must be proportional to the height of the liquid, and this is guaranteed by *Condition* 52. All of the parameters involved, including model inputs, outputs, and other parameters used for checking conditions must have the correct time relationships to each other. These are specified in *Approximation Conditions* 1-34. The Dalle Molle Model uses qualitative variables and takes as input the qualitative value of a variable along with the list of ordered qualitative values, called the *quantity space* from which the variable may take values. The Dalle Molle Model can work with quantity spaces only if they meet certain restrictions, and these are specified in *Conditions* 35-40.

C.1.2 Statistics

STATISTICS FOR DALLE MOLLE MODEL AND DESCRIPTION	
model size:	1210 K
model internal method:	qualitative simulation by QSIM
model author:	D. Dalle Molle, B. Kuipers, A. Farquhar, J. Rickel, D. Throop
model description size:	61 K
individuals in model description:	5
model map representation:	40 objects, 82 links

C.1.3 The Implemented Model Description

THE DALLE MOLLE MODEL DESCRIPTION		
	NAME OF OBJECT	ATTRIBUTE OF OBJECT
Input Variables	m8-tempout-before	value
	m8-tempout-before	value-precision-type
	m8-coldflow-before	value
	m8-coldflow-before	value-precision-type
	m8-coldflow-after	value
	m8-hotflow-before	value
	m8-hotflow-before	value-precision-type
	m8-hotflow-after	value

Appendix C

Implemented Model Descriptions

C.1 The Dalle Molle Model Description

C.1.1 The Model and its Requirements

David Dalle Molle developed this model and published it in his thesis [11]. The model uses QSIM, the qualitative simulation engine developed at the Univeristy of Texas at Austin [42]. The version of QSIM that we installed consists of about 1200 K of Common Lisp code which runs on a TI Explorer II lisp machine.

The model simulates some parameters of a continuous stirred-tank reactor (CSTR) like that shown in Figure C.1. The CSTR has two liquid streams flowing into it, a cold stream and a hot stream. The temperature of the liquid in the outlet stream will vary depending on the temperatures and flow rates of the two inlet streams. The height in the tank can also vary. The model takes as input the initial values of the HEIGHT (height of liquid in the tank) and TEMPOUT (temperature of the outlet stream) parameters as well as a step increase in either or both of the COLDFLOW or HOTFLOW parameters, the flowrates of the two inlet streams. It simulates the behavior to

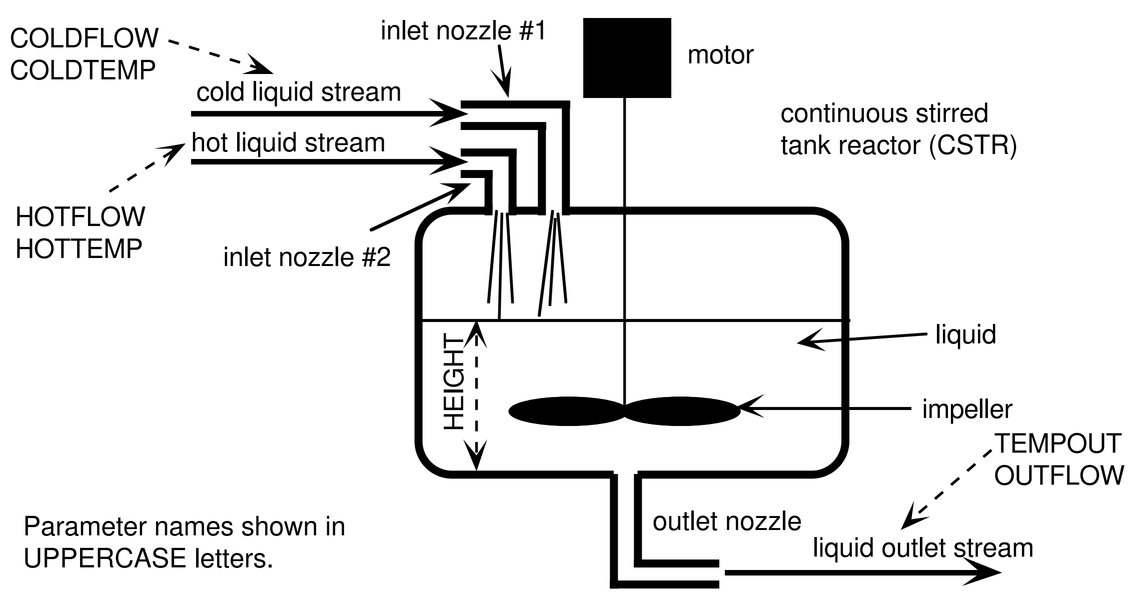
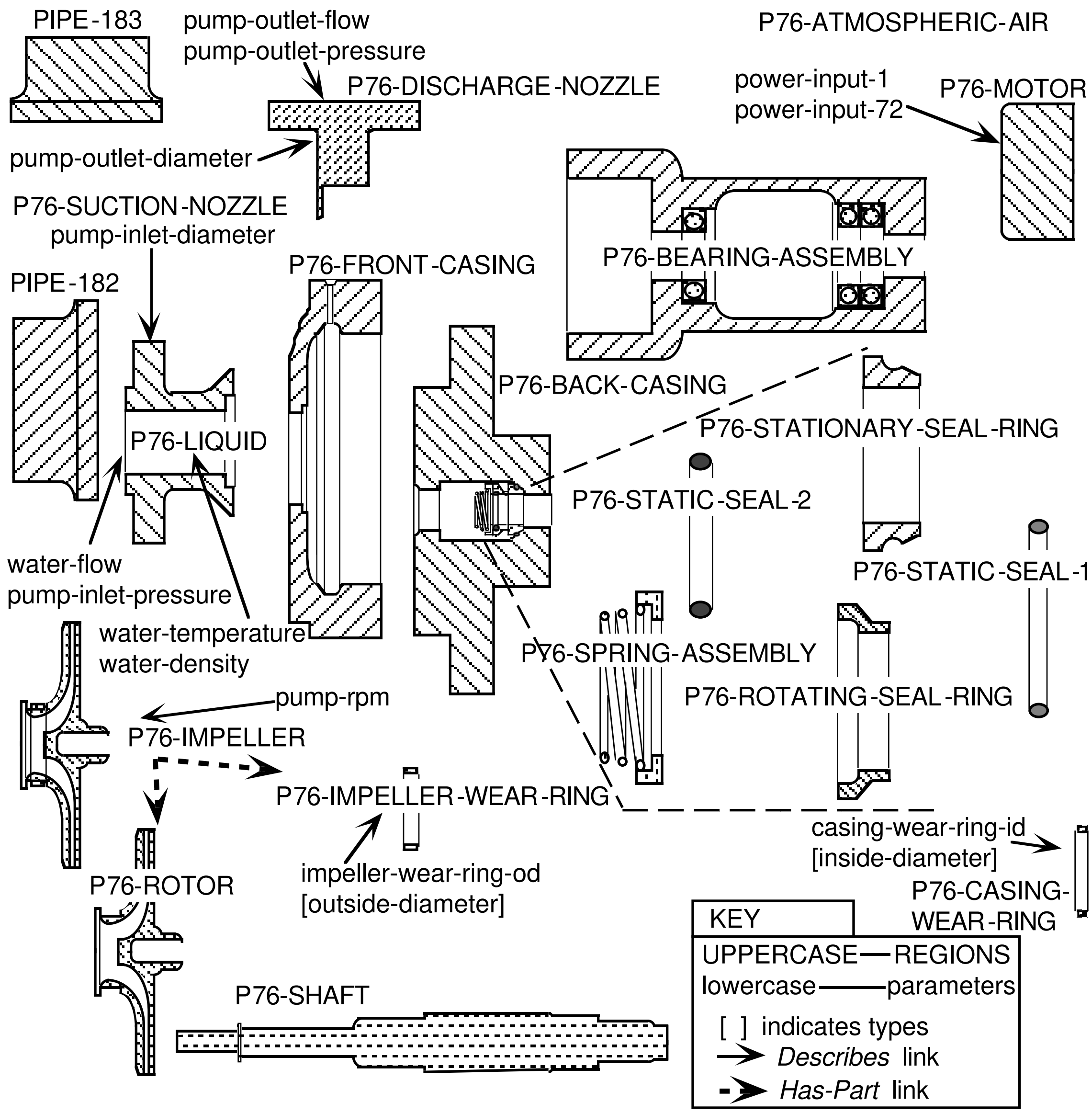


Figure C.1: The continuous stirred-tank reactor (CSTR) required by the Dalle Molle Model.


PIPE-183
pump-outlet-flow
pump-outlet-pressure
P76-ATMOSPHERIC-AIR
P76-DISCHARGE-NOZZLE
power-input-1
power-input-72
P76-MOTOR
pump-outlet-diameter
P76-SUCTION-NOZZLE
pump-inlet-diameter
PIPE-182
P76-FRONT-CASING
P76-BEARING-ASSEMBLY
P76-BACK-CASING
P76-LIQUID
P76-STATIONARY-SEAL-RING
P76-STATIC-SEAL-2
P76-STATIC-SEAL-1
water-flow
pump-inlet-pressure
water-temperature
water-density
P76-SPRING-ASSEMBLY
pump-rpm
P76-ROTATING-SEAL-RING
P76-IMPELLER
P76-IMPELLER-WEAR-RING
casing-wear-ring-id
[inside-diameter]
P76-ROTOR
impeller-wear-ring-od
[outside-diameter]
P76-CASING-
WEAR-RING
KEY
UPPERCASE—REGIONS
lowercase——parameters
[ ] indicates types
Describes link
Has-Part link
P76-SHAFT

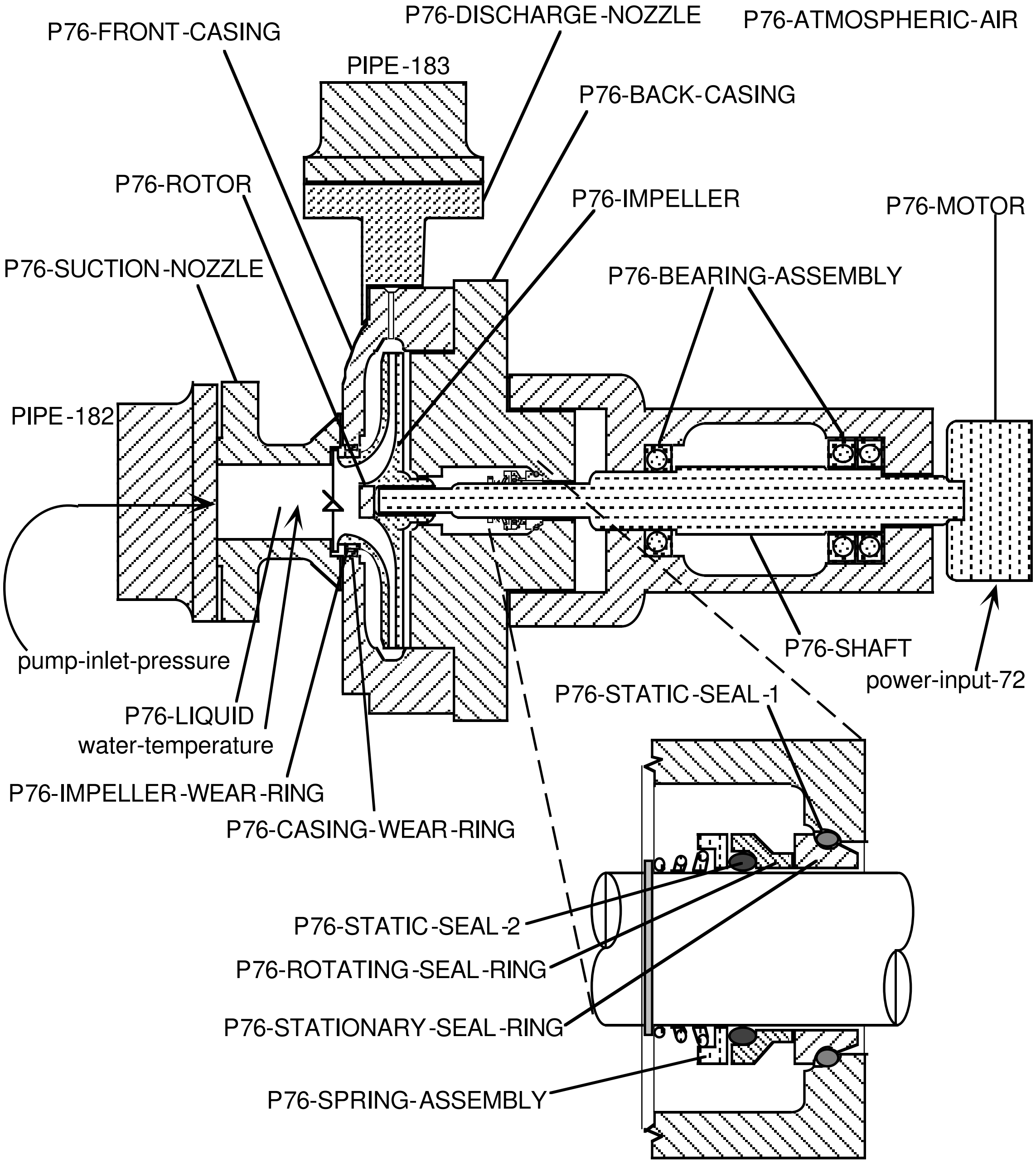


Figure B.21: Equipment represented in the DETAILED-PUMP equipment description.

B.11 DETAILED-PUMP: Single-Stage Centrifugal Pump

Statistics on the DETAILED-PUMP Equipment Description	
Size:	138 K
Individuals:	19
Objects:	187
Parameter Objects:	13
Non-nil Parameter attributes:	171
Links:	655
Objects in Type Hierarchies:*	215
Links in Type Hierarchies:*	237
* Not all objects in the type hierarchies are used in this equipment description.	

Table B.11: Statistics on the DETAILED-PUMP equipment description.

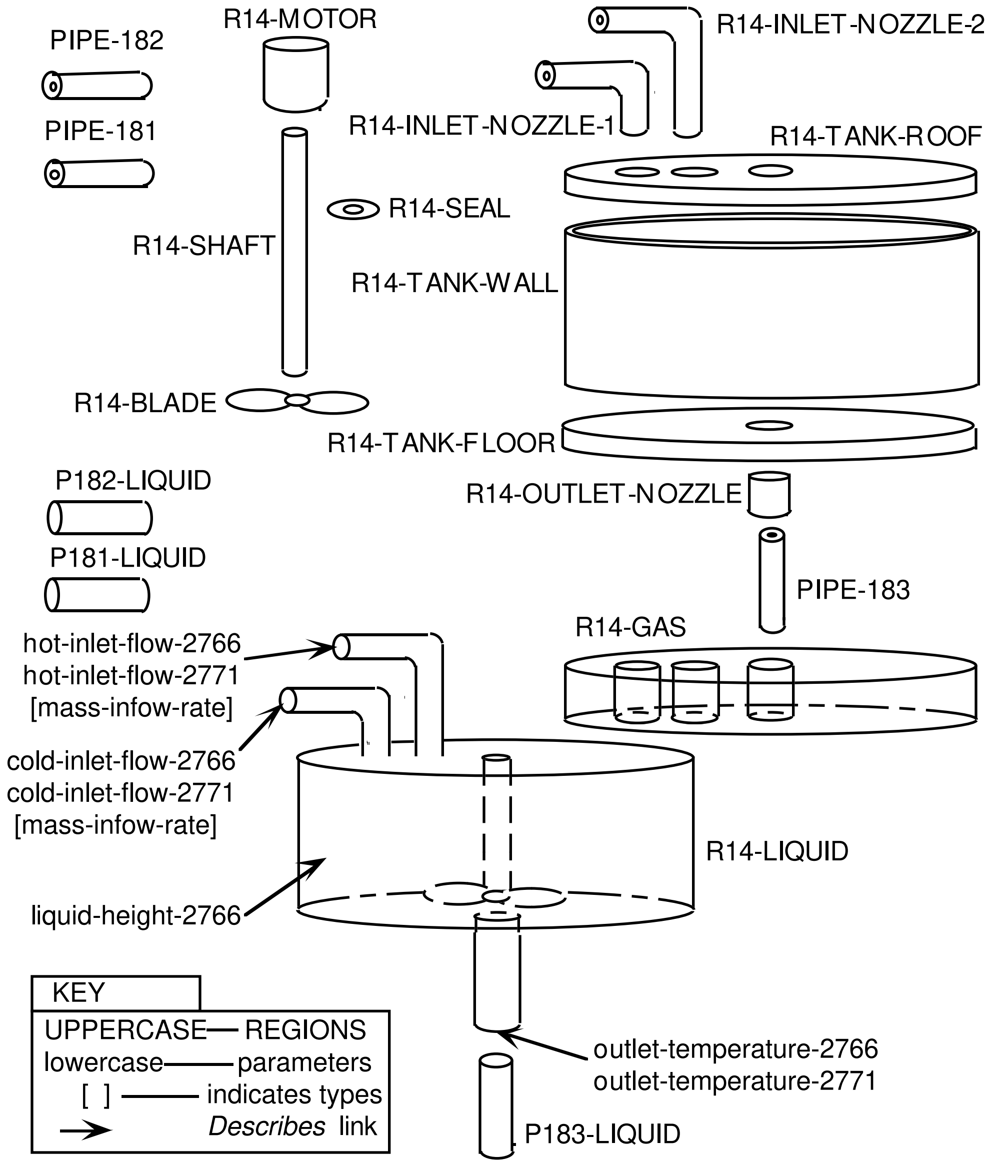


Figure B.20: The individuals represented in the CSTR-3 equipment description.

B.10 CSTR-3: A Stirred Tank Reactor

Statistics on the CSTR-3 Equipment Description	
Size:	139 K
Individuals:	18
Objects:	233
Parameter Objects:	9
Non-nil Parameter attributes:	107
Links:	746
Objects in Type Hierarchies:*	241
Links in Type Hierarchies:*	276

* Not all objects in the type hierarchies are used in this equipment description.

Table B.10: Statistics on the CSTR-3 equipment description.

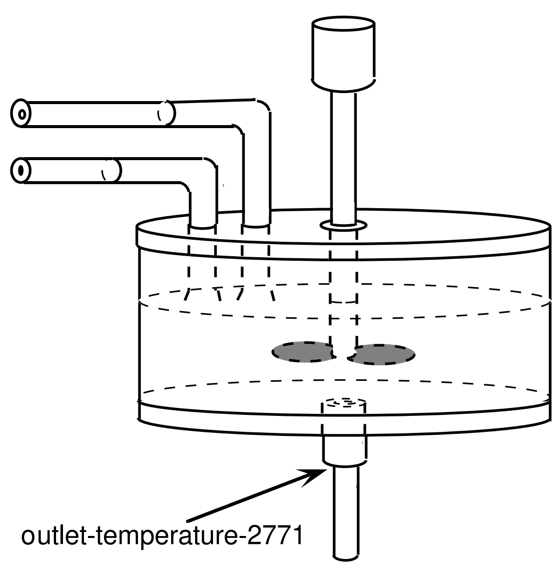


Figure B.19: Equipment represented in the CSTR-3 equipment description.

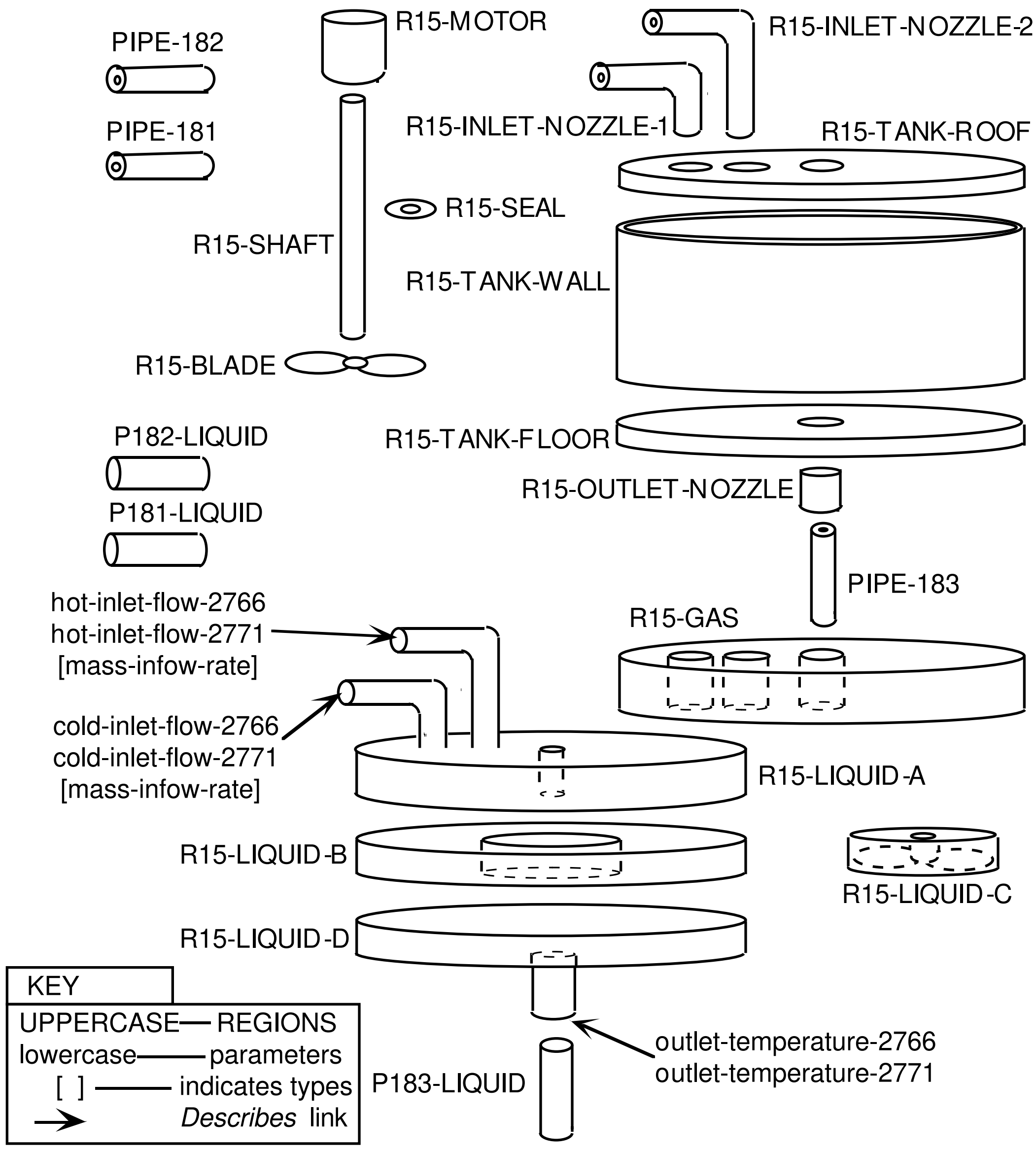


Figure B.18: The individuals represented in the CSTR-2 equipment description.

B.9 CSTR-2: A Stirred Tank Reactor

Statistics on the CSTR-2 Equipment Description	
Size:	135 K
Individuals:	21
Objects:	219
Parameter Objects:	9
Non-nil Parameter attributes:	107
Links:	706
Objects in Type Hierarchies:*	241
Links in Type Hierarchies:*	276

* Not all objects in the type hierarchies are used in this equipment description.

Table B.9: Statistics on the CSTR-2 equipment description.

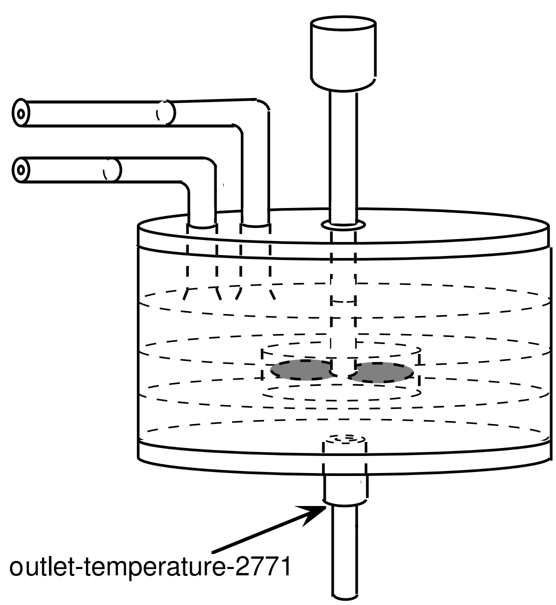


Figure B.17: Equipment represented in the CSTR-2 equipment description.

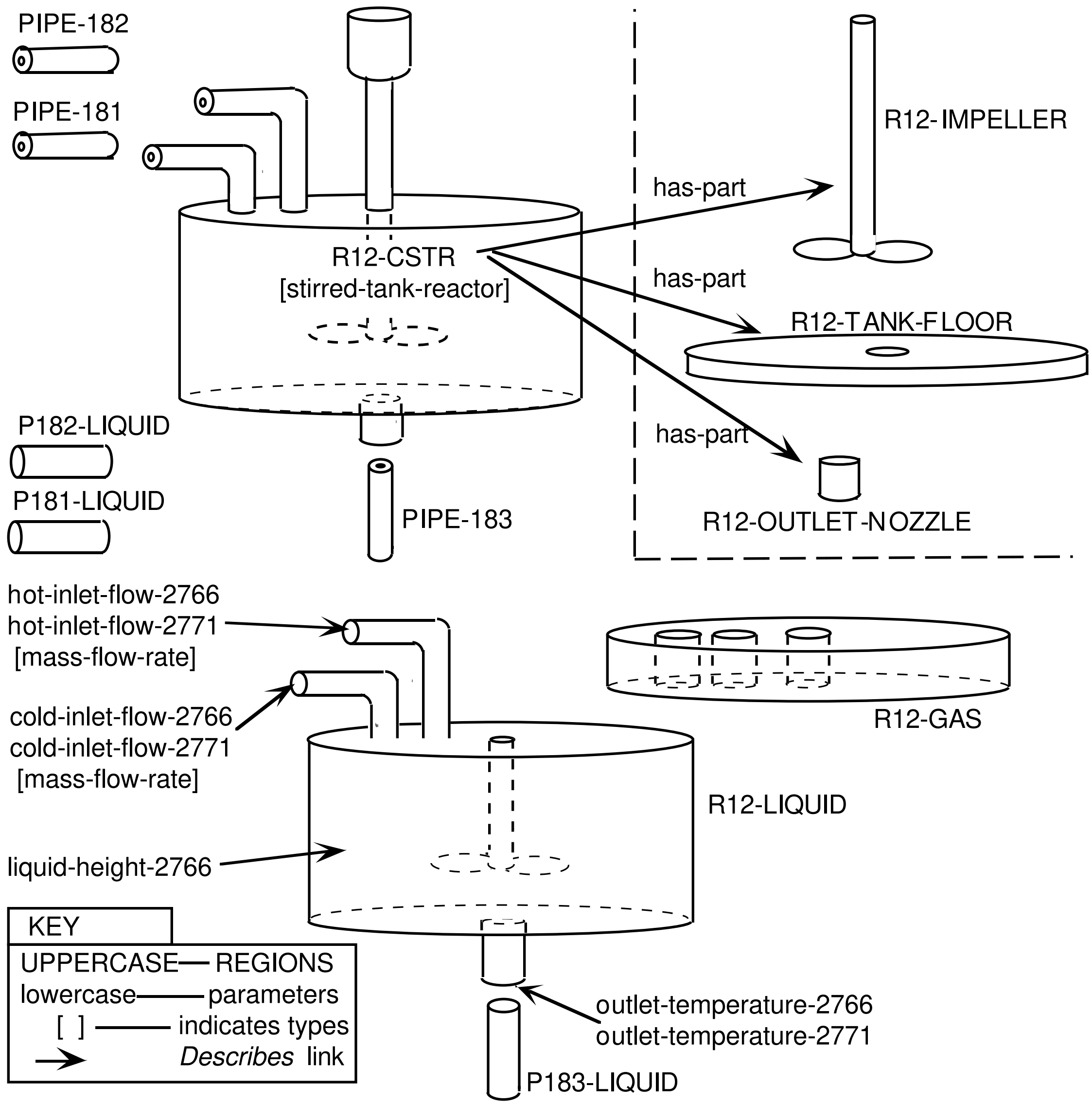


Figure B.16: The individuals represented in the CSTR-1 equipment description.

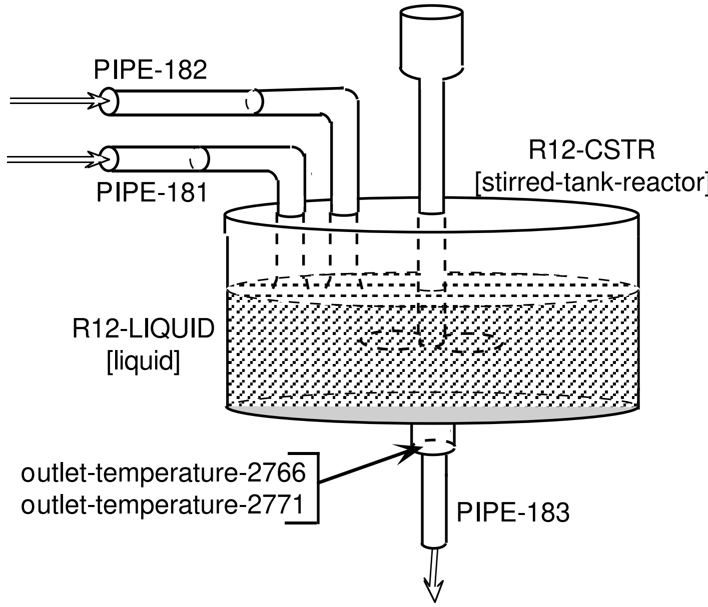


Figure B.15: Equipment represented in the CSTR-1 equipment description.

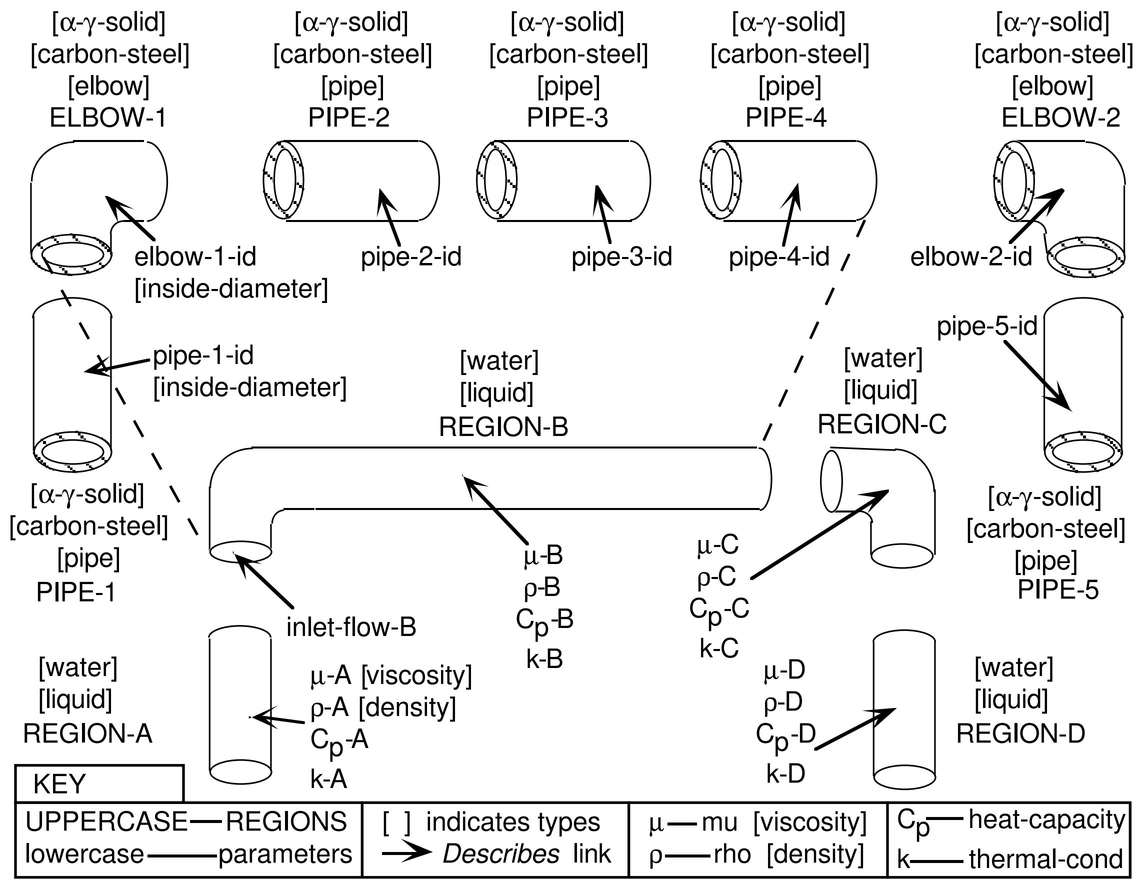


Figure B.14: The individuals represented in the PIPES-6 equipment description.

B.8 CSTR-1: A Stirred Tank Reactor

Statistics on the CSTR-1 Equipment Description	
Size:	92 K
Individuals:	12
Objects:	149
Parameter Objects:	9
Non-nil Parameter attributes:	107
Links:	484
Objects in Type Hierarchies:*	241
Links in Type Hierarchies:*	276

* Not all objects in the type hierarchies are used in this equipment description.

Table B.8: Statistics on the CSTR-1 equipment description.

B.7 PIPES-6: Liquid Flow in Connected Pipes

Statistics on the PIPES-6 Equipment Description	
Size:	85 K
Individuals:	11
Objects:	119
Parameter Objects:	33
Non-nil Parameter attributes:	420
Links:	326
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132
* Not all objects in the type hierarchies are used in this equipment description.	

Table B.7: Statistics on the PIPES-6 equipment description.

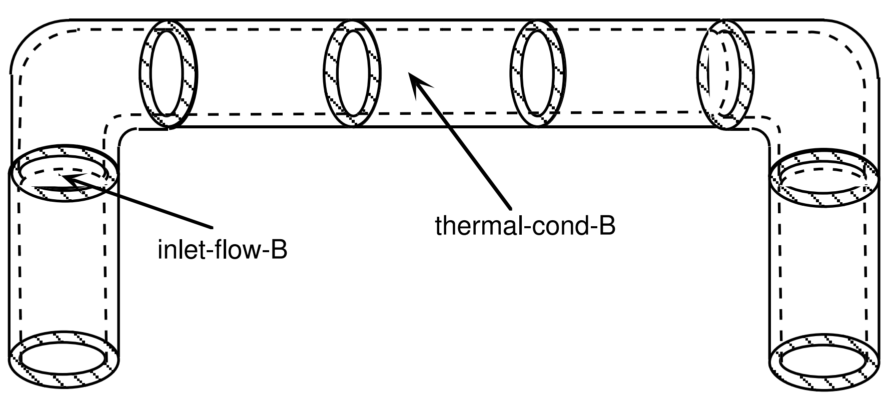


Figure B.13: Equipment represented in the PIPES-6 equipment description.

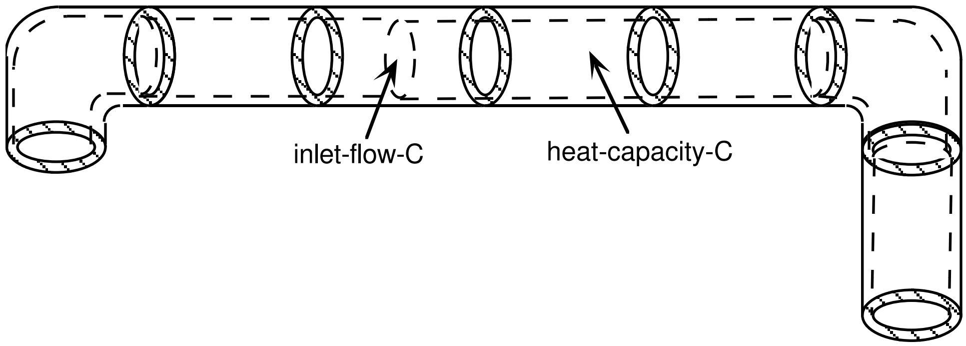


Figure B.11: Equipment represented in the PIPES-5 equipment description.

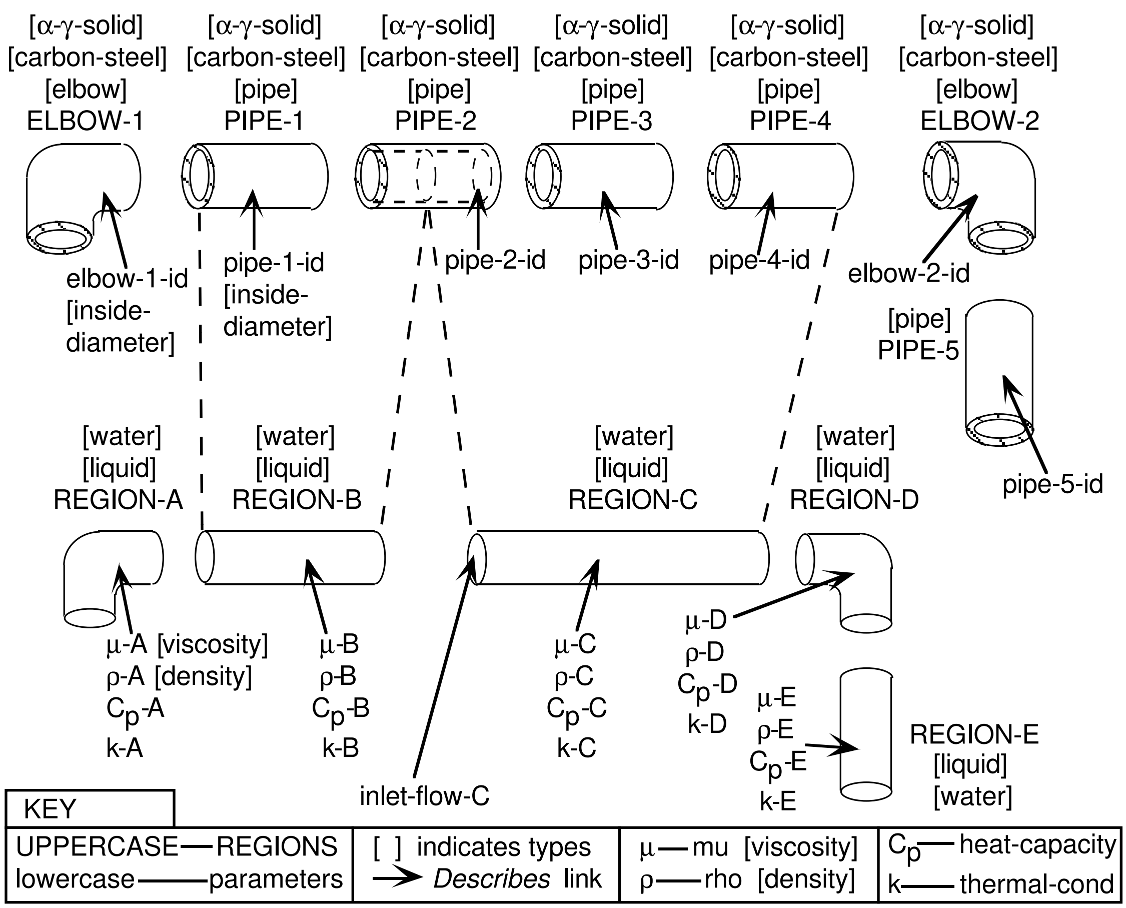


Figure B.12: The individuals represented in the PIPES-5 equipment description.

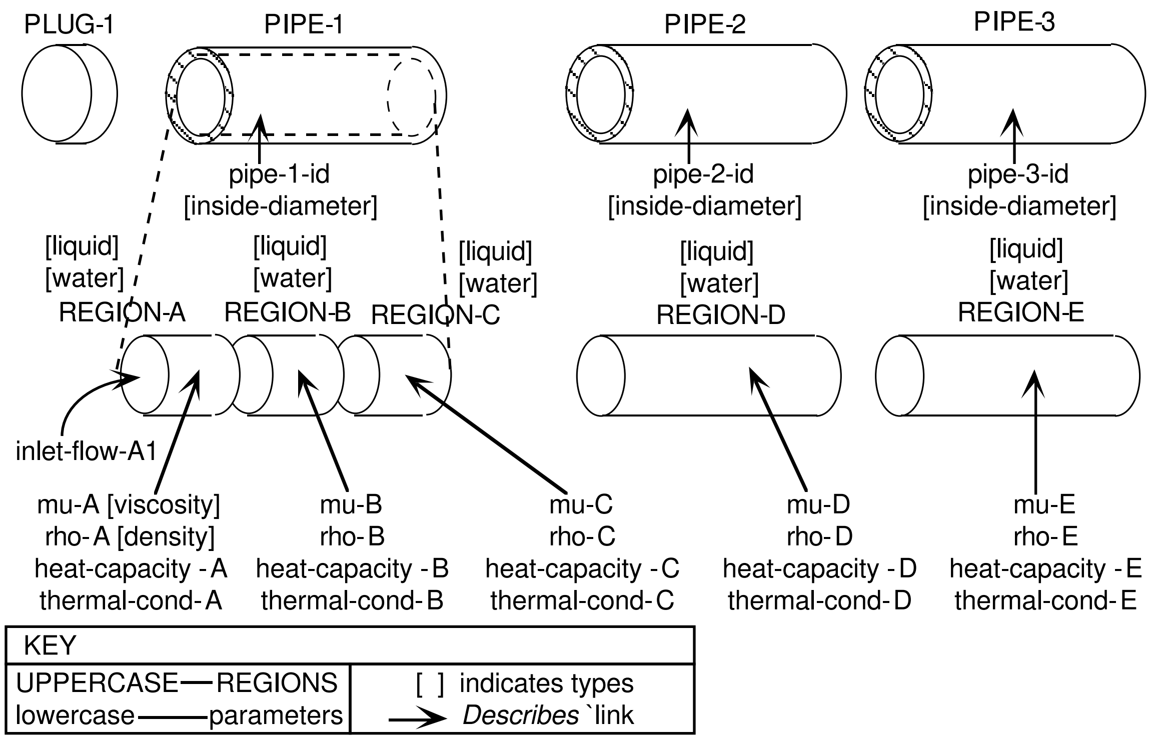


Figure B.10: The individuals represented in the PIPES-4 equipment description.

B.6 PIPES-5: Liquid Flow in Connected Pipes

Statistics on the PIPES-5 Equipment Description	
Size:	95 K
Individuals:	12
Objects:	131
Parameter Objects:	38
Non-nil Parameter attributes:	480
Links:	358
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132

* Not all objects in the type hierarchies are used in this equipment description.

Table B.6: Statistics on the PIPES-5 equipment description.

B.5 PIPES-4: Liquid Flow in Connected Pipes

Statistics on the PIPES-4 Equipment Description	
Size:	72 K
Individuals:	9
Objects:	95
Parameter Objects:	31
Non-nil Parameter attributes:	388
Links:	254
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132

* Not all objects in the type hierarchies are used in this equipment description.

Table B.5: Statistics on the PIPES-4 equipment description.

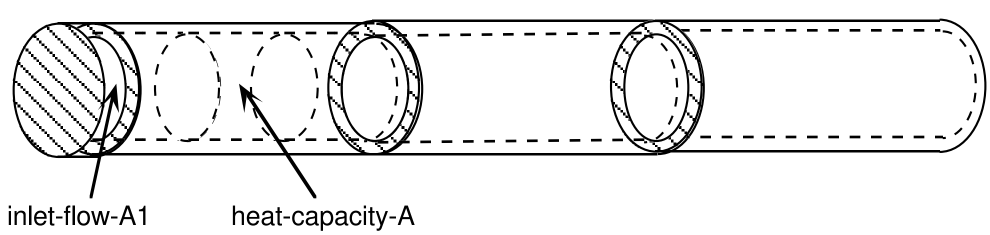


Figure B.9: Equipment represented in the PIPES-4 equipment description.

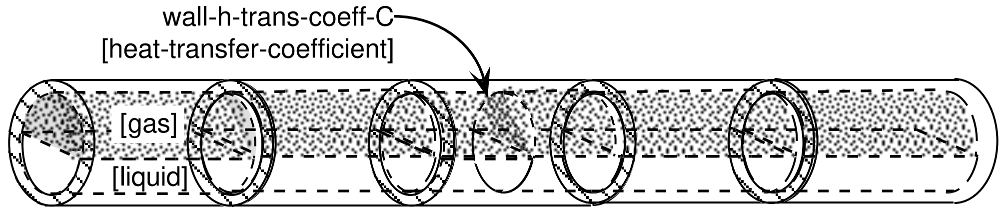


Figure B.7: Equipment represented in the PIPES-3 equipment description.

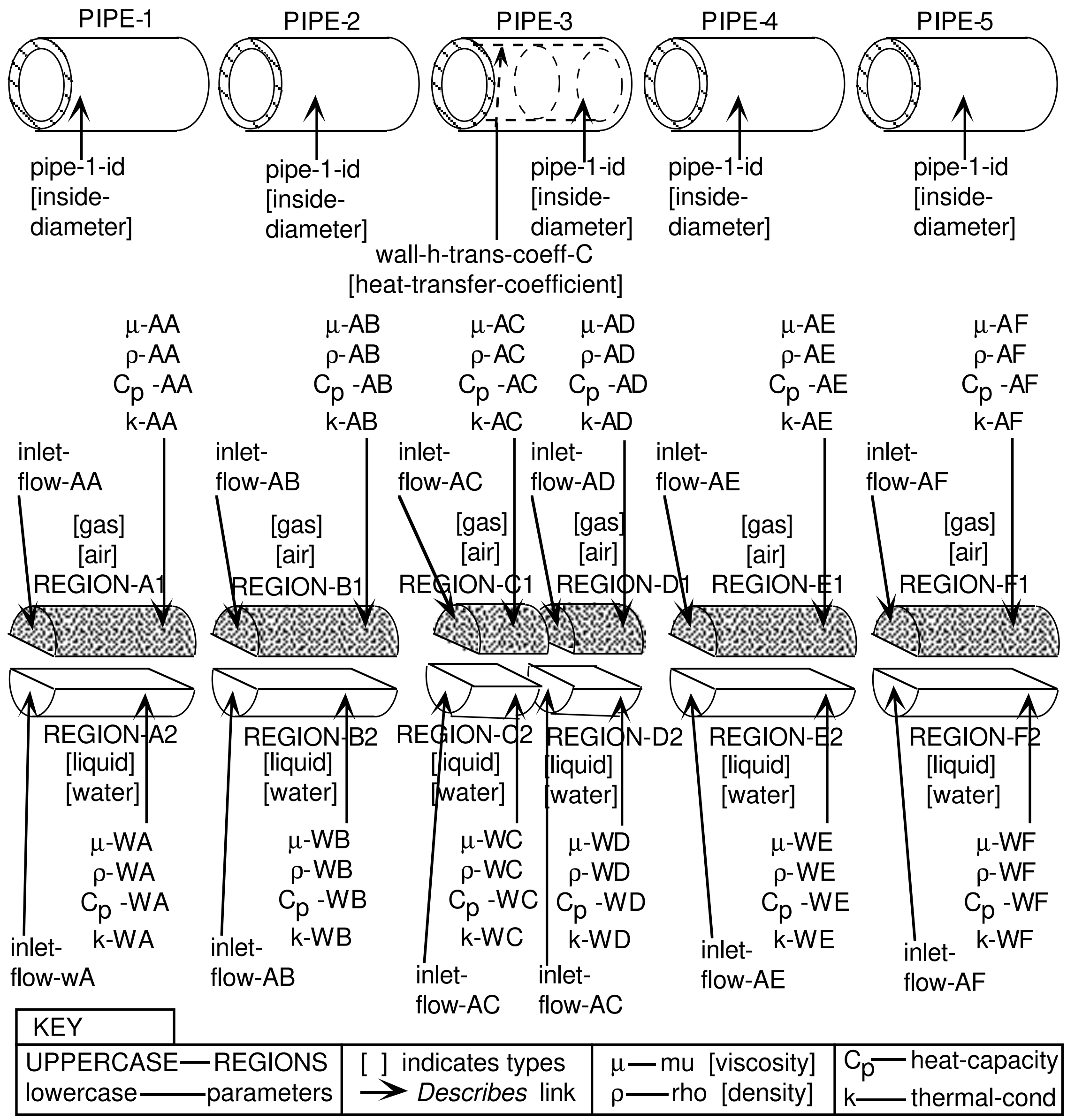


Figure B.8: The individuals represented in the PIPES-3 equipment description.

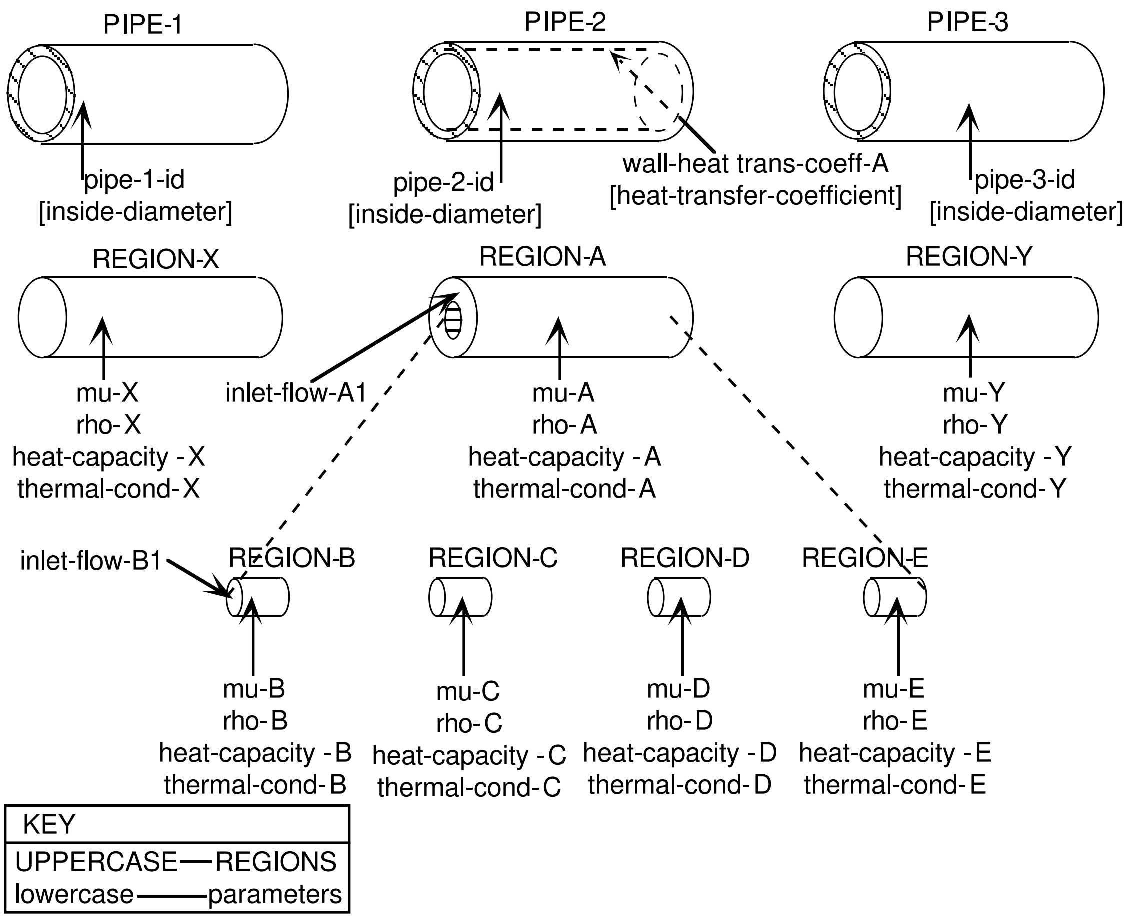


Figure B.6: The individuals represented in the PIPES-2 equipment description.

B.4 PIPES-3: Liquid Flow in Connected Pipes

Statistics on the PIPES-3 Equipment Description	
Size:	195 K
Individuals:	17
Objects:	258
Parameter Objects:	78
Non-nil Parameter attributes:	967
Links:	804
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132
* Not all objects in the type hierarchies are used in this equipment description.	

Table B.4: Statistics on the PIPES-3 equipment description.

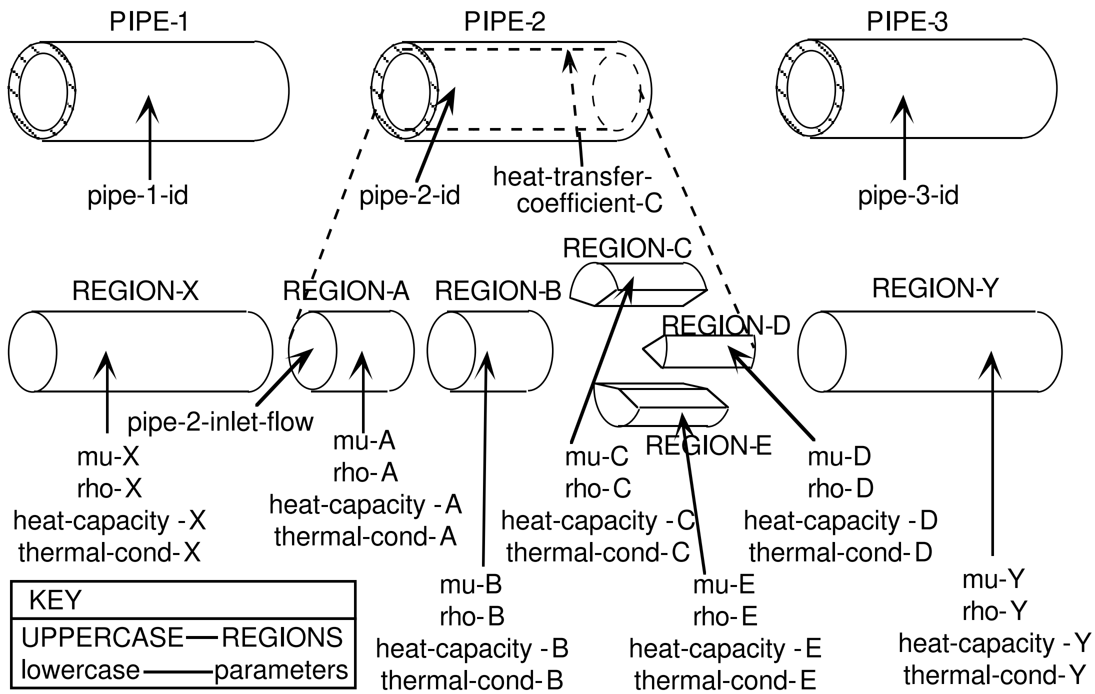


Figure B.4: The individuals represented in the PIPES-1 equipment description.

B.3 PIPES-2: Liquid Flow in Connected Pipes

Statistics on the PIPES-2 Equipment Description	
Size:	107
Individuals:	10
Objects:	130
Parameter Objects:	52
Non-nil Parameter attributes:	698
Links:	337
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132

* Not all objects in the type hierarchies are used in this equipment description.

Table B.3: Statistics on the PIPES-2 equipment description.

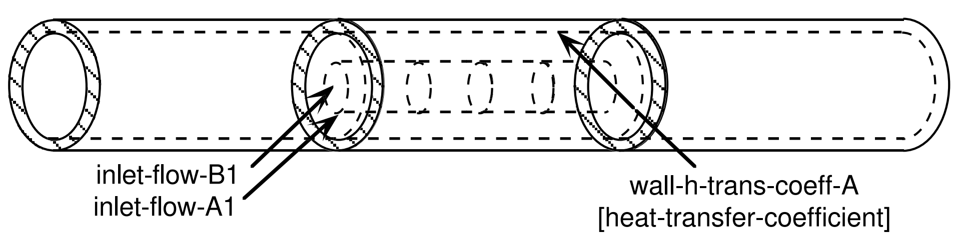


Figure B.5: Equipment represented in the PIPES-2 equipment description.

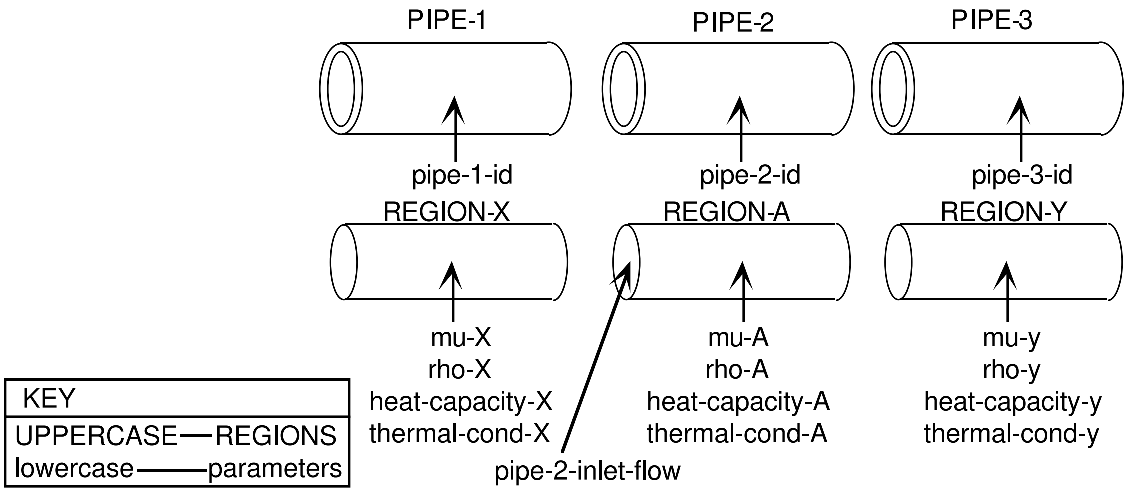


Figure B.2: The individuals represented in the PIPES-0 equipment description.

B.2 PIPES-1: Liquid Flow in Connected Pipes

Statistics on the PIPES-1 Equipment Description	
Size:	144 K
Individuals:	10
Objects:	165
Parameter Objects:	56
Non-nil Parameter attributes:	750
Links:	496
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132

* Not all objects in the type hierarchies are used in this equipment description.

Table B.2: Statistics on the PIPES-1 equipment description.

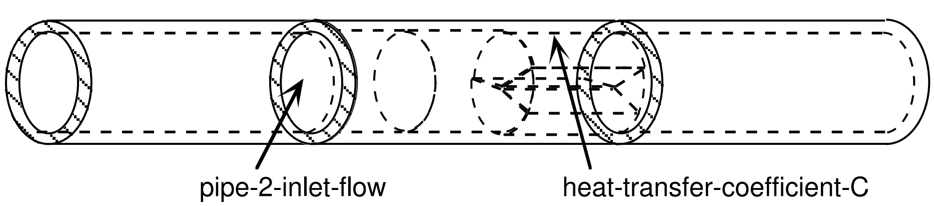
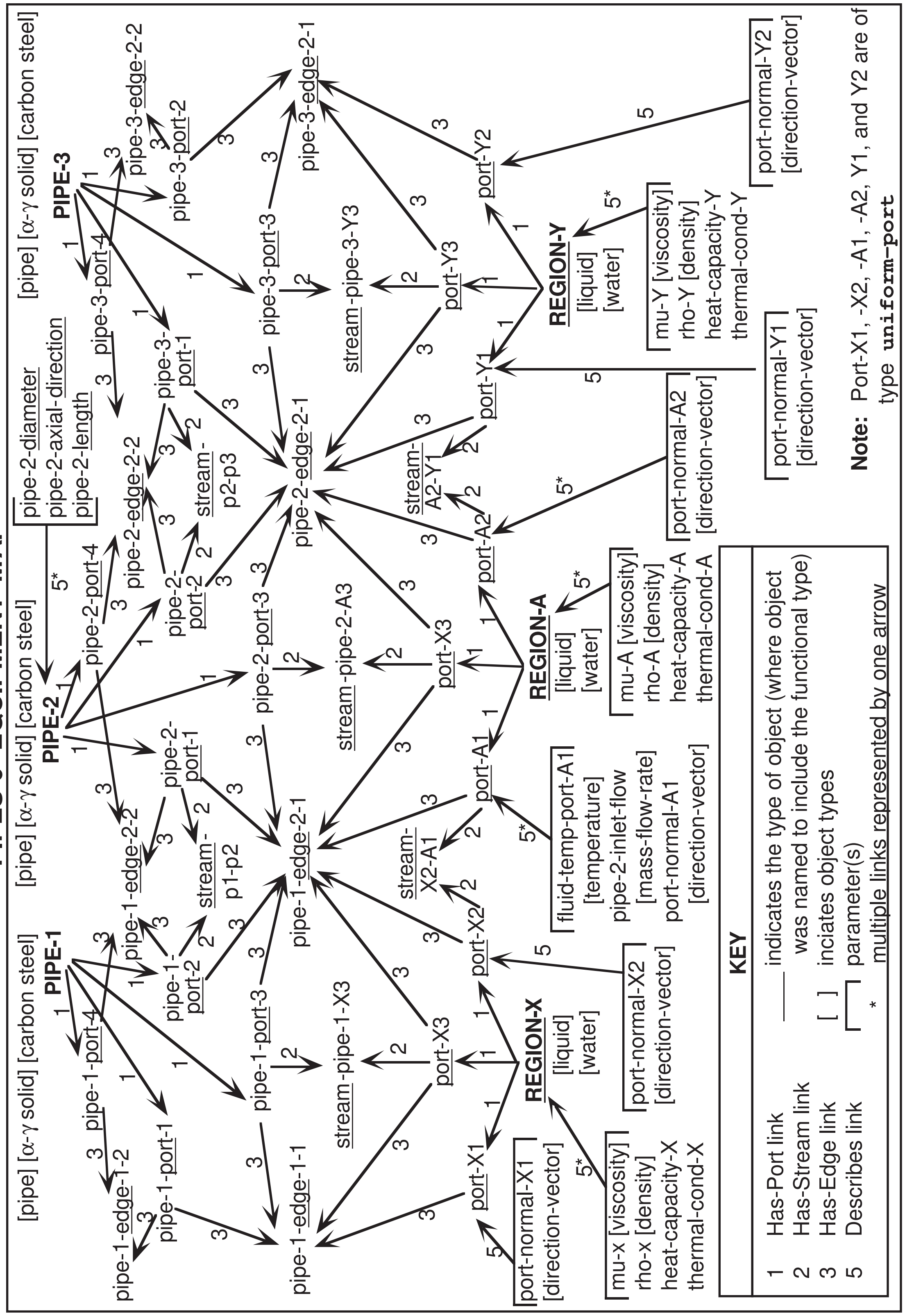


Figure B.3: Equipment represented in the PIPES-1 equipment description.


PIPES-0 EQUIPMENT MAP
PIPE-1
[pipe] [α-γ solid] [carbon steel]
PIPE-2
[pipe] [α-γ solid] [carbon steel]
PIPE-3
[pipe] [α-γ solid] [carbon steel]
pipe-2-diameter
pipe-2-axial-direction
pipe-2-length
pipe-1-port-1
pipe-1-port-2
pipe-1-port-3
pipe-1-port-4
pipe-1-edge-1-1
pipe-1-edge-1-2
pipe-1-edge-2-1
pipe-1-edge-2-2
stream-p1-p2
stream-pipe-1-X3
pipe-2-port-1
pipe-2-port-2
pipe-2-port-3
pipe-2-port-4
pipe-2-edge-2-1
pipe-2-edge-2-2
stream-p2-p3
stream-pipe-2-A3
pipe-3-port-1
pipe-3-port-2
pipe-3-port-3
pipe-3-port-4
pipe-3-edge-2-1
pipe-3-edge-2-2
stream-pipe-3-Y3
port-X1
port-X2
port-X3
port-A1
port-A2
port-Y1
port-Y2
port-Y3
stream-X2-A1
stream-A2-Y1
REGION-X
[liquid]
[water]
REGION-A
[liquid]
[water]
REGION-Y
[liquid]
[water]
mu-x [viscosity]
rho-x [density]
heat-capacity-X
thermal-cond-X
mu-A [viscosity]
rho-A [density]
heat-capacity-A
thermal-cond-A
mu-Y [viscosity]
rho-Y [density]
heat-capacity-Y
thermal-cond-Y
port-normal-X1
[direction-vector]
port-normal-X2
[direction-vector]
port-normal-A2
[direction-vector]
port-normal-Y1
[direction-vector]
port-normal-Y2
[direction-vector]
fluid-temp-port-A1
[temperature]
pipe-2-inlet-flow
[mass-flow-rate]
port-normal-A1
[direction-vector]
KEY
1 Has-Port link
2 Has-Stream link
3 Has-Edge link
5 Describes link
—— indicates the type of object (where object was named to include the functional type)
[ ] inciates object types
parameter(s)
* multiple links represented by one arrow
Note: Port-X1, -X2, -A1, -A2, Y1, and Y2 are of type uniform-port

Appendix B

Implemented Equipment Descriptions

This appendix contains statistics and pictures of the implemented equipment descriptions. We provide a picture of the equipment with its individuals in the correct relative locations. We also provide a breakdown showing the separately represented individuals as separate pieces. Each equipment description was built using the semantic net representation described in Section 4.2. The semantic net for PIPES-1 is included in this appendix to show how equipment descriptions were implemented. The other equipment descriptions are significantly larger than PIPES-1, and their semantic nets are not shown.

B.1 PIPES-0: Liquid Flow in Connected Pipes

Statistics on the PIPES-0 Equipment Description	
Size:	53 K
Individuals:	6
Objects:	66
Parameter Objects:	24
Non-nil Parameter attributes:	318
Links:	171
Objects in Type Hierarchies:*	131
Links in Type Hierarchies:*	132
* Not all objects in the type hierarchies are used in this equipment description.	

Table B.1: Statistics on the PIPES-0 equipment description.

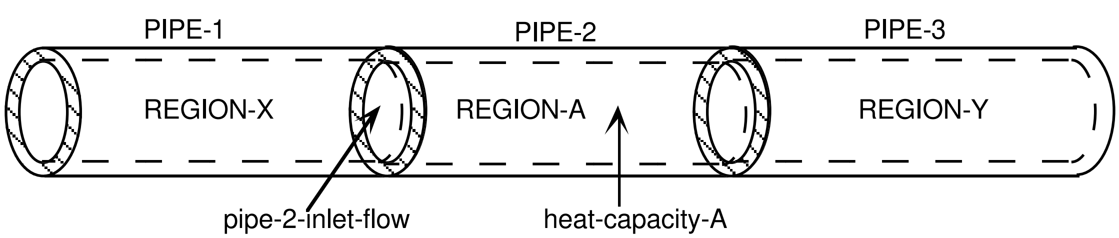


Figure B.1: Equipment represented in the PIPES-0 equipment description.

Gas Bubbles in a Fluidized Bed Reactor. Chapter 9, pages 310–314, [44].

Diffusion-controlled Reaction at Catalyst Surface. Chapter 12, pages 364–365, [44].

Tower Height Required for Transferring Reactant from Gas to Liquid. Chapter 13, pages 428–430, [44].

Tower Height for Fast Reaction in Liquid Phase. Chapter 13, pages 436–437, [44].

Diffusion with Heterogeneous Chemical Reaction. Section 17.3, pages 529-531, [2].

Mass-Transfer Coefficients in Two Phases at Low Mass-Transfer Rates. Section 21.3, pages 652–654, [2].

Height of a Packed-Tower Absorber. Example 22.5-2, pages 692–697, [2].

The Wet-and-Dry-Bulb Psychrometer. Example 21.2-2, pages 649–652, [2].

A.4 Thermodynamics

All thermodynamics examples were found in *Thermodynamics and Its Applications* by Modell and Reid [48].

Work Required to Compress Air into a Tank. Problem 4.18, pages 93-94, [48].

Work done by a "Drinking Bird" Toy. Problem 4.2, pages 78–79, [48].

Temperature and Pressure in a Tanks after Pumping Operations. Problem 4.19, page 94, [48].

Cyclic Pumping. Problem 4.14, pages 90–91, [48].

Ice Evaporation in a Low Pressure Tank. Problem 6.7, page 140, [48].

A.5 Heat Transfer

All heat transfer examples were found in *Theory and Problems of Heat Transfer* by Pitts and Sissom [56].

Heat Transfer through a Portion of Pipe Wall. Problem 2.23, pages 38–39, [56].

Heat Loss from a Vertical Wall Exposed to Nitrogen. Problem 8.5, page 209, [56].

Electrical Power Required to Maintain Wire Temperature. Problem 8.10, page 212, [56].

Heat Flux from a Wire with Constant Current and Voltage Applied. Problem 9.2, page 227, [56].

Heat Flux from the Surface of Boiling Water. Problem 9.5, page 228, [56].

A.6 Reaction Kinetics

All reaction kinetics examples were found in *Chemical Reaction Engineering* by Levenspiel [44].

Multiparameter Models to Account for Non-Ideal Flow. Chapter 9, pages 253–254, [44].

Appendix A

Textbook Examples of Non-Component Models

A.1 Introduction

This collection of examples from engineering textbooks provides examples where the individuals being reasoned about do not correspond to structural components. The principles being used, often based on conservation laws, may be a pplied to in a very large (and sometimes infinte) number different portions of the physical equipment. Usually only 1 or a few ways of defining the individuals for the model are useful, but those cannot be identified until parameter values and locations are available. Because of the large number of individuals that can be identified with a single physical system and reasoned about using these models, the examples indicate that no single a priori breakdown of physical equipment into individuals will suffice.

The collection contains five different examples in each of five areas, fluid mechanics, mass transfer, thermodynamics, heat transfer, and reaction kinetics. Each set of five examples was taken from a single textbook to show that these kinds of problems are not rare or obscure.

A.2 Fluid Mechanics

All fluid mechanics examples were found in *Transport Phenomena* by Bird, Stewart, and Lightfoot [2].

Analysis of a Capillary Flowmeter. Problem 2.L, page 66, [2].

Drainage of Liquids. Problem 2.R, page 69, [2].

Flow of a Falling Film. Section 2.2, pages 36–38, [2].

Velocity Distribution in a Turbulent Plane Jet. Problem 5.H, pages 178–179, [2].

Performance of a Liquid-Liquid Ejector. Example 7.5-2, pages 220–222, [2].

A.3 Mass Transfer

All mass transfer examples were found in *Transport Phenomena* by Bird, Stewart, and Lightfoot [2].

Diffusion through a Stagnant Gas Film. Section 17.2, pages 522-523, [2].

reveal addition characteristics and mechanisms required. This approach could benefit from a more sophisticated geometric reasoner which would also allow more parameter transformations to be included. More investigation of efficiency and complexity is recommended. These methods require a relatively knowledge-rich representation. It may be possible to classify situations in which either matching or reconfiguration does useless work or runs into severe complexity problems and then use the characteristics represented to identify those situations in advance. This thesis represents the first step into an unexplored territory and there are many avenues of research to be pursued in the area of model-matching to physical systems.

4. Findings about reconfiguration methods:

Matching Order. Proceeding through a model map in a spatially breadth first order keeps reconfiguration local. Only the current and adjacent regions within the model and also within the equipment need be considered for reconfigurations because the matcher cannot stray any further than an adjacent region before a mismatch is detected.

Using Space to Make Aggregates. Merging the subparts of an aggregate spatially as the part/whole reconfiguration does is important to retain the relations of the aggregate to neighboring individuals. Models addressing aggregate functional types often require parameters that are not located on the aggregate, but in the space surrounding. (The pump models required parameters describing portions of the liquid inside the pump.)

The general problem of matching models to the physical world is large. Any new model that one encounters combined with any particular physical situation may expose new characteristics or methods required. The set of possible physical situations (perhaps also the set of models) is infinite. Therefore one cannot guarantee that one has identified all the characteristics or methods required. We have identified a necessary (but not sufficient) set of characteristics and developed associated matching methods for some models and physical situations. We have taken the first step into this previously unexplored area.

8.3 Future Work

Our first recommendation for future work is to integrate a model-matching system with a diagnosis engine. Such an experiment could yield results on both the model-matching side as well as the diagnosis side. Integration might point out additional requirements of the model-matching system. A model-matching system, such as the one we have built, presents many more relations on physical entities that must hold for a model's calculated output parameters to be valid than have previously been used in diagnosis. Diagnosis systems typically work with only one relation per model, the relation that says, "component X is functioning properly." This was usually manifested as the type of X or by the relation $isa(X, FunctionalType)$. Our system manifests these kinds of types, but also material and phase types, as well as particular relations between the different spatial entities to which a model has been matched. These additional relations should make it possible to diagnose many new kinds of faults. These additional relations may have implications for the representations or methods used in the diagnosis system as well.

We recommend experimentation on additional mechanisms, models, and physical situations as well as an investigation of complexity and efficiency. We have only implemented a subset of the mechanisms that we observed. Pattern matching and extensive reconfigurations have been left for future work. We have not attempted to deal with models or physical situations where spatial relationships between individuals change so that adjacencies appear or disappear. Nor have our implemented cases required any sophisticated differentiation of the kinds of connections between adjacent individuals. Investigation of additional models and physical situations could

Re-usability:	1. Dittus-Boelter Series (non-component model, 7 equipment descriptions) 2. Dalle Molle Series (component model, 3 equipment descriptions)
Extensibility:	1. Pump Model Addition Series (4 component models) 2. Fluid Flow Model Addition Series (3 non-component models) 3. Use of models with arbitrary internal methods: The Dalle Molle Model

Table 8.1: Summary of the re-usability and extensibility demonstration series.

8.2 Contributions

This research contributes understanding of and ability to solve the model-matching problem, a problem which has not been considered previously. Our contributions fall into four categories.

1. Discovery and demonstration that individuation at run-time is required to use certain kinds of models and to enhance re-usability and extensibility.

2. Identification of characteristics of models and equipment required for run-time model- matching:

   Spatial Representation. For both equipment and models the 3-dimensional space occupied must be represented. Characteristics of that space that were required are:

   - functional types such as `pump` or `reactor`
   - materials
   - phases
   - parameters, including their particular spatial extent

   Model Characteristics. In addition to the spatial representation for the model's physical situation, we found certain other characteristics necessary. (* indicates characteristics not tested by the implemented model-matching cases.)

   - Parameter classifications: causal, affected, input, output, and carried
   - Syntactic information: call forms and return forms
   - Conditions on parameters: approximation conditions and enabling* conditions
   - Resources*

3. Identifications and demonstration of methods required for run-time model-matching:

   - spatial merging and splitting
   - transformation of properties (parameters and types in this work) through spatial merges and splits
   - matching algorithms–positive (simple), negative set, and pattern matching (not implemented)
   - reconfiguration algorithms–intensive, extensive (not implemented) and part/whole

1. Models deal with individuals that occupy contiguous portions of space.

2. All individuals in an equipment description are large enough so that macroscopic properties, such as temperature and pressure, make sense.

We limit our implementations by using a simplified geometric representation, by selecting 2 of the matching methods and 2 of the reconfiguration methods for implementation, and by limiting effort directed towards improving efficiency of the code. The model-matching cases implemented also do not test certain kinds of spatial changes over time nor do they stress differentiation of kinds of connections between adjacent individuals. Within the reconfiguration algorithms we also employ a heuristic to terminate intensive merging and splitting that may cause some matches to be missed. The part/whole reconfiguration algorithm assumes that every subpart of an aggregate piece of equipment has a shared surface with at least one other subpart. This algorithm also assumes that faults within the equipment will be manifested in particular ways.

Since our goal is to identify characteristics and methods required to match models and independently built equipment descriptions at run-time to gain enhanced extensibility and re-usability, we build the system and demonstrate those properties. We provide two series of model-matching cases to demonstrate re-usability and two series to show extensibility. The re-usability series both involve providing a series of equipment descriptions to the system which vary in form but contain the physical situations required to match a particular model. One series is built for a *non-component* model, a model which has at least one individual whose boundaries may be set anywhere through a continuous region of space. The second series is built for a *component* model, a model in which all individuals can be matched only at discrete locations. Similarly, we provide a *non-component* and a *component* series demonstrating extensibility. In those series, we add models that should match the physical situation in the current equipment description, but where the forms specified in the model varied from that in the equipment description. Extensibility also depends on the restrictions the system places on the kinds of models that may be incorporated. We design our system to be able to incorporate models using arbitary internal representation schemes and calculation methods to do their work. If a system is restricted to using only models built in some pre-defined representation, one generally cannot make use of engineering models that are already implemented. Our models used various calculation methods, such as mathematical equations, interpolation in data tables, and numerical methods. Our primary demonstration of this characteristic, though, is to incorporate a model using a large qualitative simulator, QSIM, into our system and successfully match and run this model. Table 8.1 summarizes the demonstration series. Since we are able to demonstrate re-usability and extensibility for some models and equipment descriptions, we conclude that the characteristics and methods identified, implemented, and used in these series form one set of necessary (but not sufficient) characteristics and methods for solving the model-matching problem.

novel in this aspect. In previous work, it has been assumed that the individuals can be identified in advance and the 3-dimensional space can be largely forgotten. Our method actually reconfigures the individuals in an equipment description, allowing a user to build the description initially with any size individuals that they find convenient. Certain characteristics of models also require that the 3-dimensional space be present. Some models apply to regions containing a particular material or phase, where that region may be a portion of some larger object. All models specified parameters that have particular locations and particular spatial extents not necessarily corresponding to whole individuals. A parameter may describe some volume of space, a surface, a curve, or a single point. Without the 3-dimensional space, one cannot identify regions within larger objects nor can one specify and match parameters correctly.

After identifying characteristics and methods, we built a matching system employing those methods. Our system takes as input a parameter in an equipment description and a goal with respect to that parameter. Our system selects models from its collection that can address the goal for the parameter, matches models from the selected set to the equipment description surrounding the input parameter, checks numeric conditions on matched parametrs, and calls the matched model to calculate its outputs. Our system will, by side effect, reconfigure the equipment description (defining different individuals) if a match cannot be made and if its reconfiguration methods are appropriate for the situation at hand. The system also installs the values calculated by a model as parameters in the equipment description, also by side effect. The system returns the values it calculated, the model conditions which it could not test at the time, and the relations which had to be matched for the model to apply. Model conditions which were not verified may be treated as assumptions by a diagnosis system. Diagnosis systems also use the relations upon which the model relies as the basis for generating fault candidates.

The bulk of the matching system's efforts lie in matching and reconfiguring. Of the three kinds of specifications for entities in model descriptions, *positive, negative sets,* and *patterns*, we implement algorithms for matching positive entities and negative sets of entities. (Negative sets specify sets of entities and relations not allowed to be present in correct matches.) Reconfigurations should be done when the right things are present but are in the wrong form. We identify three different reconfiguration methods, based on observations of what physical entities a model might require and how those physical entities could appear in equipment descriptions. We call these mechanisms *intensive, extensive,* and *part/whole* reconfiguration. Intensive reconfiguration deals with individuals whose amount is not specified. Regions which only have either material or phase specified fall into this category, assuming that no parameter (such as a mass) specifies the amount. Extensive reconfiguration addresses similar types of individuals where the amount is specified by a paramter. Part/whole reconfiguration assembles subcomponents of a larger piece of equipment (like assembling casing, shaft and impeller into a centrifugal pump) when a model requires an individual of the aggregate type that is not explicitly represented. We implement the *intensive* and *part/whole* mechanisms for this thesis.

The limitations in this work arise from our fundamental assumptions, selected implementation effort, and assumptions underlying the implementations. The fundamental assumptions are:

Chapter 8

Conclusions

8.1 Summary of Thesis

We investigate the problem of model-matching, assigning models to portions of physical systems, for use in model-based diagnosis systems. In previous work on model-based diagnosis and other kinds of reasoning about physical systems, the assignment was assumed to be given, and was usually done by hand at system building time. If the models and equipment were separate, and if an automated matcher can do that work at run-time, we can make these systems significantly more re-usable and extensible. One can give an existing system a new piece of equipment (in some suitable representation) without going through the collection of models in the system and identifying every place in the equipment where each model might apply. One can also give such a system a new model, thereby extending the system's capabilities, without finding every location in the equipment where it should be used. This obviously requires that an equipment description be provided to the system as well as some characterization of the situations in which a model may apply. Our goal is to identify the characteristics of both models and equipment that are required as well as what methods are needed to do the run-time model assignment so that we could achieve enhanced re-usability and extensibility. We investigate examples of engineering model assignment in both manufacturing equipment and in textbooks. Using the characteristics and methods identified, we build a system to match models, and then demonstrate re-usability and extensibility for a few models and equipment descriptions. If we can demonstrate the re-usability and extensibility for some models and equipment, we can conclude that we had successfully identified at least some subset of the set of required characteristics and methods.

Our investigations of textbook and manufacturing examples lead us to the conclusion that *individuals*, the portions of the equipment which are identified as separate entities, can not be defined in advance. There are two reasons.

1. Some models, like the Ideal Gas Law, require individuals whose boundaries may be set anywhere through some region of continuous space, depending on parameter availability and values.

2. If models can be added at any time, we cannot predict what individuals those models may require.

Our approach is to leave the 3-dimensional space in the equipment and model descriptions and to conceptually carve the individuals required out of the space at matching time. Our approach is equipment also occupies some region of contiguous space. Our matching methods would fail, if either the equipment or the model could not be represented as contiguous, because the matcher works through space to find make a match. The Macroscopic Properties Assumption requires that the regions we deal with be at least a few molecules, so that properties such as temperature and pressure will be meaningful. The reconfigurer cannot make the transition to reasoning about molecules as balls moving around in empty space, because the type and parameter transforms are not valid.

We also implement only a selected portion of the mechanisms that appear to be needed to match engineering models to equipment. The geometric representation used is a semiquantitative semantic net. This prevents representing some kinds of models and equipment details. It also restricts the regions that can be formed, because splitting is limited to simply connected regions which already have edges where a split can be made. We classify parameters and implement parameter transformations for the classes, but only examine limited number of parameters. Furthermore, the limited geometric representation does not allow some parameter transformations to be implemented. We identify three matching methods, simple, negative, and pattern matching, and implement simple and negative matching. We identify three reconfiguration methods, intensive, extensive, and part/whole, and implement intensive and part/whole reconfiguration. The examples implemented also are known not to exhibit two kinds of characteristics. First they do not involve appearance or disappearance of an adjacency (surface or `port`) between individuals, although sizes of adjacent surfaces did vary. Second, they do very little distinguishing between different types of connections. In the implemented cases, it is generally required only that the individuals abut. We also choose to limit attempts to improve the efficiency of implementations to what was required to achieve our original research goals. There are both research issues and implementation issues that could be addressed to improve efficiency.

The matching and reconfiguration implementations rely on some assumptions. The intensive reconfiguration algorithm uses a heuristic to terminate merging and splitting on a region when the last merge or split produced a region that did not match as well as the region that triggered the reconfiguration. This is a form of hill climbing and will only find local optima. (Section 6.4.2 contains the details.) The part/whole reconfiguration mechanism also makes assumptions, first that all subparts of an aggregate piece of equipment have at least one shared surface with another subpart. This assumption may not be valid for all kinds of equipment. The second assumption is that faults in the equipment will manifest themselves as either missing or changed types for a subpart or completely disconnected subparts. If the subparts of a piece of equipment are all present but somehow faulty, and the fault is not manifested in one of the two assumed ways, the part/whole mechanism will aggregate the parts into the whole equipment and assign it a type as if it were a correctly functioning aggregate. (Section 6.5.2 contains more details on the part/whole reconfiguration assumptions.)

situation to avoid, one where a region has a large number of surfaces to other individuals. This situation occurs when one region is large relative to its neighbors. One might devise an automatic method for restructuring equipment descriptions exhibiting this characteristic. One might also use some form of lookahead in the matching to avoid considering some of the surfaces which do not lead to correct matches. We leave this for future work.

Additional complexity is generated by the reconfigurations, since they require at least partial rematching. Part/whole reconfigurations will only usually be able to make one aggregate out of the components existing at the point of failure, and thus only one rematch is required. However, intensive reconfiguration involves a search through a number of splits and merges. Table 7.1 lists the number of rematches that occurs in each case. We implemented a termination heuristic (Section 6.4.2) to prevent reconfigurations that are probably useless. However, the intensive reconfiguration algorithm does a crude generate and test. It merges or splits regions that have the right `material` and `phase` types and are adjacent to the region at which the match failed. A more sophisticated generation mechanism could reduce the number of reconfigurations and their associated rematches. An example is a case where the matcher is looking for a particular kind of parameter. No merging or splitting can generate the parameter unless it is already available somewhere within the regions to be merged or split. By looking ahead to find the parameter, several merges could be done at one time, rematching after the final merge. Lookahead could also prevent some splits from being made or rematched. We leave this investigation for future work.

The overall efficiency of our implementations, as evidenced by the running times, is probably not good enough to allow the system to be incorporated into a diagnosis engine. However, it is likely that significant improvements could be made with relative ease. The implementations themselves have not been fully optimized. They were developed to have sufficient speed to carry out the research. The implementations also sit on top of a version of BB1, the blackboard system that provides the object oriented framework, which has some known inefficiencies. One major inefficiency involves the copying of lists of links whenever the links of an object are accessed. For most accesses, links should not by copied. The excessive copying slows match generation, where links are accessed, and also causes unnecessary garbage collection. This inefficiency has been corrected in a recently completed version of BB1. The matching system could also be speeded up by a change of platform. It currently runs on an Explorer II, a lisp machine manufactured by Texas Instruments. Porting our system to newer versions of BB1, which run in Lisp or C++ on UNIX platforms, would speed up the matching substantially.

7.5 Limitations of the Approach and Implementation

Limitations arise from three sources, fundamental assumptions underlying the approach, selective implementation, and assumptions made for implementations. The fundamental assumptions underlying our design are the Contiguous Space Assumption and the Macroscopic Properties Assumption, both discussed in Section 3.4.2. The Contiguous Space Assumption says that models require individuals that occupy contiguous space, that is, individuals that touch each other, and that the

Appendix	Equipment Description		Model Description				Run-time (hr:min)	approx. worst case $4^{(n-1)}$	max. # partial matches	# rematches
	Name	Size	Name	size	#objects n	# links				
D.2	PIPES-0	53 K	Dittus-Boelter	54 K	34	67	0:01	$7x10^{19}$	4	0
D.3	PIPES-1	144 K	Dittus-Boelter	54 K	34	67	1:29	$7x10^{19}$	312	23
D.4	PIPES-2	107 K	Dittus-Boelter	54 K	34	67	1:25	$7x10^{19}$	120	35
D.5	PIPES-3	195 K	Dittus-Boelter	54 K	34	67	0:11	$7x10^{19}$	156	4
D.6	PIPES-4	72 K	Dittus-Boelter	54 K	34	67	0:01	$7x10^{19}$	8	3
D.7	PIPES-5	95 K	Dittus-Boelter	54 K	34	67	0:14	$7x10^{19}$	24	18
D.8	PIPES-6	85 K	Dittus-Boelter	54 K	34	67	0:15	$7x10^{19}$	40	20
D.9	CSTR-1	92 K	Dalle-Molle	61 K	40	82	0:13	$3x10^{23}$	240	0
D.10	CSTR-2	135 K	Dalle-Molle	61 K	40	82	4:46	$3x10^{23}$	1840	33
D.11	CSTR-3	139 K	Dalle-Molle	61 K	40	82	2:04	$3x10^{23}$	2048	2
D.12	Detailed-Pump	138 K	NPSH	36 K	38	84	0:08	$2x10^{22}$	24	1
D.13	Detailed-Pump	138 K	Wear Ring	42 K	32	92	0:10	$5x10^{18}$	13	1
D.14	Detailed-Pump	138 K	Hydraulic HP	32 K	31	71	0:11	$1x10^{18}$	59	1
D.15	Detailed-Pump	138 K	Hypo-thetical	34 K	39	85	0:18	$8x10^{22}$	52	2
D.16	PIPES-1	144 K	Friction Factor	30 K	25	54	1:38	$3x10^{14}$	816	12
D.17	PIPES-1	144 K	Hagen-Poiseuille	47 K	33	70	0:02	$2x10^{19}$	30	0

Table 7.1: Observed complexity and run times for the 16 implemented matching cases.

Each of these 3 kinds of additional information are used during matching, and will eliminate some of the isomorphisms that one could find in a graph containing only nodes and links without labels. For the purposes of theoretical analysis, though, the amount or distinguishing power provided by this information cannot be guaranteed. It could be that all node types are the same, link types are the same, and parameters are the same in both the model map and equipment description. When the types and attributes are removed, we are left with the problem of finding a subgraph isomorphism with an anchored starting point. Small changes in the definition of a problem, like this anchored starting point, can move the problem from the NP-complete class to the polynomial class in terms of complexity. However, in this case, we construct a proof showing the problem with the anchored starting was still NP-complete. Appendix F contains the proof.

The proof of worst case complexity indicates that we could see problems where the number of steps required to solve the problem is exponential in the number of nodes in the smaller graph (the model map). In the matching algorithm we have implemented, matches are built node by node in a breadth first fashion. The matcher finds all correct partial matches with 2 nodes. Then for each of those matches, it finds correct partial matches with 3 nodes, and continues the process until all the nodes and their associated links have been matched. At each step, the number of partial correct matches can be multiplied by the number of links from the node already in the match from which we are searching in the breadth first order. Because of the exponential nature of this process, the number of matches made at the nth node in the model map will swamp the number of matches made at the $n-1$ node. Thus we simply keep track of the maximum number of correct partial matches that the matcher had accumulated at any point in the matching to measure the complexity occurring in implemented cases. These numbers are summarized in Table 7.1 along with the sizes of the model maps and running times. The table also lists an approximate theoretical worst case, which is calculated from the average number of links per node in equipment maps, which is about 4, and the number of nodes in the particular model map. All of the cases we implemented exhibit a number of orders of magnitude less than the worst case. This evidence suggests that the types, link relations, and parameter attributes provide distinguishing information that is eliminating most of the partial matches before the numbers get large.

The evidence indicates that even though the underlying algorithm may be exponential in its computational complexity, in practice the worst case may not occur. We implement only a small number of examples from a perhaps infinite space of examples, but the evidence is encouraging. It may also be possible, upon observing enough implemented examples, to predict circumstances under which we get a large number of partial matches that are later eliminated, and thus avoid some of the bad cases. In this work, the two cases that exhibited significantly worse complexity were two matchings of the Dalle Molle model (Appendices D.10 and D.11). By observing the matches as they proceed, we find that the large number of partial matches are being generated when matching ports (surfaces) of the liquid region. The liquid region in these two equipment descriptions has a large number of surfaces, a surface to each of the separately represented components of the stirred tank reactor. The model has only a few surfaces to be matched at the same point. Thus a large number of correct partial matches were generated at that point in the matching. This may be a

One ran the model by defining a lisp function containing the parameters and then calling that function without arguments. The results were displayed on a screen in graphical form for humans to observe. To make the model fit our function-call assumption, we provide a simple interface, a lisp macro (called `simulate-mixing-tank`) that takes parameters as arguments and generated the necessary function on the fly. This macro with 2 supporting functions is about 2.5 pages of Lisp source code and is shown in Appendix E. It contains some parameter generating code (in the first `let*` construct) and some output handling code (in the last `let` construct) which are necessary because QSIM usually presents its outputs not as parameters but as a graph displayed on a screen for a human user. The middle portion of the macro (the `'progn` construct) generates exactly the modeling function as presented in [11], which is also included in the appendix. Since our system requires that models be *functions* and not *macros*, the function `dalle-molle-simulation` was written to take arguments to the model, dynamically load QSIM and the model, and call the macro that executes the model. With the function-call interface installed, we were able to successfully match and execute the Dalle Molle model, providing a demonstration that the system can incorporate models with various internal representations and methods without knowing or rewriting the internal methods. Appendices D.9, D.10, and D.11 give details of the matches and executions of this model.

7.4 Computational Complexity and Efficiency

Since this research is the first look at a previously unexplored problem, we do not set any goals regarding efficiency. We investigate what knowledge had to be made explicit as well as what a matcher had to do, analyzing the algorithms and observing the actual complexity occurring in our implemented examples. Of the four steps in the matching (generating a set of models that are potential matches, matching the model map, checking model conditions, and executing the model), all are simple operations except the matching. Our analysis focuses on that operation.

When matching the objects and links in a model map to those in an equipment description, the matcher is doing a problem very similar to the known NP-complete *subgraph isomorphism* problem [30]. In *subgraph isomorphism*, given two graphs composed of nodes and links which have no types or other distinguishing information, the problem is to find whether one graph contains a subgraph that is isomorphic to the other graph. The matching problem in our semantic nets has two differences from the subgraph isomorphism problem. First, the matcher is given a starting point. It is given a model parameter that is known to match an equipment parameter by PMSG, the potential match set generator. The objects representing these two parameters are therefore tied to each other and the matcher can only consider isomorphisms where those nodes are matched. The second difference is the additional information available.

1. Links are labelled with one of the 13 relations or their inverses.

2. Nodes have up to four different types attached.

3. Nodes that are parameters have up to 29 attributes.

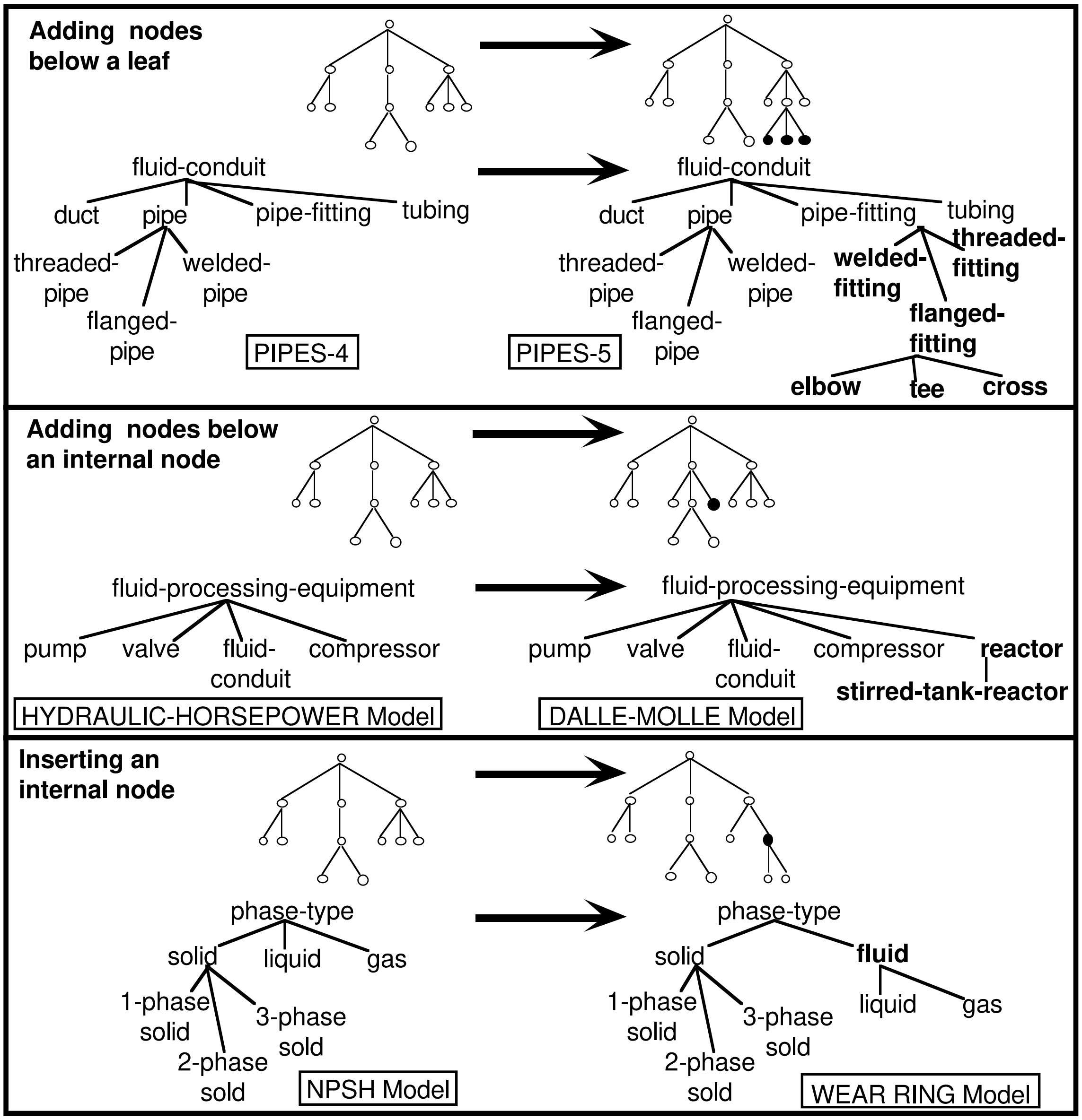


Figure 7.8: Extensions made to the vocabulary hierarchies.

of data to calculate its outputs. We also built and matched a model of a heat exchanger which used a numerical method. (This model was not used in an extensibility or reusability series.) However, to make a stronger demonstration, we have incorporated a large model that was implemented by others, the Dalle Molle model (Appendix C.1). This model simulates hot and cold streams mixing in a stirred tank reactor and was built by David Dalle Molle [11] as part of his thesis work. It uses the QSIM qualitative simulator [42] which consists of about 1.2 M of Common Lisp code.

We choose this model for our demonstration because it is large, it was built by others, it is readily available, and it runs on our platform. The model did not meet our assumptions regarding models, however. It was not executable by calling a function with arguments that returned parameters.

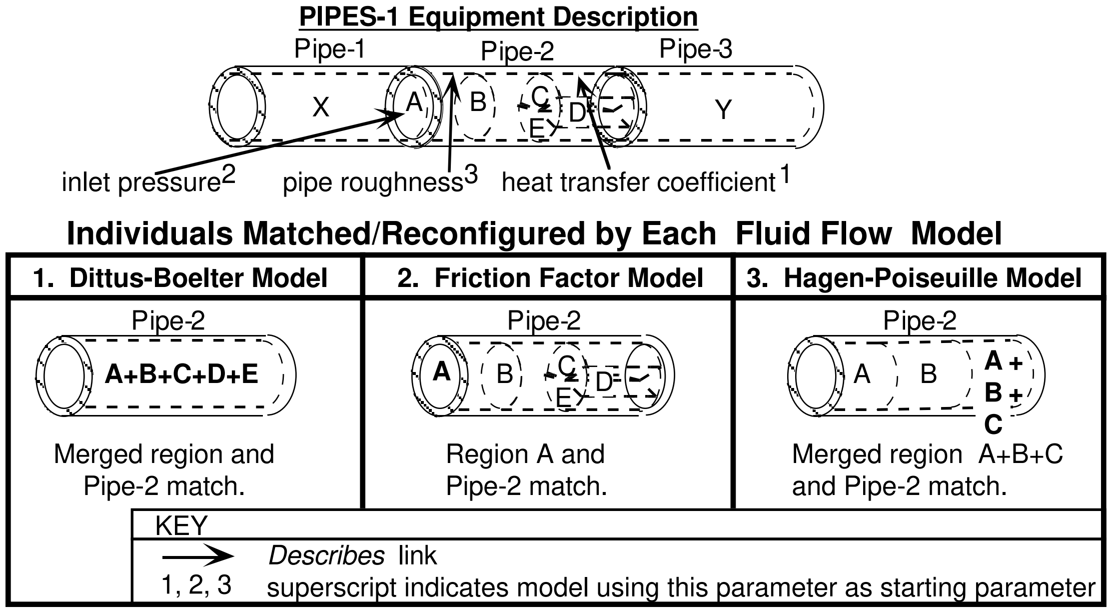


Figure 7.7: Reconfigurations made and individuals matched in the fluid flow extensibility series.

previously calculate.

7.3.3 Extending the Type Hierarchies

Our algorithms rely only on the partial order within the type hierarchies, and not the specific types, except for the parameter classifications used by the parameter transforms (Section 6.2.2) and the types used by the merger and splitter, `region, port, stream, edge`, and `endpoint`. This aspect of our approach is not novel, but we provide some evidence here to verify the independence of the algorithms from the specific types. As this work progressed, we used the same hierarchies for all test cases, adding types as either the added equipment descriptions or the added models required. We added nodes below a leaf, nodes below an internal node, and inserted internal nodes which the algorithms proceeded to use correctly in matchings. Figure 7.8 shows an example of each of these types of additions.

7.3.4 Diversity of Model Calculation Methods

It is important for extensibility to be able to include models that use various representation schemes and calculation methods, so that one can add models that already exist and so that one can accomodate the variety of methods that are used in engineering models. By design, our matching system does not access the internals of the model. It only finds input parameters, passes those in the correct form to the model's function, and accepts outputs. To demonstrate that the algorithms can work with models using various calculation methods, we have included models with different internal methods. Many of the models use mathematical equations, such as the Dittus-Boelter model, but each was arbitrarily and independently implemented. The NPSH model does interpolations in tables

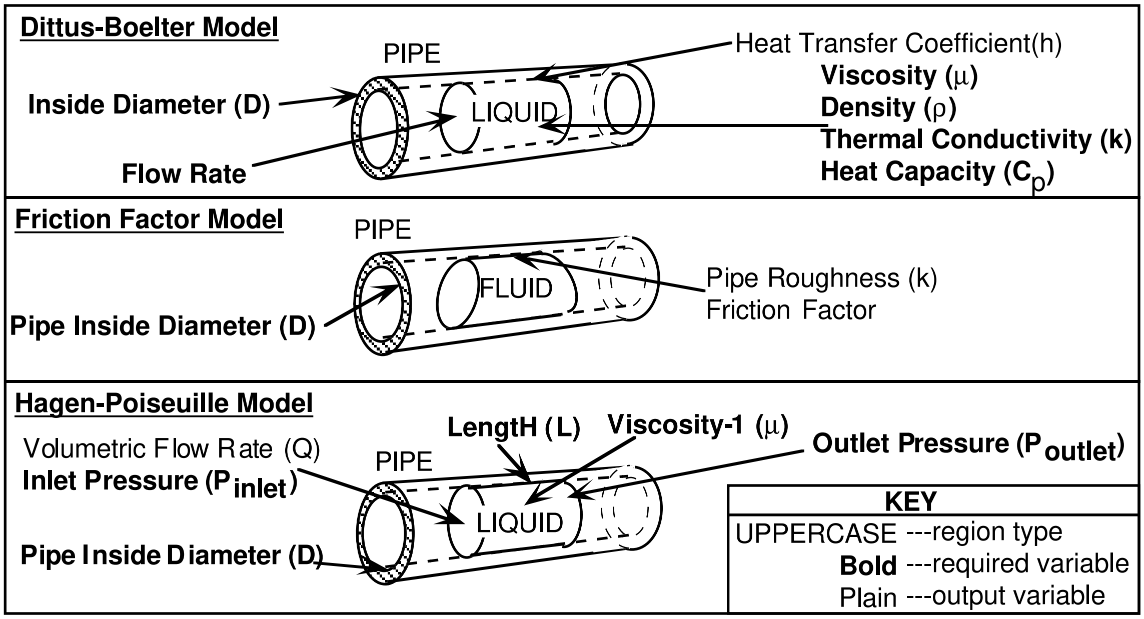
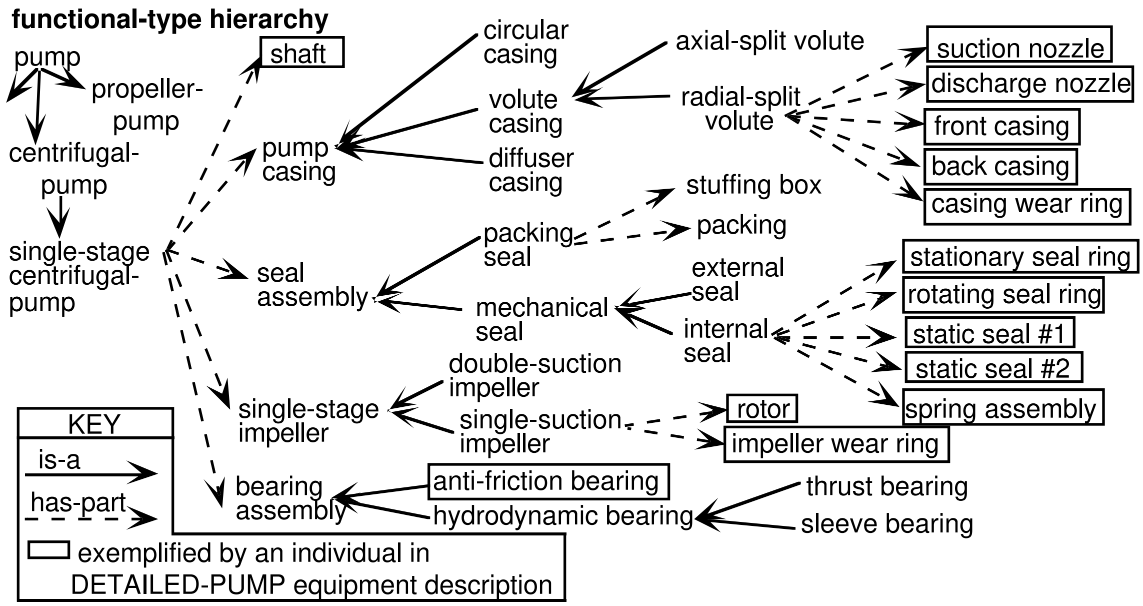


Figure 7.6: Pictorial representation of the model maps for the 3 fluid flow models, showing individuals, required parameters, output parameters, and parameter locations.

fluids (liquids or gases) and the region of fluid within the pipe does not have to have parallel ends. The model requires pipe roughness as an input parameter, and thus we restrict the individual to reside within one pipe. Appendix C.7 gives further details of this model and its description. Appendix D.16 describes the matching.

The Hagen-Poiseuille Model. This model calculates the volumetric flow rate in a pipe from the fluid viscosity and the pressure difference across a cylindrical section of fluid inside a pipe. The Hagen-Poiseuille model requires a cylinder of liquid, just like the Dittus-Boelter model, but the cylinder is allowed to cross pipe boundaries into adjacent pipes. This model exposed the need for a pattern matching capability to allow us to specify any number of pipes connected in series. However, we have not implemented pattern matching, and thus we treat the Hagen-Poiseuille model as if it were restricted to a single pipe. The restricted model still has significant differences from the Dittus-Boelter model's individual because it requires different parameters describing the individual. Appendix C.8 gives details of this model and its description. Appendix D.17 describes the matching.

Figure 7.6 summarizes these 3 models pictorially, showing their individuals, required parameters, parameters calculated, and locations of parameters. Figure 7.7 shows the portions of the PIPES-1 equipment description that were reconfigured and matched for each of the models. This figure also shows the starting parameter for each match. This series demonstrates enhanced independence of model descriptions from equipment descriptions for non-component models and thus enhanced extensibility over previous approaches, because one is not required to tailor model descriptions to the exact individuals in the current equipment description. Notice that the individual matched by Hagen-Poiseuille is a portion of the individual matched by Dittus-Boelter, illustrating that no single individuation of equipment will be sufficient for these kinds of models. Furthermore, each model addition enhanced the system's capabilities, enabling it to calculate a parameter that it could not



TYPES of Individuals Matched/Reconfigured by Each Pump Model

NPSH Model	Wear Ring Model	Hydraulic Horsepower	Hypothetical Model
SINGE-STAGE-CENTRIFUGAL-PUMP SINGLE-SUCTION-IMPELLER pump-speed LIQUID, WATER temperature LIQUID-INLET-PORT flow-rate inlet-pressure	CENTRIFUGAL-PUMP CASING-WEAR-RING c-wear-ring-OD IMPELLER-WEAR-RING i-wear-ring-ID FLUID MOTOR power-1 power-2 **Key** INDIVIDUAL TYPE required parameter *Describes* link	PUMP FLUID specific-gravity FLUID-INLET-PORT inlet-ID flow-rate suction-pressure FLUID-OUTLET-PORT outlet-ID discharge-pressure	SINGLE-STAGE-CENTRIFUGAL-PUMP PUMP-CASING IMPELLER pump-speed LIQUID, WATER temperature LIQUID-INLET-PORT flow-rate inlet-pressure

Figure 7.5: Functional type hierarchy with boxed types indicating individuals that exist in DETAILED-PUMP equipment description and the types of individuals that each pump model reconfigured and matched.

The Dittus-Boelter Model. The Dittus-Boelter model requires a cylindrical region of liquid within a pipe. It was previously described in Section 7.2.1, and Appendix C.2 gives complete details of its model description. The matching is described in Appendix D.3.

The Friction Factor Model. The friction factor model calculates the Fanning friction factor from the pipe diamater and roughness. The correlation is only valid under particular flow conditions, though, which require the fluid to be completely filling the pipe, similar to the Dittus-Boelter model. Unlike the Dittus-Boelter model, the Friction Factor model applies to

The Hydraulic Horsepower Model. Hydraulic horsepower is the theoretical minimum horsepower required to boost the velocity and pressure of a fluid from from the inlet conditions to the outlet conditions of a pump. This minimum horsepower does not take into account any losses within the pump or motor. This model applies to all pumps, not just centrifugal pumps, and thus has an even wider range of applicability than the previous two models. It does not require any of the parts of the pump to be explicitly represented as individuals. Appendix C.5 provides the details for the Hydraulic Horsepower model description. Appendix D.14 provides additional detail for this matching, including inputs and outputs for each of the four stages of the matching algorithm, side effects, and a summary of reconfigurations tried.

The Hypothetical Model. This model is not a real model, but simply a model description that we built to test the system's capability to reconfigure to two different levels of detail within the same piece of equipment. Models clearly can require more than one level of detail, as the NPSH and Wear Ring models do, but our equipment description did not provide the opportunity to reconfigure at two levels. The Hypothetical model has all the same individuals and required parameters as the NPSH model, except that it does not require an impeller and does require a casing. The casing is represented as 5 separate individuals in DETAILED-PUMP, P76-Back-Casing, P76-Front-Casing, P76-Suction-Nozzle, P76-Discharge-Nozzle, and P76-Casing-Wear-Ring. Appendix C.6 provides the details of this model description. Appendix D.15 provides additional detail for this matching, including inputs and outputs for each of the four stages of the matching algorithm, side effects, and a summary of reconfigurations tried.

Figure 7.5 summarizes the pump extensibility series. We show the relevant portion of the `functional` type hierarchy including both subparts and subclasses. In the hierarchy shown, boxes indicate the types which have exemplifying individuals within the DETAILED-PUMP equipment description. This portion of the hierarchy is used by the part/whole reconfiguration algorithm to find the necesary parts for the reconfigurations in this series. Below the hierarchy, a table lists the *types* of individuals required in each of the models along with required parameters. These models require different `functional` types of pumps, different subparts, and different `material` and `phase` types on the internal fluid. All of these models apply to the situation in DETAILED-PUMP and in all cases, the system correctly reconfigures to make the individuals of the required types and proceeds to match. This series demonstrates enhanced independence of model descriptions from equipment descriptions and thus enhanced extensibility over previous approaches, because one does not need to strictly tailor model descriptions to use the exact terms as in the current equipment description. Furthermore, each model addition enhances the system's capabilities, enabling it to calculate a parameter that it could not previously calculate.

7.3.2 Adding Models: The Fluid Flow Series

The fluid flow extensibility series uses the PIPES-1 equipment description (Appendix B.2) and matches 3 different models to the flow through the pipes. All these models are non-component models in that at least one of their individuals is intensive and could be matched multiple ways in a given phsyical situation. All involve fluid flow within pipes, but the requirements on the fluid individual and the parameters are somewhat different, forcing different reconfigurations to be made.

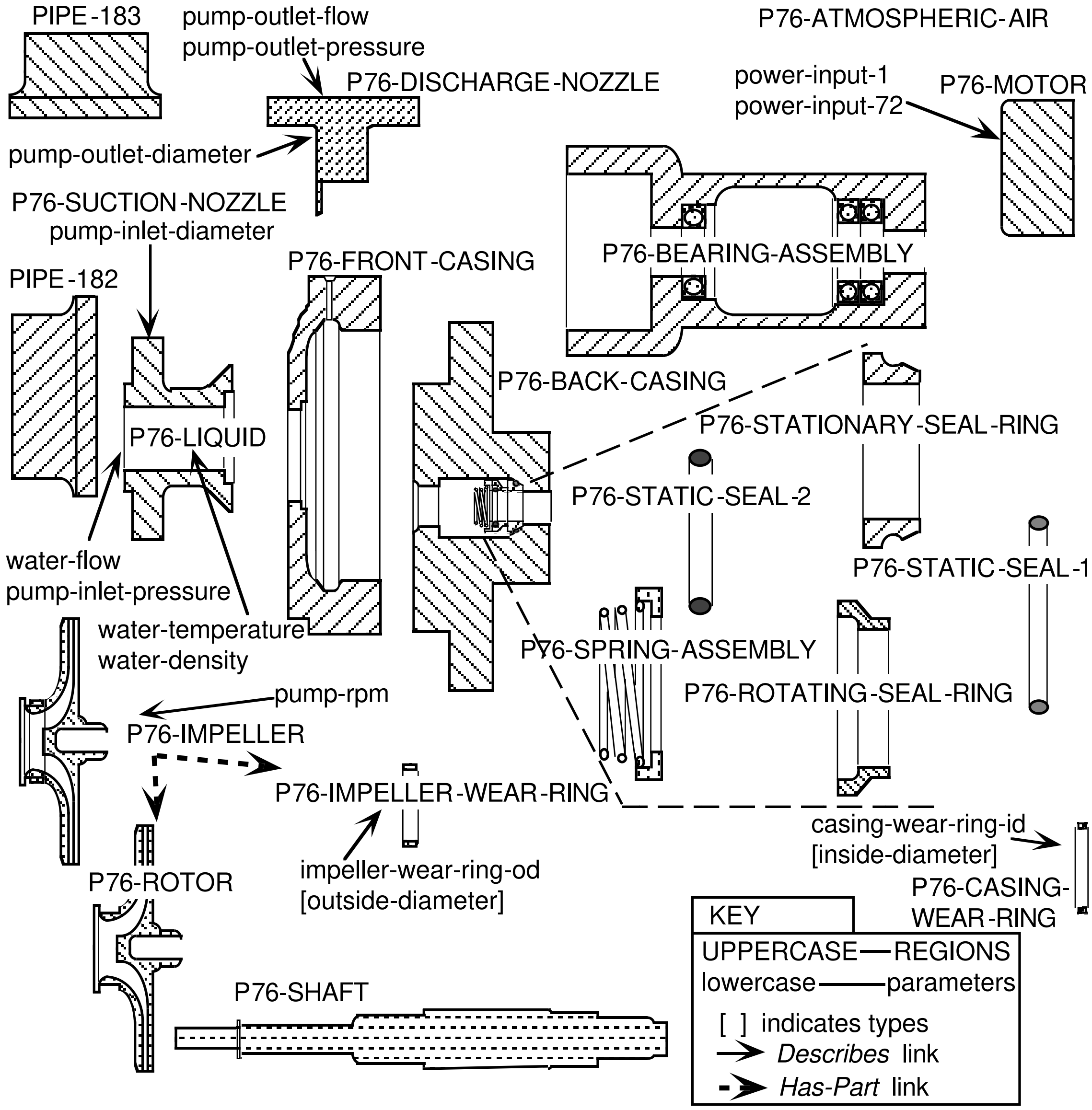


Figure 7.4: Individuals in the DETAILED-PUMP equipment description shown as disembodied pieces.

impeller wear ring and casing wear ring in a centrifugal pump as a function of increased power consumption. As the pump wears and the clearance becomes larger, the power required to generate the same flow rate at the same pressure will increase. This model applies to a larger class of pumps than the NPSH model because it doesn't require a single stage pump, nor does it require a single suction impeller. It requires different individuals as well because the casing and impeller wear rings must be explicitly represented. This model is also less restrictive in that it applies to any fluid (liquid or gas), not just liquid water. Appendix C.4 gives the full model description for the Wear Ring model. Appendix D.13 provides additional detail for this matching, including inputs and outputs for each of the four stages of the matching algorithm, side effects, and a summary of reconfigurations tried.

incorporated. This task may be prohibitive for large models. Furthermore, it is unlikely that any single represenation is appropriate to handle a large number of engineering models, because these models use a large variety of calculation methods and different ways of abstracting the physical world.

7.3.1 Adding Models: The Pump Series

We provide two extensibility series paralleling the re-usability series, one for component models and one for non-component models. In each series, the system starts out with an equipment description in place and models are added one by one. The models require different parameters and individuals which may require different reconfigurations to match. The models also calculate different values. Thus as we add models, the system's capabilities increase. These two series demonstrate that model descriptions which specify their requirements in forms different than that expressed in the equipment description can be matched correctly.

The pump extensibility series uses the DETAILED-PUMP equipment description to match four different models. This equipment description represents 14 different parts of a single-stage centrifugal pump as separate individuals. Figure 7.4 shows the individuals as disembodied pieces to make explicit what is present in the equipment description. DETAILED-PUMP also represents two pipes attached to the inlet and outlet of the pumps as individuals. The liquid inside the pump is a single individual. The liquid regions inside each of the attached pipes are also represented. In addition, an individual that is an aggregate of some of the other individuals is represented explicitly. P76-Impeller is composed of P76-Rotor and P76-Impeller-wear-ring. These parts are represented at two levels of detail to show that the approach does not restrict equipment descriptions to only one representation of each portion of space. More details of the DETAILED-PUMP equipment description are provided in Appendix B.11.

The models in this series vary in the classification of pump they require, the levels of detail, and the parameters that they require and calculate. They also vary in their requirements on the fluid in the pump.

The NPSH Model. The NPSH model calculates minimum net positive suction head (NPSH) required for a single stage centrifugal pump pumping liquid water. The minimum NPSH requried can be thought of approximately as a minimum pressure required at the pump inlet to ensure proper operation. In addition to the pump, this model also requires the impeller to exist as a separate individual, because the speed of the impelller is a required parameter. The impeller also must be a *single-suction* impeller, which is one of several impeller types that the single stage centrifugal pump may have. This model clearly shows the need to represent multiple levels of detail. If we had to rely only on the pump type, we would be forced to invent arbitrary types like `single stage centrifugal pump with single suction impeller` that are not used in the domain. Appendix C.3 provides full details for the NPSH model description. Appendix D.12 provides additional detail for this matching, including inputs and outputs for each of the four stages of the matching algorithm, side effects, and a summary of reconfigurations tried.

The Wear Ring Model. The Wear Ring model calculates the change in clearance between the

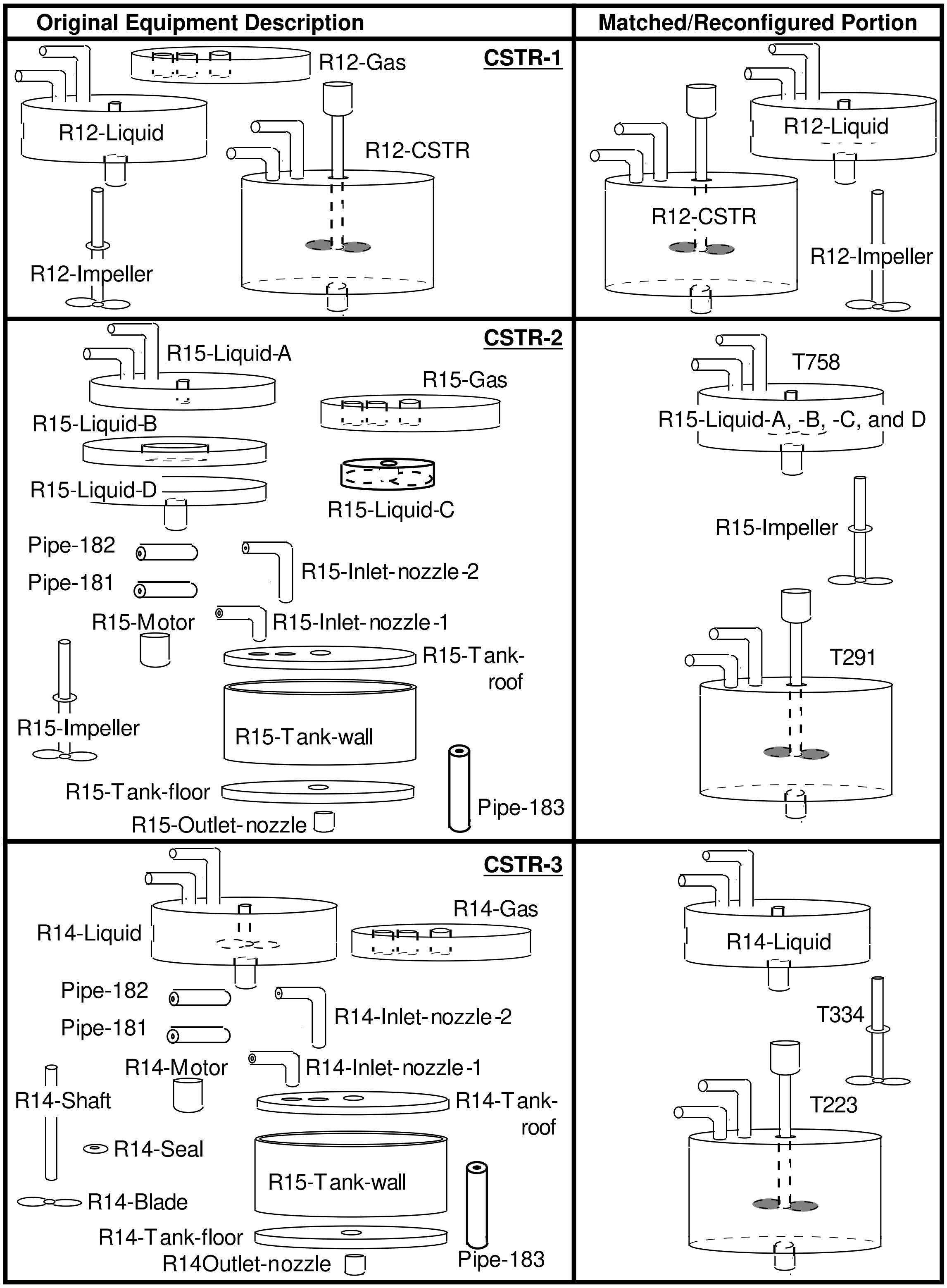


Figure 7.3: Equipment descriptions and reconfigurations in the Dalle Molle re-usability series.

requirement of the Dalle Molle model that is not necessarily implied by the functional types is that the outlet nozzle be installed in the bottom of the tank, rather than through the side wall. This requirement is a result of the model's assumption that outlet flow is proportional to height of liquid in the tank. (The location of the outlet is not sufficient to meet this condition, though, and an approximation condition checks numerical values in addition to this spatial relation.) This requirement is specified as a shared surface (`ports`) between the outlet nozzle and tank floor in the model map. Appendix C.1 gives the full model description for this model, including the model map and approximation conditions.

The Dalle Molle re-usability series applies the Dalle Molle model to the three different equipment descriptions pictured in Figure 7.3, where the individuals are shown as disembodied pieces. In the equipment description, all individuals are assembled in contiguous space. These three equipment descriptions as implemented in the semantic net geometric representation range in size from 92 K to 139 K and are described in Appendix B. The figure also shows the reconfigurations made and the matches found. The part/whole reconfiguration algorithm generates the required reactor from the assembled parts in both the second and third cases. The part/whole reconfigurer also makes the impeller in the third case, because only its parts were represented. The second case employs intensive reconfiguration to generate the liquid individual inside the reactor. The model has to find an individual that includes all the liquid within the tank. Appendix D provides additional detail for each of these 3 matches, including inputs and outputs for each of the four stages of the matching algorithm, side effects, and a summary of reconfigurations tried. This series demonstrates for component models how run-time matching and reconfiguration decouples the equipment descriptions from the model details. This added independence increases the re-usability of such systems by allowing the equipment descriptions to be built without tailoring details to each model in the system.

7.3 Demonstrations of Extensibility

We address three dimensions of extensibility. First, by showing that a series of models may be added without their requirements being specified in the exact terms present in the current equipment description, we show increased ease of incorporating models. This independence of model descriptions from equipment descriptions allows models to be described with the least detail possible to state their requirements. One also does not have to identify every place in the current equipment description where new models should be match, and tailor the description to have the exact same form(s) as all the places where it may be applied. Second, we show that the algorithms allow the type hierarchies to be changed without affecting the ability to match or reconfigure. Type hierarchies provide the vocabulary for describing models and equipment. If the system were restricted to the particular details of our hierarchies, the system would not be readily extensible. Third, we demonstrate that models using different internal calculation methods can be used by this system. Previous approaches require a uniform representation for models, usually because they somehow access the model internals. Such a requirement forces one to re-write any model before it can be edges with the two end surfaces.) The merging has wandered beyond the end of the pipe, violating some other requirements that had been met before merging started. The matcher also merges in the other direction with region WE and finds that the match is again degraded. It then terminates without finding matches because no other regions can be merged.

PIPES-4: This case tests the requirement that the fluid continue beyond the cylindrical individual identified. In this case, a cap has been placed on the left end of Pipe-1, perhaps to block the pipe for maintenance of other equipment. A flow sensor still reports a flow rate at that end of the pipe, even though the value is 0. Starting with the goal of finding effects of a change in heat capacity in Region B, the matcher merges A and B, finding the flow rate parameter it needs. (The value is not tested until after the match.) However, the requirement that the end surface of the cylinder be adjacent to more liquid is not met. The matcher attempts to merge B and C as well as B, C, and D when the termination heuristic stops it from attempting any further merges. The merging has wandered outside the pipe and degraded the previously matching portion of the model map.

PIPES-5: This case tests the requirement that the liquid cylinder not extend past the end of a pipe. Starting with the goal of finding effects of the heat capacity in Region C, the matcher finds the only flow rate available (a required parameter) at the end of Region C. The inability to match the shared edge between the liquid's surface to the pipe and an end surface triggers an intensive reconfiguration. Splitting the region, the matcher eventually finds the portion which is wholly contained inside Pipe-2 and which also has the flow rate parameter.

PIPES-6: This case ensures that the system find liquid within a pipe, and not some other flow conduit. Given the goal of finding effects of the thermal conductivity in Region B, the matcher finds the required flow rate at the left end of Region B. The system finds it necessary to split the region, because it extends over the ends of a containing pipe. Then it finds a region wholly contained within Elbow-1 and having the required flow rate. No match is found, though, because the containing conduit, Elbow-1, is not of functional type `pipe`.

Appendix D contains the details of each of these test runs, including all inputs and outputs from each of the four stages of the matching system, as well as the side effects and a summary of the reconfigurations tried.

This series demonstrates a significantly increased independence of the equipment description from the model details over previous approaches. This independence enhances the re-usability by allowing equipment descriptions to be built without tailoring to the exact details of each model.

7.2.2 The Dalle Molle Series

The Dalle Molle re-usability series uses the Dalle Molle model, which is a qualitative simulation of hot and cold liquid flows mixing in a stirred tank reactor. It simulates the temperature change and the change in liquid level in the tank over time after a step increase or decrease in either the hot inlet stream or cold inlet stream. Even though it involves liquid inside the reactor, it is a component model because the liquid inidividual must include all the liquid in the reactor. Thus for any particular physical situation, there is only one such individual no matter what parameters may be available. The Dalle Molle model requires that not only a whole stirred tank reactor be present, but also that its impeller (a subpart) be explicitly represented, thus requiring the ability to represent and match multiple levels of detail within the same match. A spatial relation

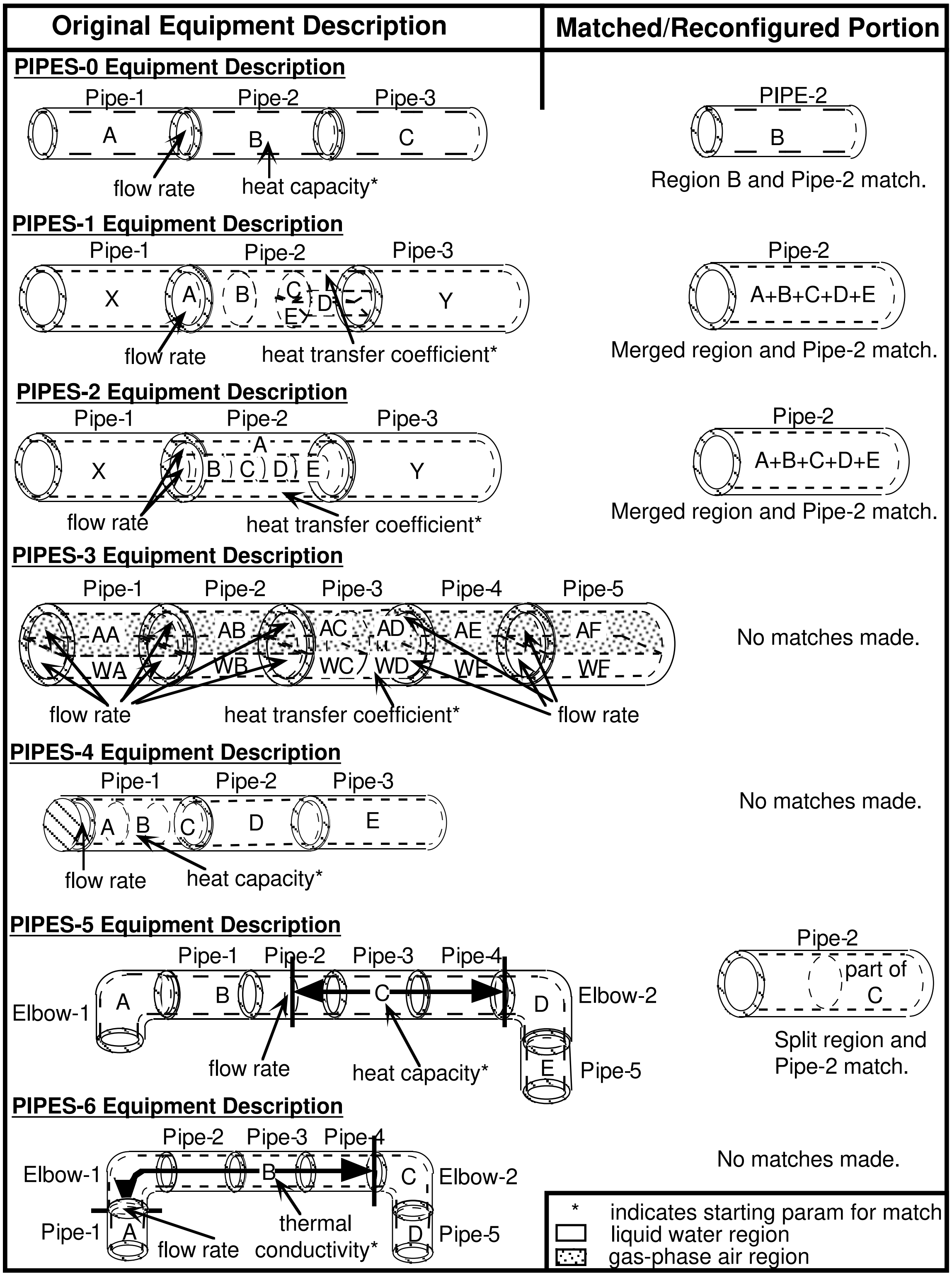
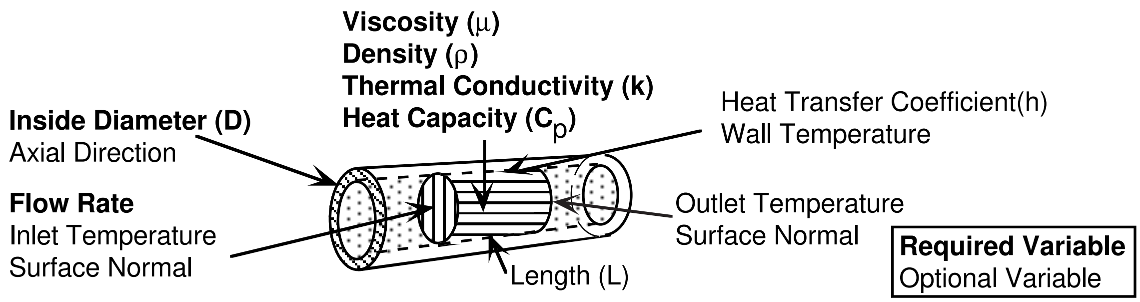


Figure 7.2: Equipment descriptions and reconfigurations in the Dittus-Boelter re-usability series.



Model Requirement	Equipment Description which Demonstrates
parameters describe places in space.......	PIPES-1
nothing else is in this space.....................	PIPES-2
fills cross-section......................................	PIPES-3
fluid continues beyond individual.............	PIPES-4
doesn't extend past pipe end..................	PIPES-5
liquid inside PIPE.....................................	PIPES-6

Figure 7.1: Some non-numeric requirements of the Dittus-Boelter model.

research.

The requirements listed in Figure 7.1 are each tested by one of the equipment descriptions in the series. The equipment descriptions are shown as pictures in Figure 7.2 along with the reconfigured portions that matched. This figure also shows the starting parameter (*) for each attempted match. The pictures in Figure 7.2 are only drawings of what we have built using the semantic net geometric representation. These equipment descriptions range in size from 53 K to 195 K and are described in more detail in Appendix B. We believe that given the requirements for the model, it will be obvious to all observers (engineers and non-engineers) whether the system is exhibiting the desired behavior with respect to each requirement.

PIPES-1: This equipment description tests the ability to identify locations and spatial extents of parameters. The match starts in Region C with the goal of matching a model that will calculate a heat transfer coefficient at the wall. It ultimately merges A, B, C, D, and E to get a correct match. The driving force behind this merge is the required flow rate parameter which must describe one end of the cylindrical region. Note that the requirement for a full cylinder could have been met by merging only C, D, and E.

PIPES-2: This equipment description tests the requirement that nothing else be present inside the cylinder of fluid, a requirement which is expressed through negative sets. Starting from Region A, the matcher finds the interior regions, B, C, D, and E, which happen to be mergable with A in this case. By reconfiguring, the system correctly finds a match. If these interior regions had been pipes or portions of a wire, for example, the merging would not have been possible and no match would be found.

PIPES-3: This equipment description tests the requirement that the pipe's cross section be filled with liquid. Liquid is flowing through only the bottom half of each pipe. The matcher, being unsophisticated geometrically, tries a few reconfigurations, starting with the initial parameter's region, Region WD, merging WC and WB, at which point the termination heuristic finds the match is worse than when merging started. (The surface to the pipe no longer shares its number of possible situations the physical world may present, it is not possible to show that any set of methods or characteristics is sufficient.

7.2 Demonstrations of Re-usability

The demonstrations of re-usability involve a series of replacements of the system's current equipment description with a new equipment description. Each of the equipment descriptions built is large enough so that the matcher and reconfiguring algorithms did not run into the boundaries. Through the series of replacements, the models in the matching system and the matching algorithms themselves remain the same. These demonstrations mimic what would be done to re-use such a system for some different physical equipment. The intent is to show that the system can successfully match a model to a number of equipment descriptions whose forms vary from the form required by the model. If the system can make the correct forms, then the user does not have to tailor the equipment description to the form of this model and to the forms of all the potentially large number of models that such a system could contain.

7.2.1 The Dittus-Boelter Series

The Dittus-Boelter series is a set of matchings of the Dittus-Boelter model to 7 different equipment descriptions. This series is longer than the others because it is also intended to show that we could express the requirements for a non-component model and that the matcher could identify and correctly match examples where different requirements were the determining factor in the match. Some of the cases in this series provide situations that should not be matched under any reconfigurations, to show that the matcher correctly recognizes a requirement not being met. All of the non-matching cases are "near" matches in the sense that they have only one characteristic that is different than the model requires. If the matcher can detect these subtle differences it will also be able to detect large differences.

Figure 7.1 shows the physical situation and requirements of the Dittus-Boelter model. The cylindrical liquid region which forms the individual of interest is conceptually cut out of the flowing stream of liquid. The individual does *not* have to fill the pipe from end to end, but cannot extend beyond the ends. This variability makes the Dittus-Boelter model a non-component model. Appendix C.2 gives the full model description for the Dittus-Boelter model, including the model map and the approximation conditions.

The Dittus Boelter re-usabililty series focuses on the model requirements that are not numerical. Figure 7.1 lists these requirements. They also are not purely geometrical. They involve combinations of material, phase, and functional types occupying space and having particular relationships to each other. Besides the listed requirements, all of the models implemented require that parameters exist at particular locations and have particular spatial extents. These requirements often are not stated explicitly in engineering textbooks because they are part of the background knowledge of engineers. The ability to express and check these requirements is one of the original contributions of this fully investigated efficiency issues within the implementations. The system runs in Common Lisp on lisp machines, which are known to be slow machines.[1] We also found a serious implementation inefficiency in the BB1 blackboard system upon which our matcher is built. (This inefficiency has been removed from a new version of BB1.) The cases all run fast enough (ranging from less than 1 minute to 4 hours) for the purposes of this research. However, before such a system could be put to practical use, efficiency should be improved.

The limitations of this approach arise from our fundamental assumptions, the selection of mechanisms implemented and tested, and implementation assumptions.

Fundamental Assumptions. The assumptions on which we based this approach require that the models apply to contiguous portions of space (Contiguous Space Assumption) and that individuals are large enough that macroscopic properties, such as temperature and pressure, are meaningful (Macroscopic Properties Assumption.) Section 3.4.2 described these two assumptions.

Selected Implementation. To limit the scope of the work, we selectively implement some of the methods from those we observed.

1. Geometric reasoning (merging and splitting) is limited to cases that can be described using a semi-quantitative semantic net.
2. Parameter transformations are limited by the geometric representation.
3. We implement simple and negative matching, but not pattern matching.
4. We implement intensive and part/whole reconfiguration, but not extensive reconfiguration.
5. The cases implemented do not exhibit characteristics which are known to be important for some kinds of engineering models, including different properties of the connections between individuals (such as abutting vs. rigidly connected) and changes in spatial relationships between individuals that cause `ports` (surfaces) to appear or disappear.
6. Efficiency of the current implementation is optimized only enough to carry out the research, and run times on the current platform are probably not small enough for practical use in a diagnosis system.

Implementation Assumptions.

1. Intensive reconfiguration uses a search termination heuristic that is a form of hill-climbing, and thus may find only local optima.
2. Part/whole reconfiguration can only reconfigure equipment which have parts that all share at least one surface with some other part. It can also only detect that a set of parts will not form a correctly functioning whole if the fault is manifested in particular ways. (See Section 6.5.2 for details.)

We identify a particular set of model and equipment characteristics and a set of matching and reconfiguration methods required, and implement and test a subset of the identified set. We do not claim to have identified all the characteristics and methods. When working with the infinite

[1] Lisp machines were more widely used in research when we began implementions. We chose to complete the research on the lisp machine rather than porting to the now more commonly used unix machines or C++ language to avoid delaying the research.

match a single equipment description. The models in the series require different reconfigurations to meet their specifications. We provide a non-component model series and a component model series, paralleling the two re-usability demonstrations. One series matches 3 non-component models, the Dittus-Boelter model (Appendix C.2), the Hagen-Poiseuille model (Appendix C.8), and a friction factor model (Appendix C.7). The Hagen-Poiseuille model implements the Hagen-Poiseuille flow equation which calculates volumetric flow rate for liquids in pipes. The friction factor model calculates Fanning friction factor for fluid flow in pipes. They each have somewhat different requirements for parameters and individuals which causes them to reconfigure the single equipment description in three different ways. The second extensibility series involves 3 different pump models, all of which are component models. The system had to reconfigure for 6 of these 7 models, and did so successfully in each case.

Ability to extend also requires ability to extend the vocabulary. The matching and reconfiguration algorithms use only the partial order within the type hierarchies (except for the top level classifications such as `region`, `port`, `edge`, etc.), and thus can be extended with new types as needed. As we added models, we were forced to add new types to the hierarchies, both as leaf nodes and as internal nodes. The matcher handled all additions successfully.

Since engineering models do not all use the same calculation methods and since many models are already implemented, we design our approach to allow models using arbitrary internal methods to be included in the system. By design, our system does not access the internal representations of the model, only providing inputs to models and taking outputs. As a demonstration, we include several different kinds of internal calculation methods. We include models using various equation forms, a table interpolation method, a numerical method (not used in the demonstration series), and a large qualitative simulator. The qualitative simulator is QSIM [42] and it is internal to the Dalle Molle model [11]. It is a large system, about 1.2 M of Common Lisp code. This model was built by other people and we do not know its internal data structures or algorithms. We have added this model to our model-matching system as our primary demonstration that various models can work together. The matching system successfully reconfigures, matches, and calculates values with this model.

Even though our primary research goals do not include efficiency, we analyze the methods and monitor the implemented examples to collect efficiency data. We show that the matching problem, as we have formulated it using semantic nets, is provably NP-complete. In the worst case, an exponential number (in the number of objects in the semantic net) of partial matches could be examined to find the correct match. However, the worst case is probably significantly different than what we would usually find in real matching situations. This difference results from not being able to guarantee that we have any distinguishing information (types and link types) within the net. Most semantic nets will have distinguishing types and links that eliminate a lot of the partial matches we would have to consider at each step in the matching. We monitor the number of partial matches examined in all of our cases and see numbers that are very small compared to the exponential worst case. The worst case analysis does not take into account reconfiguration methods, which may force us to reconfigure and rematch a number of times. We also have not

Chapter 7

Evaluation of Approach

7.1 Summary

Our motivation for embarking on this research is to find ways that modeling of physical systems and therefore model-based reasoning systems, especially diagnosis systems, can be made more readily re-usable and extensible. Our goals are to identify characteristics of models, identify characteristics of equipment, and develop methods to match models to equipment at run-time that allow models and equipment to be described independently from each other. If these tasks are independent we can significantly reduce the work required to add models and to re-use the system on a different piece of equipment. Clearly the problem of independence and matching engineering models to equipment is a large one, and we view the characteristics and methods we identified as a subset of those required to solve the general problem. To evaluate the set of characteristics identified and the matching methods developed, we demonstrate re-usability and extensibility.

We implement a series of examples to provide two separate demonstrations of re-usability and two demonstrations of extensibility. The re-usability demonstrations consist of giving the matching system a series of different equipment descriptions, and having the system reconfigure (if necessary) and match the same model correctly in all the cases. The equipment descriptions contain similar pieces of equipment to which the model can be applied, but the descriptions take a number of different forms. We have two re-usabililty series, one for a *non-component* model and one for a *component* model. *Non-component* models are those models, like the ideal gas law, that have at least one individual whose boundaries may be set anywhere through a region of space. *Component* models have only individuals with boundaries that occur at discrete locations. The non-component model used for re-usability demonstrations is the Dittus-Boelter model, which is described in Appendix C.2. One of its individuals is a cylinder of space inside a pipe through which liquid is flowing. The ends of the cylinder may be at any locations within the pipe. The series consists of 7 different equipment descriptions of fluid flow in pipes. We also provide a re-usability series for a model of hot and cold liquid mixing in a stirred tank reactor. This model was implemented by David Dalle Molle [11] and referred to in this work as the *Dalle Molle model*. The individuals in this model can be chosen exactly one way for any physical situation. Three different equipment descriptions are built for matching of the Dalle Molle model. The matching and reconfiguration algorithms correctly match and reconfigure for both of these series.

The extensibility demonstrations consist of adding a series of different models to the system that two cases require that some aggregate, in one case a reactor and in the other a pump, be explicitly represented as well as some subpart. The reactor model requires an impeller and the pump model requires a casing. The system correctly generates the aggregate and subpart in both cases from even smaller subparts. In another case, D.10, both part/whole and intensive reconfigurations are used. As for the intensive examples, all of these examples find the correct reconfigurations by looking no further than the region of failure or the adjacent regions.

The important finding in building these test cases is that aggregates must be made spatially. By merging the subparts spatially, we automatically maintain the proper spatial relationship of the aggregate to the surrounding equipment and likewise retain parameters for the aggregate. The parameters involved in models of functional type components such as pumps and reactors often describe regions adjacent to the equipment, and not the equipment itself. An example is the pressure of the fluid inside the pump entrance. In previous work the geometric space occupied by the equipment has been mostly ignored. Physical entities act on each other through space. Parameters all have particular locations and spatial extents. Our view is that by keeping the spatial information in the representation, we can automatically manage parameters and relations between the equipment of interest and the equipment around it without predetermining levels of detail.

6.5.4 Limitations

The limitations of the part/whole reconfiguration algorithms derive from the Adjacency Assumption and the Manifested Failures Assumption. The Adjacency Assumption states that every subpart of some piece of equipment is assumed to share a surface with some other part of the equipment. This is a reasonable assumption for pumps and reactors, the types of equipment we model in this thesis, but may not be a good assumption for all types of equipment we might encounter. The Manifested Failures assumption is likely to present more difficulty in the kinds of equipment we model because all the components of a piece of equipment may be present and may meet the Adjacency Assumption, and each component would be in working order, but the aggregate would not be in working order because the relations between the parts are not correct. Part/whole reconfiguration would not be able to distinguish this non-working aggregate from a correctly assembled piece of equipment. We expect the impact of this limitation to be relatively small, though, because equipment descriptions would most likely be built for working equipment, and only a few faults would be incorporated by a diagnosis sytem as it reasoned about the equipment. The impact is also reduced by the fact that some faults would be detected by the part/whole algorithm, those that meet the Manifested Failures Assumption. This problem could benefit from future work.

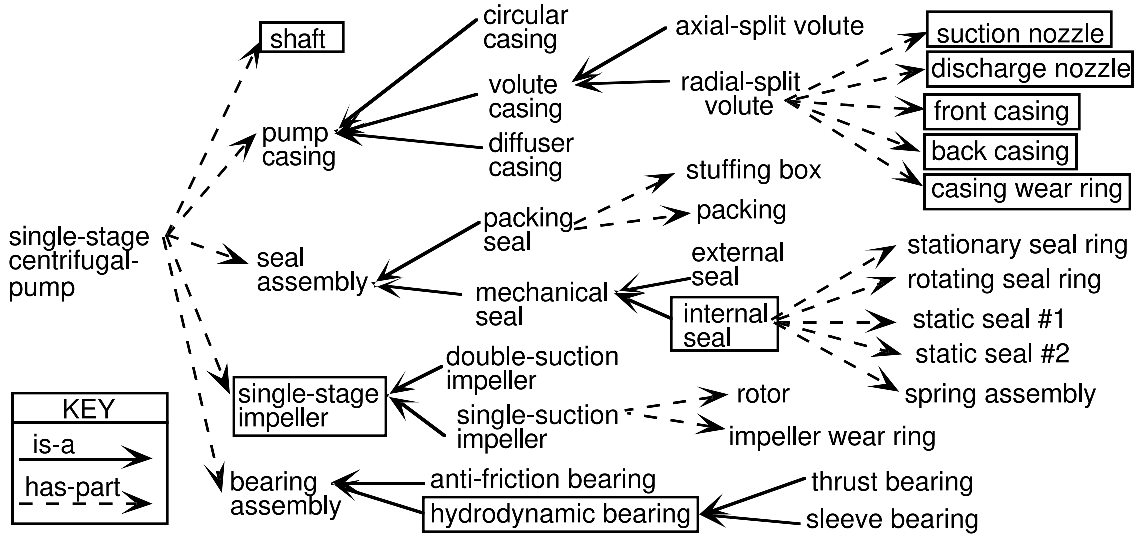


Figure 6.4: A portion of a functional type hierarchy showing a legal set of parts for reconfiguring to a single-stage-centrifugal-pump.

not in working order, the defect will be described as either a missing subpart, a subpart whose proper `functional` type has been removed, or as a completely disconnected part.

The first assumption allows the part/whole merger to find the parts by checking the space adjacent to `s-region`. The second assumption allows it to find the parts without checking the nature of their interconnections. Since we expect that most of the equipment description will depict working equipment, this assumption will be correct in most cases. Furthermore, since diagnosis systems use the `is-a` relation (which corresponds to `exemplifies` in our system) to signify not only type but that the individual is behaving correctly as specified by that type, faulty components would typically be depicted as having no functional type or a different functional type. To give users the flexibility to arbitrarily divide equipment components into spatial pieces (much like they might divide large liquid regions into arbitrary pieces), the part/whole merger also checks the spatial superregions of any equipment regions when looking for the subparts to reconfigure.

6.5.3 Implementations

Figure 6.3 shows the flow of control to and from the part/whole reconfiguration functions. It is called by `do-simple-match` when a failure occurs and the triggering conditions for `part-whole-reconfig` are met. `Part-whole-reconfig` calls `merge`, the same function as used by the intensive reconfiguration algorithm, which will call the same type and parameter transforms. `Part-whole-reconfig` also must call `do-simple-match` via $\xrightarrow{**}$ to do its rematch work. Just as intensive reconfigurations disturb previously matched portions and require rematching, so do part/whole reconfigurations.

The test cases that use the part/whole reconfiguration algorithm are Cases D.10–D.15, which are summarized in Table 6.1. More detail is provided in the corresponding appendices. Two of the cases, D.11 and D.15, require two separate part/whole reconfigurations. The models in these

2. The model has a coalescible region R that contains `current-m-obj` or is adjacent to `current-m-obj`'s region.

3. R has not yet been matched.

6.5.2 The Part/Whole Merging Algorithm

The part/whole merger finds all the subparts of the coalescible type in the equipment description, merges them applying the type and parameter transforms, and assigns the resulting region to have the coalescible type as its `functional` type.

The coalescible type, the `functional` type of the coalescible model region, is given to the part/whole merger by the triggering mechanism. The part/whole merger relies on the `functional` type hierarchy to define what subparts the coalescible type has through `has-part` links from the coalescible type. We assume that the set of subparts in the `functional` type hierarchy is complete. The part/whole merger then must find a complete set of subparts within the equipment description. Finding a complete set of subparts is complicated by the fact that we allow subparts to be defined by a tree of `has-part` links that extend from the coalescible type we are trying to generate. The algorithm then finds a complete set of subparts, finding sibling subparts at whatever level they might be represented in the equipment description. There is no restriction on equipment descriptions to represent everything at the same level of detail.

We define the condition of all subparts having a match recursively. Here we use the predicate `matched(x)` to mean that for the type object, `x`, an equipment region that is of that type or a subclass has been found.

$$AllSubpartsMatch(T) \equiv$$

$$matched(T) \vee (\forall t_i(\mathtt{HasPart(T,t_i)} \rightarrow (matched(t_i) \vee AllSubpartsMatch(t_i))))$$

An example is shown in Figure 6.4. This is a portion of the `functional` type hierarchy used in several cases implemented for this thesis (Appendices D.12, D.13, D.14, and D.15). If the part/whole merger was attempting to generate a `single-stage centrifugal-pump`, then one way that it could create it is by finding equipment regions that match all the boxed subparts and merging them.

Besides knowing what to look for, the merger must also know where to look. We start with any equipment region that is near enough to the point of failure that a reconfiguration could fix the failure. Thus we look at equipment regions either at the point of failure or adjacent to the region in which failure occurred. Any nearby region that is a subpart (at any level in the type hierarchy's subpart tree) of the coalescible type becomes a starting point, called `s-region` for the part/whole search. We make the following two assumptions about the nature of subparts manifested in the equipment description.

Adjacency Assumption: All the subparts t_i such that $\mathtt{has-part(T,t_i)}$, share a surface with at least one other subpart t_j, where $t_i \neq t_j$.

Manifested Failures Assumption: If the subparts or aggregate in the equipment description is

6.4.4 Limitations

The limitations in the intensive reconfiguration methods implemented derive from the search heuristic used to terminate merges and splits and also from the simplicity of the geometric representation. The heuristic is included to reduce the number of useless reconfigurations that the system will attempt. However, it can cause correct reconfigurations to be missed. It may be possible to characterize cases where the heuristic should not be used, but this is left for future work. The geometric representation used severely restricts the splitting capability. The system will only attempt to split simply connected regions and can only split those where edges already exist. The geometric representation also prevents us from handling certain extensive parameters, such as `length`, as they should be handled during merges and splits.

6.5 Part/Whole Reconfiguration

Part/whole reconfiguration is a merging of regions that form the `functional` subparts of a `functional` superpart type region required by a model. For example, part/whole reconfiguration can find and merge the casing, shaft, impeller, and other parts to form a single region of type `centrifugal--pump`, when the model calls for such a pump, and the equipment description only explicitly represents pump parts.

6.5.1 Triggering Part/Whole Merging

Part/whole reconfiguration is triggered when a match completely fails, and some model object "near" the point of failure in the model map has a `functional` type with subparts. During the matching process, we move through the objects and links in the model map, building partial matches to the equipment objects. When, on attempting to extend the partial matches for the next model object and link, none of the partial matches have a linked object matching the current model object, we have a complete failure. Let the model object which was being matched at the failure point be called `current-m-obj`. Any type in the `functional` type hierarchy that has subparts, other types linked to it via a `has-part` link, is called *coalescible*. If the model object `current-m-obj` lies in a model region that is coalescible, or in a model region adjacent to a coalescible region, then part/whole reconfiguration may be able to fix the mismatch. We consider only the local region and immediately adjacent regions for the same reasons as in the intensive reconfiguration method. Any nearby coalescible model region also cannot have been matched previously, because part/whole reconfiguration will change the required type (`functional` type), unlike intensive reconfiguration that preserves required types (`material` and `phase` types). By taking a previously matched region for part/whole reconfiguration we would destroy part of a correct match. Thus, the criteria for triggering part/whole reconfiguration are:

1. Complete failure of the match at some model object, `current-m-obj`.

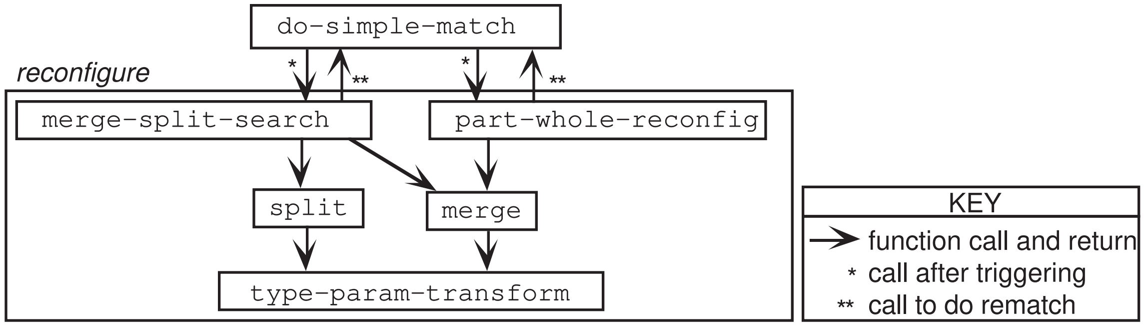


Figure 6.3: Flow of control between the reconfiguration functions.

situation where a parameter could be lost during merging but regained later. For example, consider a model that required both temperature and flow rate at the end of a liquid region. The base region could have the temperature parameter at one end. Several regions further downstream there is a region that has the required flow rate and temperature at the same point. As we merge regions onto the base region, the previously matched end temperature is lost. Our current heuristic terminates the merge before the appropriate parameters are found. In the cases implemented, we observe the first situation, where additional merging would never repair the mismatch, but not the second where we missed correct matches. However, further investigation of the heuristic is left for future work.

6.4.3 Implementations

Figure 6.3 summarizes the implemented reconfiguration algorithms and shows flow of control. The large box labeled `reconfigure` is an expansion of the `reconfigure` box from Figure 5.3. Each arrow represents a *call and return* of a function. Both the `intensive-reconfig` and `part/whole-reconfig` methods use the same underlying spatial reasoning (`merge` and `split` functions) and type/parameter transforms. The reconfiguration functions are initially triggered by `do-simple-match`, as represented by $\xrightarrow{*}$ in Figure 6.3, when it cannot proceed any further with a match. The `intensive-reconfig` and `part/whole-reconfig` functions call `do-simple-match`, as indicated by the $\xrightarrow{**}$ in the figure, to do their rematching work. Rematch calls restrict `do-simple- match` to only match the portion of the model map that is disturbed by the reconfiguration.

Cases D.3, – D.8, D.10, and D.16 all use intensive reconfigurations. These cases are summarized in Table 6.1 and more detail is given for each in Appendix D. The intensive reconfiguration mechanism can be triggered by mismatches occuring at positive objects in the model map, at required parameters, or at negative objects, as is also shown in the table. The reconfiguration mechanism can be triggered more than once during a single model-matching to fix mismatches, as is illustrated especially well by Case D.4, where intensive reconfigurations are done 3 times. Two of the reconfigurations require 5 different merges before a region that solved the mismatch was generated, and one reconfiguration requires 7 merges.

6.4.2 Searching for Regions to Merge and Split

The triggering function for intensive matching identifies a model region and a matching equipment region. This equipment region is the base region which will be split and merged. A search begins to generate a new region through merging and splitting that will match all the previously matched objects and additionally match the `current-m-obj`, the model object that failed to match. For the purposes of merging, the system considers only regions adjacent to the base region that are of the same `material` and `phase` type as required by the intensive model region. We have implemented the intensive reconfiguration algorithm to merge in a breadth first manner, first generating and testing all merges of the base region with adjacent regions (generating level 2 regions), then considering all merges of level 2 regions with all possible adjacent regions. The motivation for this breadth first order is that it prevents the merger from passing by whatever it is looking for. If it were to continue merging a bigger and bigger chain of regions, it may go off into deep space while the parameter or adjacency needed was within 2 regions (2 merges) but was not on the first chain selected. This choice of search order was sufficient for the examples implemented, but no other order was tested.

To control the merge search, we have implemented a termination heuristic. The heuristic terminates further merging onto a region when the region fails to *rematch* the previously successful portion of the match. Without a heuristic, the system could continue to merge regions until it ran out of regions to merge, when no correctly matching regions are generated. Whenever the equipment description is changed through a merge or split, some portion of the partial match that had been made before failure will have to be redone. We use the term *rematching* to refer to the process of identifying the portion of the match that may have been affected by the merge and regenerating that portion of the match for the reconfigured equipment description. The heuristic says that if we cannot rematch the portion of the model map that we had successfully matched prior to reconfiguring, then disqualify the current equipment region from further merging or splitting. Inability to rematch indicates that something is now missing or incorrect that was previously available. This heuristic is a form hill-climbing, where we only consider merges or splits whose resulting regions generate matches that are as good as previous matches.

As with any hill-climbing approach, the system can only find the local optimum. Will this cause us to miss matches? Some kinds of rematch failures clearly will never be fixed by forming bigger regions. For example, some of the model descriptions implemented, such as the Dittus-Boelter model (Appendix C.2) require a fluid region inside a pipe where the ends are **`uniform ports`** that share edges with the cylindrical **`non-uniform port`** adjacent to the pipe wall. If we proceed to merge fluid regions until we are outside the original pipe, no such shared edge will ever exist. The implemented matching case D.5 is an example of this kind of termination. For the pump models (Appendices D.12, D.13, and D.14) where a negative port is specified for the liquid inside the pump (indicating that the liquid) can have no additional ports beyond the exact positive ports specified), merging can only produce additional ports once the merged liquid region extends outside the pump into adjacent piping. Similarly in splitting, if we lose an adjacency (port) to a pipe, we will not reaquire that through splitting because the region will only get smaller. However, one can imagine a

The `merge` and `split` functions call the parameter and type transforms not only when they generate the new region, but also when they merge ports or edges. Parameters may describe ports, edges, or endpoints as well as entire regions. For example, we may know the temperature at the outer surface of a pipe. If this surface were to be merged with some adjacent surface, the transforms are called for the two merging surfaces and return the temperature parameter (if any) to be installed to the new surface.

6.4 Intensive Reconfiguration

Intensive reconfiguration is the merging and splitting of equipment regions to match model regions that are *intensive*, that is, model regions that have only `material` or `phase` types and have no specified values for extensive parameters.

6.4.1 Triggering Intensive Reconfiguration

Intensive reconfiguration is triggered when a match completely fails, and the model object at which the failure occurred is "near" an intensive object in the model. During matching, we move through the objects and links in the model map, finding equipment objects and links that match the current model object and link. During this process, we accumulate a number of correct partial matches. When, on attempting to extend the partial matches for the next model object and link, none of the partial matches have a linked object matching the current model object, we have a complete failure. Let the model object which was being matched when the failure occurred be called `current-m-obj`.

If any of the model objects in the same region as `current-m-obj` or in regions adjacent to `current-m-obj`'s region are intensive regions, then reconfiguration is possible. The `current-m-obj` may be a `region` object but may also be an `endpoint`, `edge`, `port`, `stream`, or `parameter` among others, and we implemented functions that find the appropriate containing regions and adjacent regions for each of these types of objects. If any of the model regions "near" `current-m-obj` are intensive, and they have already been matched, then intensive reconfiguration proceeds, starting with the matched equipment region. Thus, the criteria for triggering intensive reconfiguration are:

1. Complete failure of the match at some model object, `current-m-obj`.
2. The model has an intensive region R that contains `current-m-obj` or is adjacent to it.
3. R has already been matched.

Considering only "nearby" regions to the point of failure assumes that the matching process cannot stray any further than to the adjacent equipment region before it fails. By moving breadth first through the geometric space, which is represented explicitly in the model map, the matcher does not go any further than into an adjacent equipment region before the current region is entirely matched. Section 5.3.4 discussed the effect of matching order on the location of possible reconfigurations. In the implemented cases, checking only the local model region and adjacent regions for reconfigurations is sufficient to find all the possible correct reconfiguration points.

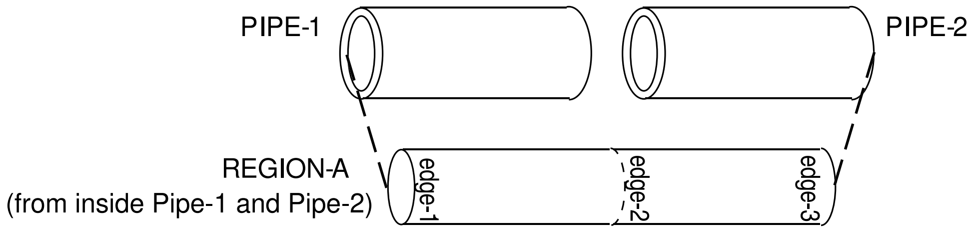


Figure 6.2: An example of an existing cycle of edges where the splitter may split a region.

These categories and parameter transforms clearly do not encode all the knowledge about application of parameters in the physical world. For example, even though the mass in a region need not be uniformly distributed, if we know that it is, we can assign the mass to split subregions according to their relative volumes. If we know that two solid bodies are rigidly connected, and we know the angular speed of one body, when we merge these two regions, we also know the angular speed of the resulting region. Other limitations derive from the spatial representation employed. Our system is not sophisticated enough, for example, to add the lengths of two cylinders when those cylinders are merged end-to-end. By encoding some of the parameter transformation knowledge, though, we have been able to successfully reconfigure and match models, thus identifying parameter transformations as a necessary component of such systems.

6.3 Merging and Splitting Regions

The spatial operations that underlie the intensive and part/whole reconfiguration algorithms are `merge` and `split`. `Merge` takes two regions that have a shared surface and forms a single region. The semantics of our geometric representation require that regions have surfaces (`ports`) in one-to-one correspondence with adjacencies to neighboring regions. When two regions are merged, adjacent ports may now represent adjacencies to the same neighboring region. `Merge` finds these ports and merges them into a single port. Similar to region merges necessitating port merges, port merges can require edge merges. The `merge` function ensures that the resulting region retains the required semantics by performing the necessary merges at all levels.

`Split` takes a region and a cycle of connected edges within that region and returns two regions, where the two regions are formed by splitting the old region and inserting new ports (surfaces) with the cycle as the port boundary. In our limited geometric representation, splitting is performed only on regions that meet two conditions. First, the region must be simply connected. That is it cannot have any holes like a doughnut. Second, the region must have a set of edges that are connected in a cycle that do not already form the boundary of some port in the region. Figure 6.2 shows an example of where our system can split a region of liquid flowing through two pipes. Region-A has no holes in it and it has a cylce of edges, in this case a single edge which is a closed curve (Edge-2), that does not already form the boundary of a port in the region. Our system cannot make arbitrary splits. A more sophisticated geometric reasoning system could do this and it would be necessary for an `extensive` reconfiguration algorithm.

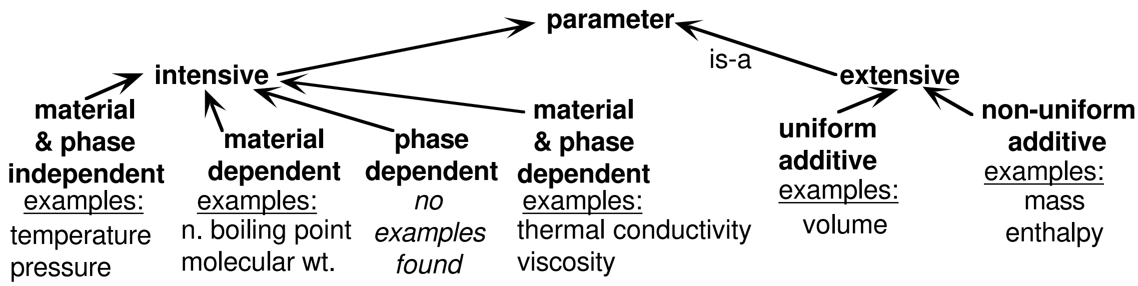


Figure 6.1: The portion of the functional type hierarchy that classifies parameters for parameter transformations.

material dependent: If the `material` types of the merging entities are the same, the `values` are the same within precisions, and all other parameter attributes match, apply the parameter to the new entity.

phase dependent: If the `phase` type of the merging entities are the same, the `values` are the same within precisions, and all other parameter attributes match, apply the parameter to the new entity.

material & phase dependent: If the `phase` and `material` types of the merging entities are the same, the `values` are the same within precisions, and all other parameter attributes match, apply the parameter to the new entity.

uniform-additive extensive: If all other parameter attributes match, add the `value` attributes.

non-uniform-additive extensive: If all other parameter attributes match, add the `value` attributes.

extensive: Parameter not applied to new entity.

<u>**Parameter Transforms for Splitting**</u>

material & phase independent: Apply the parameter to the new entity.

material dependent: Apply the parameter to the new entity.

phase dependent: Apply the parameter to the new entity.

material & phase dependent: Apply the parameter to the new entity.

uniform-additive extensive: If all other attributes match, divide the `value` into two pieces proportional to the size of the resulting entities.

non-uniform-additive extensive: Parameter not applied.

extensive: Parameter not applied.

Here we use the *extensive/intensive* terminology as it is used in thermodynamics [48,71] to mean that the parameters either do or do not depend on amount.

Type Transforms for Splitting

material type: The new entities get the same `material` type as the old entity.

phase type: The new entities get the same `phase` type as the old entity.

functional type: The new entities get one of (`region, port, edge, endpoint`) as its functional type, depending on which of these types is a `functional type` superclass of the `functional` type of the old entity.

geometric type: The new entities get no `geometric` type assigned.

The `geometric` type transformations are particularly weak in this system, because the geometric representation is weak. If a more quantitative representation were used with a more sophisticated spatial reasoner, significantly more geometric information could be preserved through merges and splits. If a different spatial representation were used, it may also eliminate the geometric type hierarchy used in this work in favor of some other more convenient representation.

6.2.2 Transformations of Parameters

When regions are merged or split, parameters describing the old region(s) may also describe the new region(s). Whether a parameter gets carried through a merge or split depends (at least) on the type of parameter the spatial extent of the parameter, the types of the regions, and whether they are being split or merged. For example, if we have two adjacent regions of water, one liquid and one vapor, both at a uniform temperature of 100 C, then the merged region also has a temperature of 100 C. However, the viscosities of the two regions are not the same (because viscosity depends not only on material but also on phase) and so we do not have a viscosity parameter that describes the whole region. However if we split the liquid region, both parts would have the same viscosity. The parameter types, region types, and reconfiguration types (merge or split) all play a role in this example. To more clearly illustrate the role of spatial extent, consider two cylindrical regions of liquid, perhaps inside a pipe, being merged. We know the surface temperature at one end of one cylinder. Call this surface A. If the two cylinders are merged end-to-end so that surface A becomes an end of the merged cylinder, then we know the end surface temperature for the merged region.

It may appear that one would have to encode a seperate transform for each parameter type and region type, but in this work we find that we can classify the parameters we encountered in implemented cases into 7 catagories (one of which is not used). Two transforms are encoded for each category, one for merging and one for splitting. The same transforms apply to merging and splitting regions, as well as merging and splitting ports (surfaces) and edges. Port and edge merging can occur during merging of regions, depending on the geometric adjacencies. Figure 6.1 shows the 7 categories we identified, which are part of the `functional` type hierarchy.

Parameter Transforms for Merging

material & phase independent: If the `values` are the same within stated precisions, apply the parameter to the new entity.

methods, only a subset of those identified are implemented In both the intensive and part/whole reconfiguration methods, we make assumptions that we believe will be satisfied in many situations. For intensive reconfiguration we implement a search termination heuristic that may miss some correct matches. In the part/whole reconfiguration, some equipment faults will not be recognized and the components will be reconfigured into an aggregate that is assumed to be working properly. However, the implemented methods have been sufficient to handle some engineering models in a number of different matching and reconfiguring situations, indicating that methods can be developed incrementally, adding methods as more models are examined. We view the space of possible model matchings onto physical situations as being infinite, but still worth exploring. This work takes a step into that mostly unexplored territory.

6.2 Transformations during Merging and Splitting

When regions are merged or split, types and parameters must be assigned to the new regions. We use the term *transformations* for the methods that calculate the new types and parameters from old types and parameters. These methods encode fundamental engineering knowledge about the nature of properties (types and parameters) of the physical world. The transformations described here are used whenever any two regions are merged or any region is split, regardless of the reconfiguration method (intensive or part/whole) which is ultimately responsible for the reconfiguration. The reconfiguration method can add to or modify the types or parameters transformed, because reconfiguration methods encode special case knowledge about transformations. For example, the part/whole reconfiguration will modify the functional type of the merged regions it creates. The transforms are applied to all spatial entities that are split or merged, including `ports` and `edges` in addition to `regions`. `Port` and `edge` merging occurs within `region` merging.

6.2.1 Transformations of Types

Types from each of the four types hierarchies are transformed using the following rules.

Type Transforms for Merging

material type: The new entity gets the nearest common ancestor in the `material` type hierarchy of the two old entities' `material` types.

phase type: The new entity gets the nearest common ancestor in the `phase` type hierarchy of the two old entities' `phase` types.

functional type: The new entity gets one of (`region, port, edge, endpoint`) as its functional type, depending on which of these types is a `functional type` superclass of the `functional` types of the old entities. (The old entities will both be either `region, port, edge,` or `endpoint` in any legal merge.)

geometric type: The new entity gets no `geometric` type assigned.

Appen-dix	Model	Equipment Description	Reconfigurations triggerd by failure to find:			Reconfiguration Method Used:		# Recon-figs	# Matches Found	Run-time (hr:min)
			p-obj[1]	req-p[2]	neg-s[3]	Inten-sive	Part/ Whole			
D.3	Dittus-Boelter	PIPES-1		x	x	x		2	1	1:29
D.4	Dittus-Boelter	PIPES-2			x	x		3	1	1:25
D.5	Dittus-Boelter	PIPES-3			x	x		1	0	0:11
D.6	Dittus-Boelter	PIPES-4		x		x		1	0	0:01
D.7	Dittus-Boelter	PIPES-5	x			x		1	1	0:14
D.8	Dittus-Boelter	PIPES-6	x			x		2	0	0:15
D.10	Dalle-Molle	CSTR-2	x	x	x	x	x	3	16	4:46
D.11	Dalle-Molle	CSTR-3	x				x	2	16	2:04
D.12	NPSH	Detailed-Pump	x				x	1	2	0:08
D.13	Wear Ring	Detailed-Pump	x				x	1	2	0:10
D.14	Hydraulic HP	Detailed-Pump	x				x	1	1	0:11
D.15	Hypo-thetical	Detailed-Pump	x				x	2	4	0:18
D.16	Friction Factor	PIPES-1			x	x		2	2	1:38

1. p-obj = positive object 2. req-p = required parameter 3. neg-s = negative set

Table 6.1: The 13 test cases implemented that do reconfigurations.

also be divided into its parts, each having an appropriate functional type such as **impeller**, **shaft**, or **casing**. *Part/Whole* reconfiguration handles these cases. It identifies and merges the spatial portions in a collection identified in the equipment description as being a pump, or it identifies the subparts of the pump present, using the **functional** type hierarchy as a guide (which has not only **is-a** but also **part-of** relations) for finding the appropriate subparts.

For models to be correctly matched after reconfigurations in an equipment representation, the reconfigurations must correctly transform both types and parameters describing the old region (or portions of it) to types and parameters of the new region. For example, if two adjacent regions of **liquid water**, both at 54 F are merged, then the resulting region should also have types **liquid** and **water** and has a temperature of 54 degrees. However, if a **stainless-steel tank** holding the water is split in half horizontally, the two halves are still **stainless-steel** but are not of type **tank**. Transformation methods are fundamental knowledge about the physical world that are implicitly used by engineers when applying models. We encode some of the transformation knowledge for both types and parameters. This transformation capability forms the foundation for the reconfiguration methods.

We implement the intensive and the part/whole reconfiguration methods on top of the encoded parameter and type transformation methods. Reconfigurations are triggered when a match fails and a region nearby the failure point in the model map will allow a reconfiguration of the equipment description. Since a merging or splitting can affect only the region being merged or split and the immediately adjacent regions, the methods can be triggered by considering only the immediate or adjacent regions at the point of failure in a match, if our matching never strays more than one region before a non-match was detected. By moving through the geometric space of the model map in a breadth first fashion, we match the immediate region and adjacent regions completely before moving farther afield. (See Section 5.3.4 for a detailed example.) Of the 16 test cases implemented, 13 do a least one reconfiguration, and 10 of the 13 reconfigure to successfully match the model map. Table 6.1 summarizes the 13 reconfiguring cases. Three cases, D.5, D.6, and D.8, were designed to allow reconfigurations but to not be reconfigurable to a complete match. In all of the cases, local triggering and reconfiguring within the current region or adjacent regions in both the model and the equipment description were sufficient to generate the correct forms, if it was possible to do so. In two cases, D.4 and D.10, three separate reconfigurations (where each reconfiguration can contain multiple merges or splits) were triggered and performed during a single model-matching.

Limitations in the methods implemented arise from the simplicity of the geometric representation, the limited coverage of the reconfiguration and transformation methods, and the assumptions made within the reconfiguration algorithms. The geometric representation is adjacency based and is only semi-quantitative. Some quantitative geometric requirements of models cannot be expressed. The splitting capability is also limited by this representation. The geometry prevents some parameter transformations from being implemented. For example, we should be able to add the lengths of two merged cubes, but our geometric representation is not detailed enough. In terms of limited coverage, we implement only 2 of the 3 reconfiguration methods identified, and examination of additional models may reveal more reconfiguration methods. Similarly with the transformation

Chapter 6

Individuating: Reconfiguring Equipment Descriptions

6.1 Summary

Reconfiguration algorithms split and merge regions existing in an equipment description. A reconfiguration method is triggered when a model requires a physical entity that may be present in the equipment description but is not represented in the same form as required by the model. We identified three reconfiguration methods by examining engineering models, problems, and solutions to determine what the models required and the range of ways that the corresponding physical entities could be divided into individuals in a representation.

1. Intensive Reconfiguration
2. Extensive Reconfiguration
3. Part/Whole Reconfiguration

Intensive reconfiguration deals with merging and splitting regions when the individual required by the model does not implicitly or explicitly specify the amount or size. (The *intensive/extensive* terminology is based on that used in physical sciences such as thermodynamics, where *intensive* properties are those that do not depend on amount and *extensive* properties do depend on amount [48,71].) Model regions having only `material` and `phase` types without any specifications on size, mass, volume, etc., are intensive, and the intensive reconfiguration algorithms merge and split equipment regions of appropriate phase and material types to meet other model requirements such as adjacency or parameters. Some models specify regions that have `material` or `phase` types but also have some quantitative size or shape, such as "1 lb of liquid" or "1 cubic centimeter of the metal in the tank wall." An *extensive* reconfiguration method is required to merge and split as in the intensive case, but to meet a quantitative specification.

Model regions may also have `geometric` or `functional` types specified, which are inherently extensive. If something is a `centrifugal-pump` and we split some portion out or merge some adjacent region onto the pump, the resulting region is not necessarily a `centrifugal-pump`. How might a pump be present in an equipment description other than as a single individual with type `pump`? It could be divided arbitrarily into pieces. If so, the equipment description would still somehow have to say that the collection of pieces were spatial portions of a pump. The pump could

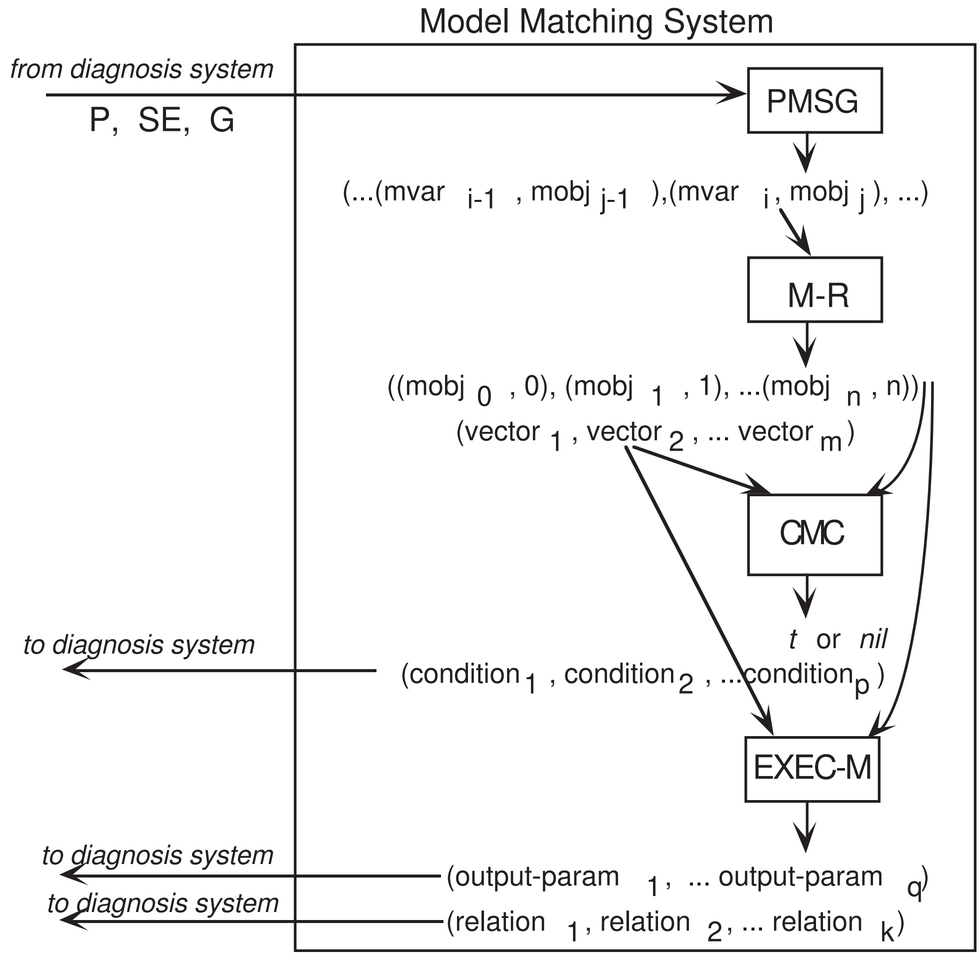


Figure 5.4: Flow of data between the four components of the model-matching system.

Appen-dix	Model	Model Descr Size	# Model Objects	# Negative Sets	Equipment Description	Equip Descr Size	Goal	# Matches Found	Run-time (hr:min)
D.2	Dittus-Boelter	54 K	34	8	PIPES-0	53 K	effects	1	0:01
D.9	Dalle-Molle	61 K	40	1	CSTR-1	92 K	values	16	0:13
D.17	Hagen-Poiseuille	47 K	33	6	PIPES-1	144 K	effects	1	0:02

Table 5.1: Statistics on test cases that did not require reconfiguration.

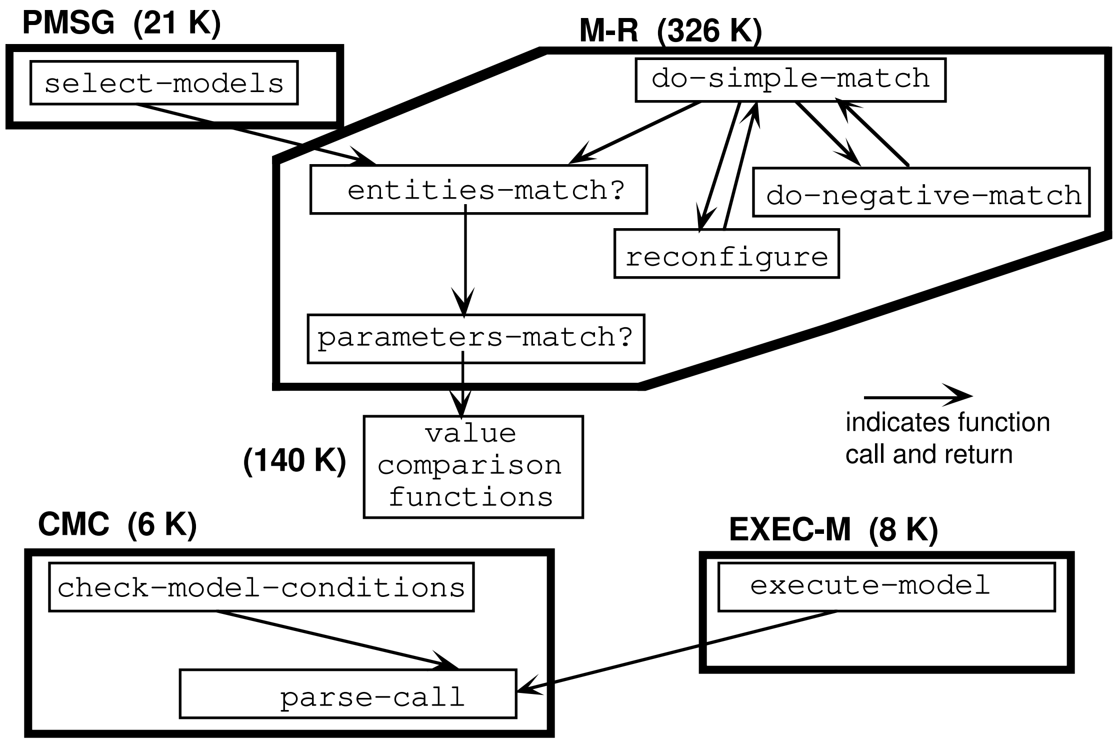


Figure 5.3: Flow of control within the four components of the model matching system. Each component is currently called interactively.

specifications observed, positive object matching, negative set matching, and pattern matching, we have implemented the first two methods. Thus the system as it currently exists cannot handle models which require specifications of patterns of objects and relations rather than a particular fixed set of objects and relations. Figure 5.3 shows the flow of control within PMSG, M-R, CMC, and EXEC-M along with the sizes of the algorithms as implemented in Common Lisp.

The inputs and outputs to the system as well as the inputs and outputs passed between the four steps of the algorithm are summarized in Figure 5.4. Appendix D gives inputs and outputs generated by the four steps for each of 16 test cases. The appendix also shows the side effects which include reconfigurations of the equipment description as well as parameter objects created and installed into the equipment description.

The data in Appendix D illustrate the capabilities described in this chapter. The test cases provide examples of use of each of the three kinds of goals, `causes`, `effects`, and `values`. All cases use both positive and negative matching. The appendix shows the order in which model objects were matched. Model objects are annotated as to whether they are positive objects, required parameters, negative objects, or optional parameters, illustrating the priority breadth first search. Three cases which match without reconfiguring probably best illustrate the algorithms described in this chapter. These cases are summarized in Table 5.1. We discuss complexity and efficiency of the algorithms in Section 7.4.

5.6 Executing Models and Installing Outputs (EXEC-M)

EXEC-M calls models and installs returned values into the equipment description using information from the model description that specifies how to do that. It takes only the assocation list and one of the match vectors generated by M-R as input. It explicitly returns the names of parameter objects (that it creates by side effect) which hold the output values from the model.

EXEC-M parses the *call form* for the matched model, identifying the equipment parameter objects matched to the model variables, and passing the specified attributes of each equipment parameter to the model function. Our assumption is that models are accessed via function calls. The function name is specified in the *call form.* For each output variable in the *return form* of the model description, EXEC-M creates a new parameter object. This object is installed into the equipment description with links corresponding to those in the model map. EXEC-M installs `EXEMPLIFIES` links to the same `types` to which the model variable is linked. For other links, EXEC-M finds the equipment object that matches the model object linked to the model parameter and installs the equipment link there.

EXEC-M generates attribute values for output paramters in three different ways.

1. specified

2. carried

3. calculated

When a model's output parameter object is copied to make the equipment parameter, any attribute values that are *specified* in that model object are copied to the equipment object. For example, the Dittus-Boelter model implemented for this thesis (Appendix C.2) calculates a heat transfer coefficient that has `value-units` of $BTU/hrft^2F$. This is specified in the model variable's `value-units` attribute. EXEC-M also parses the *carried variables* list in the model description. We find that models may not calculate or use some parameter attributes, but those attribute values are implicitly assumed to be the same as for inputs. The models examined during this work often treat `time` and time-related attributes implicitly. In the Dittus-Boelter model, `time` for the output heat transfer coefficient is carried from the `time` for the input mass flow rate. EXEC-M parses the *return form* to install the *calculated* values returned by the model function into the correct attributes of the generated output parameter objects. We observe specified, carried, and calculated attributes in *all* of the models implemented for this work. Appendix D shows the parameter objects generated for each successful match with annotations (*specified, carried, calculated*) indicating the source of the installed value.

5.7 Implementations

We implement all four steps described, potential match set generation (PMSG), matching (M-R), condition checking (CMC), and model execution (EXEC-M). Of the three kinds of matching entities and relations in the set. If all entities and relations in the set match, then a failure is signalled to the simple matcher. We do not allow reconfigurations if the negative match fails to find the whole negative subgraph, since all negative specifications in this thesis deal with the form of regions having only phase and material types rather than functional type. If negative specifications for functional type objects are found in models, then this restriction should be revised. If the simple match fails because the negative match succeeded, though, reconfigurations are attempted.

5.5 Checking Model Conditions (CMC)

Once a match is made to a model map by M-R, CMC checks conditions listed in the *approximation conditions* and *enabling conditions* lists of the model description. This function loops through the *call forms* in these lists, passing the appropriate attribute of the the matched equipment parameter to the condition function specified. The inputs to CMC are

1. condition-list name, one of [`approx-conditions`, `enabling-conditions`],

2. association-list, and

3. match-vector.

The condition-list name simply tells which of the two kinds of conditions to check. The association-list and match-vector are outputs from M-R. The match-vector can be chosen from amongst the vectors in the vector-list returned by M-R. Since no model preference techniques, like those developed by Nayak [51] or Iwasaki and Levy [38], have been incorporated into the algorithms, the vector choice is made interactively.

CMC returns NIL (false) if any condition in the list returns nil. Some conditions may not be testable, given the current match, because some parameters may not have been matched. Parameters are not required to match unless they are input parameters for the model. Conditions may be specified on the output parameters, which are not available for testing until after model execution. Conditions may also be specified on parameters that do not participate in the model. For example, the Dittus-Boelter model requires that the inlet, outlet, and pipe wall temperatures meet certain conditions for that model to apply (See Appendix C.2.3, Approximation Condition 32), but temperatures are not used in the calculations made by that model. Since a model may be executed when inputs are available, and since diagnosis systems keep track of assumptions on which conclusions are based, our approach is to allow a model to be executed if all of its testable conditions are met, returning untestable conditions to be treated as assumptions. Thus, the outputs of CMC are either NIL or (T, list-of-conditions). Appendix D shows the return values of CMC for each of the 16 test cases implemented for this thesis.

other side of Port-A1 (not shown in the figure) can possibly fix the failure. Reconfiguration is discussed in Chapter 6.

Some preferences are interjected into the breadth first traversal order, partly for efficiency and also for correctness in matching negative objects. Instead of a simple queue, a priority queue is used. At each step of the match, one is considering some model object A previously matched with some link that has not yet been matched leading to another model object B. B may or may not have been matched. (Most objects in model maps have more than one link, and they will be seen during matching one time for each link, verifying that each link matches an equipment link.) The following priorities based on characteristics of A and B are used.

Priority 1: B has been matched.

Priority 2: B is a required parameter.

Priority 3: B is non-negative entity.

Priority 4: B is an optional parameter.

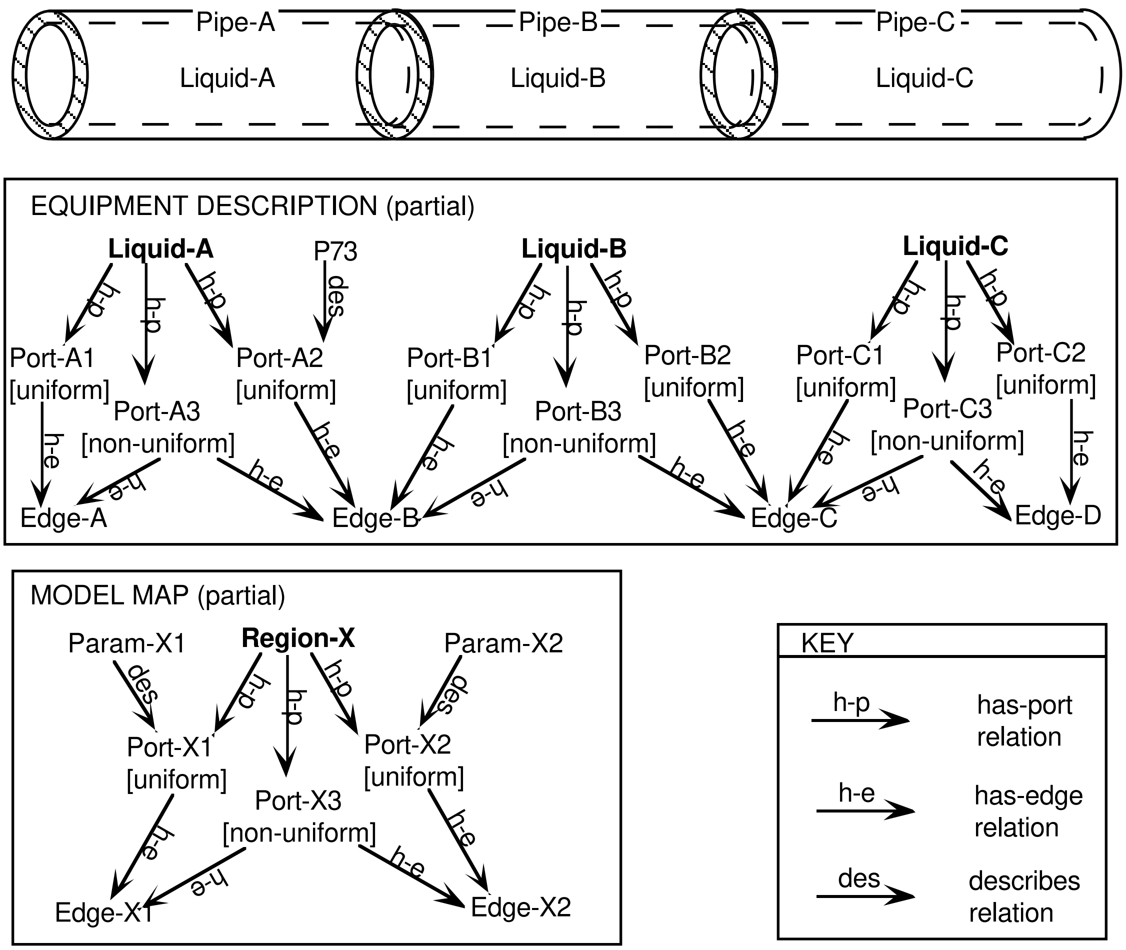
Priority 5: B is a negative entity.

The guiding principle for setting up Priorities 1-4 is earliest possible elimination of partial matches to reduce work done later when these partial matches are extended. Objects that have been matched get top priority, because the matcher verifies that another link to that object exists and can eliminate partial matches that were already made, but cannot add any new matches. Priority 2 and 3 items can both add and remove matches, but parameters occur less often in the model maps and equipment descriptions and thus have more chance to delete matches. Priority 4 items, optional parameters, cannot delete any matches. Negative entities must be matched last because of the semantics of negative objects, as described in Section 5.4, although negative matches can only delete partial matches from the list of correct matches.

5.4 The Negative Matching Algorithm

When negative entities are encountered in the M-R step, simple matching calls negative matching. The priority queue within simple matching ensures that all positive objects have been matched before any negative objects are encountered. Negative matching is done last because of the semantics of the negative objects and sets. They indicate entities and relations which are not present *beyond* the positive entities and relations. For example, the negative `Port-4` in Figure 4.5 specifies that the cylindrical region of liquid has no ports beyond the three specified ports, `Port-1, Port-2`, and `Port-3`.

The intention of grouping negative entities into sets is to indicate that there is no matching set of entities in the equipment description exhibiting the same relationships. Thus we do negative matching set-wise. Each set of negative entities is assumed to be a connected subgraph within the model map's semantic net. The simple matching algorithm is applied to this set to match all



A "DEPTH FIRST" PARTIAL MATCH

MODEL	EQUIPMENT
Param-X2	P73
Port-X2	Port-A2
Edge-X2	Edge-B
Port-X3	Port-B3
Edge-X1	Edge-C
Port-X1	Port-C1

A "BREADTH FIRST" PARTIAL MATCH

MODEL	EQUIPMENT
Param-X2	P73
Port-X2	Port-A2
Region-X	Liquid-A
Edge-X2	Edge-B
Port-X3	Port-A3
Port-X1	Port-A

Figure 5.2: An equipment description and model map where depth first matching of the model map strays through three regions.

tested are returned by the condition checker, described in detail in Section 5.5, for possible use by a diagnosis system.

5.3.4 Matching Relations and Order of Matching

Relations that appear in a model map successfully match links in the equipment description when the two links have the same name. `EXEMPLIFIES` relations are excluded from this portion of the matching because they indicate `types` and are used in matching entities as described in Section 5.3.2. The starting point for the match is the model variable $mvar_i$ and model object $mobj_j$ returned by PMSG along with the corresponding inputs to PMSG, P and SE from the equipment description. Any parameter in the model may be a starting point for a match. (The implemented algorithms actually will take any object in the model map with a corresponding physical object as a starting point.) The matching proceeds through the model map attempting to match all links by name and all objects by type. Since this work assumes that models address connected regions of space, the model map is a single semantic net, allowing the matching to start at any point and find all objects by following links. Two cases implemented for this thesis illustrate the Dittus-Boelter model map being matched in two different orders. Appendices D.3 and D.7 show the association lists (which indicate the order of matching) for these two cases.

In this system, objects have unique names while links of the same type are not distinguishable, so object names are stored to keep track of matches. The matcher proceeds in an approximately breadth first traversal of the model map's semantic net.

Finding: A breadth first traversal of the model map's semantic net allows local reconfiguration.

We ran an experiment using a stack to get a depth first traversal in place of the breadth first traversal, ostensibly to compare speeds of the two approaches. We find that depth first traversals can make it difficult to reconfigure, because one cannot predict which equipment regions to reconfigure for certain depth first traversals. Figure 5.2 shows a situation where a depth first traversal can result in reconfiguration difficulty. That figure shows three connected pipes with liquid inside the pipes. The liquid is divided into three regions, Liquid-A, Liquid-B, and Liquid-C. (This is the equipment represented in the PIPES-0 Equipment Description in Appendix B.) A partial equipment description for the liquid regions is shown in the figure. A portion of a model map (similar to the DITTUS-BOELTER Model Map in Appendix C) is also shown in the figure. Consider the partial match shown in Figure 5.2 generated during a depth first traversal of the model map. On attempting to match Param-X1 via a `DESCRIBES` link from Port-C1 (the last object matched in the series), the match fails. The match has strayed through three regions in the equipment description, when the ports and edges in the model map were all part of one region. Breadth first traversals do not allow a match to stray any further than the adjacent region. Thus when a failure occurs, reconfigurations (merging or splitting of regions) only have to be tested in the immediate vicinity of the object at which the failure occurred. Consider the breadth first partial match shown in Figure 5.2. This match also fails when a match for Param-X1 is sought via a `DESCRIBES` link from Port-A1. However, at this failure point, reconfiguring by merging or splitting Liquid-A or the region adjacent on the

5.3.2 Matching Entities

Each object in the model map semantic net that is not a parameter is matched to equipment objects using only the types. For each type specified for the model object, the equipment object must have either the same type or a subclass of that type. The model object may have up to four types, all of which must meet this condition. Any of the four possible types that are not specified in the model object are ignored for the purposes of matching. Thus if a model map contains an object A where `EXEMPFLIFIES(A, pump)` is the only type, and the equipment description contains object B where `EXEMPLIFIES(B, centrifugal-pump)` and `EXEMPLIFIES(B, stainless-steel)`, then B matches A, assuming that `centrifugal-pump` is a subclass of `pump` in the functional type hierarchy. The model object A "doesn't care" about the material type, and thus any material type will suffice.

5.3.3 Matching Parameters

Parameters, which are also objects in the model map semantic net, are matched first on types just like entities. They will only have one type, a functional type. Model variables may have values for their 29 attributes (See Figure 4.4) which also are matched to physical parameter attributes. For example, if the model requires scalar values, the `value-dimension` will be 1. A physical parameter must have 1 for its `value-dimension` to match. Model variables usually do not specify a particular value for a parameter (because models usually calculate a range of values from a range of possible inputs), but if a `value` is specified, it is matched to the physical parameter's `value` taking into account the precision specified in `value-precision`. Thus the two values are considered equal if the possible ranges for both values have any overlap.

Parameter matching code handles a number of different types of values, interval types, precision types, and dimensions so as not to unnecessarily limit the kinds of models that could later be added to the system. Since parameter objects may be meaningfully compared to other parameters that do not have identical characteristics on all attributes, for example, comparing a `point` value to a `closed` interval value, the number of cases is relatively large. This supporting code makes up about 28% of the code written (See Table 3.1) to implement the four steps of the model matching algorithms.

Some of a model's variables are treated as optional with respect to matching. Any variable that is *not* an *input variable* may be ignored by the matching algorithm. These variables participate in the *approximation conditions* (Section 4.3.6) or in the *enabling conditions* (Section 4.3.7)). If no values are found, the models may still be run because these values are not inputs. However, since some *approximation conditions* or *enabling conditions* cannot be tested, one cannot assess the applicability of the model. These variables are treated as optional, though, because the value may become known later and more importantly because the condition may apply to an output of the model which will not be available until the model is run. Nayak [51] has identified some models with this characteristic. Not knowing a value is being treated differently than having a value that fails to meet a condition. Such untestable conditions can be treated as assumptions by a diagnosis sytem's truth maintenance component to be later verified or denied. Conditions that cannot be type hierarchy starting with type T, where `EXEMPLIFIES`(P, T). It gets all model variables that also `EXEMPLIFIES`$(mvar_i, T)$, which is an easy operation since relations are implemented as named links. One finds the objects linked to T via `EXEMPLIFIES` links. For each $mvar_i$, the entities satisfying `DESCRIBES`$(mvar_i, mobj_j)$ are found. This also is a matter of getting all objects linked via a `DESCRIBES` link. If $mobj_j$ matches SE, that is all of $mobj_j's$ types are superclasses of types of SE, then the pair $(mvar_i, mobj_j)$ is returned as a potential match. Each $mvar_i$ is a part of only one model map, so specifying $mvar_i$ specifies the corresponding model.

5.3 The Simple Matching Algorithm

The simple matching algorithm implements the Match-Reconfigure (M-R) step by matching entities and relations, calling negative matching when negative entites are encountered, and calling reconfiguration when matching fails. Reconfiguration is described in Chapter 6.

5.3.1 Inputs and Outputs

The simple matching portion of M-R matches the objects and links in the semantic net representation of model maps to the semantic net representation for equipment descriptions. M-R takes four inputs.

1. P, the physical parameter input to PMSG
2. SE, the spatial extent of P, an object also input to PMSG
3. $mvar_i$, a model variable matching P, an output of PMSG
4. $mobj_j$, a model entity matching SE and corresponding to $mvar_i$, also an output of PMSG

The inputs act as a starting point for the match of this particular model to the equipment description surrounding paramter P. Simple matching finds all matches in the equipment description available. (It may call the reconfiguration algorithms, though, which do not try all possible reconfigurations.)

The simple matching algorithm returns a list of matches for the objects in the model map containing $mvar_i$. It has the following two outputs.

1. An association list, giving an ordering on model map objects
2. A list of vectors, indicating correct matches to physical objects

The association list associates integers 0 through n with the objects in the model map. Each vector in the vector list represents a match. The vector has n + 1 elements and in element i holds the name of a physical object matching model element i (as numbered in the association list). All the links in the model map amongst these objects have also been matched. Appendix D shows the association list and matching vectors generated for each of the 16 test cases.

```
(defun SELECT-MODELS (P, P-attribute, SE, goal-G)
for each superclass Tk of T0, where EXEMPLIFIES(P, T0),

    for each model variable mvari where EXEMPLFIES(mvari, Tk),

       if   mvari matches P on type and attributes
            AND
            mvari's model satisfies goal-G with respect to
            P-attribute and mvari

       then for each entity mobjj such that DESCRIBES(mvari, mobjj),
            if mobjj matches SE,
            then return (mvari, mobjj))
```

.

Figure 5.1: Pseudocode for the PMSG algorithm as implemented by the function `select-models`.

model (Appendix C.4), has an *affected* variable as an *input* variable, and thus can be used to identify causes.

Goal: find effects. Models that can satisfy the goal of finding effects must satisfy the following conditions.

1. P is in the *causal variables* list. (Section 4.3.4.)
2. P is in the *input variables* list. (Section 4.3.2.)

OR

1. P participates in an *enabling condition.* (Section 4.3.7.)

Finding effects is the reverse of finding causes for some abnormal value. If a model calculates acceleration from force, and P is force, than that model is relevant to finding effects of that force. If P participates in enabling conditions, its value may allow the phenomenon described by the model occur, in which case it does not have to be an *input variable.*

Goal: calculate values. For a model to be able to generate a value for a parameter P, P must be an output and the model must satisfy the following condition.

1. P is in the *output variables* list. (Section 4.3.3.)

The same model may be chosen to satisfy different goals as illustrated by two cases implemented for this thesis. In one case (Appendix D.2), the Dittus-Boelter model is chosen to find `effects` of a changed heat capacity. In another case (Appendix D.3), the same model is identified to calculate the value of a heat transfer coefficient.

The PMSG algorithm takes advantage of the type hierarchies to consider only model parameters of the correct type. The algorithm is written in pseudocode in Figure 5.1. It traverses the functional is an object representing something that occupies volume, is a surface, a line, or a point. PMSG generates a list of pairs, $(mvar_i, mobj_j)$, where $mvar_i$ is a model variable and $mobj_j$ is an object in the model map which satisfy the following conditions.

1. P *matches* $mvar_i$.

2. SE *matches* $mobj_j$.

3. `DESCRIBES(mvar_i,mobj_j)`.

4. The model to which $mvar_i$ belongs satisfies the goal G.

Model Entity A matching Physical Entity B means that for each type of A, B has a type that is identical or a subclass. Model Variable A matching Physical Parameter B means in addition to matching types, for each attribute of A that has a value, B has an equal value (within the stated precision if the attribute is numerical.) Entity matching and parameter matching are described in Sections 5.3.2 and 5.3.3, respectively. Appendix D lists inputs and outputs of PMSG for all 16 test cases.

Generating the potential match set is essentially a backwards application of model matching, finding model maps that match the equipment. We have made an arbitrary choice as to how far to go in the reverse matching to identify the set of models. One could attempt to do the entire matching for each model encountered at this step. However, applying model preference techniques before complete matching can eliminate the need to match some models, and thus eliminate the significant work involved in matching. We have chosen to do a small amount of work that can significantly reduce the size of the potential match set from the set of all models in the system. We have not investigated the tradeoffs in doing more work to further reduce the size of this set. We interactively choose models from the set generated, and assume that we could employ model preference techniques in the future to make choices automatically.

We assume that the diagnosis system will have one of three goals with respect to P, either to find `causes`, find `effects`, or calculate `values`.

Goal: find causes. If the goal given to PMSG by the diagnosis system is `causes`, then only models that can potentially identify a cause for the parameter P are selected. The models must satisfy the following conditions.

1. P is in the *affected variables* list. (Section 4.3.5.)
2. P is in the *input variables* list. (Section 4.3.2.)

If the diagnosis system is looking for causes for P having some abnormal value, then only models in which P is an *affected variable* are relevant. In this case P must also be an *input variable*, so that the output will be determined by the value of P. An example is a model that calculates force from acceleration. Acceleration is both an *affected variable* and an *input variable*. Force is a *causal variable* and an *output variable.* Thus such a model could calculate what force might be causing some known abnormal acceleration. One of the models implemented for this work, the Wear Ring

1. Simple Matching
2. Negative Matching
3. Pattern Matching

Simple (or positive) matching (Section 5.3) takes specifications of a fixed set of objects and relations and matches them. Negative matching (Section 5.4) takes specifications of objects and relations that cannot be present (beyond the simply matched entities) and ensures that they are not there. Pattern matching, which was not implemented, takes a specification of a variable number of entities and relations in some pattern and matches them. Future work with other engineering models may reveal additional specification types and matching algorithms.

Building these algorithms and the 16 test cases that use them revealed several important findings. First, the order in which a model map is traversed for matching must be approximately breadth first through the physical space. The breadth first approach ensures that when failures occur, only local reconfigurations within the current region or adjacent regions need be considered to try to repair the failed match. Second, all the models implemented required some kind of negative specification and thus used the negative matching capability. Third, all the models tested implicitly assumed values for output parameter attributes beyond those calculated by the model itself. Typically they assumed that the `time` and time-related attributes had values the same as the `time` of the input parameters, but in some models, such as the Wear Ring Model, `value-units` were also implicitly assumed. Thus EXEC-M generates calculated values by calling the model, and also carries values from inputs to outputs in all test cases.

5.2 Generating A Potential Match Set

Potential Match Set Generation (PMSG) identifies models that are candidates for a match from amongst the possibly large set of models available to the model matcher. All models that may successfully match the equipment description at the point indicated and satisfy current goals of the diagnosis system are guaranteed to be in the set that PMSG identifies. One could apply model preference or model selection techniques such as those developed by Nayak [51] or Iwasaki and Levy [38] to identify the best model in this set to match first. Preference techniques are not included in the implemented system. Model selection from the model set generated by PMSG is done interactively.

PMSG takes three inputs from a diagnosis system.

1. a parameter P (and an attribute of P) in the equipment description
2. the spatial extent SE of P
3. the goal G with respect to P

In the system implemented for this thesis, the parameter and spatial extent are objects such that `DESCRIBES(P,SE)`. Along with the parameter P, one of its 29 attributes (usually `value`) is specified, so in this discussion, P will refer to both the parameter object and the specified attribute. SE

Chapter 5

Model-Matching

5.1 Summary

The algorithms developed break the model-matching problem into four steps.

Step 1: Potential Match Set Generation (PMSG). PMSG, described in Section 5.2, takes a parameter in the equipment description, its spatial extent, and a goal (one of find `causes`, find `effects`, or calculate `values`) and returns a list of the models in the system that address a variable of the same type and spatial extent as the equipment parameter for the desired goal.

Step 2: Match-Reconfigure (M-R). M-R takes a model from the list generated in Step 1 and attempts to match the rest of the model map (beyond the variable identified in the first step.) If this matching fails, the system evaluates the potential for reconfiguration and may attempt to reconfigure. The *matching* portion of M-R is described in this chapter, Sections 5.3 and 5.4, and *reconfiguration* is described in Chapter 6.

Step 3: Check Model Conditions (CMC). CMC, discussed in Section 5.5, evaluates the conditions in the model description's *approximation condition* and *enabling condition* lists given the equipment parameter values matched to the model's variables in Step 2. CMC returns nil if at least one condition is not met. It returns t if all *testable* conditions are met along with a list of *untestable* conditions. A condition is *untestable* if it involves some *optional* parameter that was not matched. Any parameter that is not an input for the model is *optional.* The list of untested conditions are returned to the diagnosis system to be treated as assumptions.

Step 4: Execute Model and Install (EXEC-M). If the conditions in Step 3 are met, then the model is called using the *call form* and the results are installed into parameter objects in the equipment description using the *return form* and *carried variables* list from the model description. The relations on the entities as required by the model map are also made available to the diagnosis system for dependency tracking for the newly created parameter. (See Section 3.3 for a discussion of how relations are used by a diagnosis system.)

The examples of engineering models revealed three kinds of specifications on individuals, and thus three kinds of matching capabilities.

4.3.9 Carried Variables

Many models implicitly require that a number of attribute values of outputs are the same as some input parameter's corresponding attribute value. For example, all the output values from any form of the Ideal Gas Law are implicitly assumed to occur at the same time as the input values used. (All the input values must be for the same time.) MDS provides the *carried variables* list to describe these attribute values that are carried unchanged from some input to some output variable. They are specified as a list of 4-tuples. The first element in the 4-tuple is the input variable. The second element is the attribute of that variable, such as `time` that is carried to some output. The third element is the output variable. The fourth element is the attribute of the output that receives the value. After a model is run and its output values are installed into a parameter object as specified in the return form, the attributes specified in the carried variables are installed.

4.3.10 Resources

To make decisions about which models to run in a dynamic environment, models' resource requirements must be considered, where resources may be running time, memory requirements, the type of processor required by a model, and the availability of cycles on that machine. *Resources* have been included in MDS in anticipation of future work where model-matching will be integrated into a diagnosis system working in a dynamic environment, but the examples in this thesis do no use *resources*.

molecules, which are affected by their kinetic energy, how close they are to each other, size and shape, attractive forces, etc. are approximately the same as the those of mass-less points that are the molecules of the ideal gas. The pressure condition being met does not cause a particular phenomenon. Many other researchers, including Nayak [52] and Falkenhainer and Forbus [24], use some form of approximation conditions in their modeling formalisms.

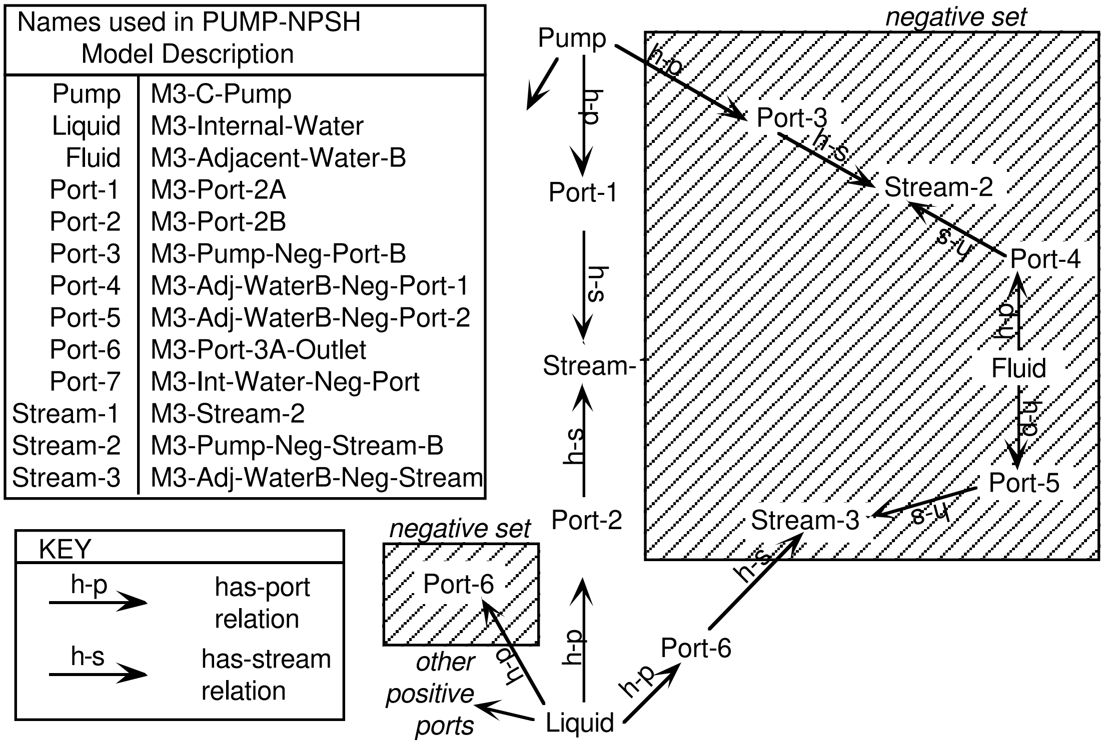
In the system implemented for this thesis, approximation conditions may be any function that is free of side effects, uses inputs specified in the model map, and does not maintain internal state. The value returned is treated as *true* for all non-nil values and *false* for nil values. The specification of the approximation conditions is then a list of *call forms* for these condition checking functions. A *call form* is a list where the first element is the function name and the succeeding elements are in pairs, one pair for each argument to the function. Each pair has a model variable as its first element, and the second element is the variable's attribute that the function takes as input. The variables themselves are represented as parameter objects in the model map just as for input and output variables. These objects may contain attribute values (such as `value-type`) indicating the requirements of the function on its inputs.

4.3.7 Enabling Conditions

Enabling conditions are restrictions on physical parameters that must be met to allow the phenomenon being modeled to occur. For example, the vapor pressure of a liquid must be equal to the ambient pressure for boiling to occur. A fuel/air mixture must be at a temperature above its flash-point to enable burning. (An ignition source may be required as well if the temperature is not above the auto-ignition point.) The associated model may or may not use these parameters as input or output. Enabling conditions are another form of causes, and so are separated from approximation conditions because they are used to identify models that might reveal causes. Enabling conditions take the same form as approximation conditions.

4.3.8 Call Form and Return Form

Since models are implemented computer programs, running them requires not only having values for inputs, but also calling a program using the appropriate syntax. MDS thus provides a way to describe the calling syntax for each model. For this work, we assume that all models are run by function calls. The call form is a list which has the name of the function to call as its first element. The remaining elements form pairs, where the first in each pair is a model variable and the second element is an attribute of that variable that is to be passed as an argument to the model. These pairs are in the order required by the model function. The return form is similar except that there is no function name, only pairs of variables and attributes. All the variables listed must exist as parameter objects in the model map. If the model requires any attributes of its input variables or output variables to be fixed (for example, `value-type` real and `value-units` gallons/minute), those attribute values are specified in the parameter object and are used at matching time to find the right physical parameters. See Section 5.3.3 for a description of parameter matching.



Names used in PUMP-NPSH Model Description	
Pump	M3-C-Pump
Liquid	M3-Internal-Water
Fluid	M3-Adjacent-Water-B
Port-1	M3-Port-2A
Port-2	M3-Port-2B
Port-3	M3-Pump-Neg-Port-B
Port-4	M3-Adj-WaterB-Neg-Port-1
Port-5	M3-Adj-WaterB-Neg-Port-2
Port-6	M3-Port-3A-Outlet
Port-7	M3-Int-Water-Neg-Port
Stream-1	M3-Stream-2
Stream-2	M3-Pump-Neg-Stream-B
Stream-3	M3-Adj-WaterB-Neg-Stream

Figure 4.6: Two negative sets used in the Pump-NPSH Model Map.

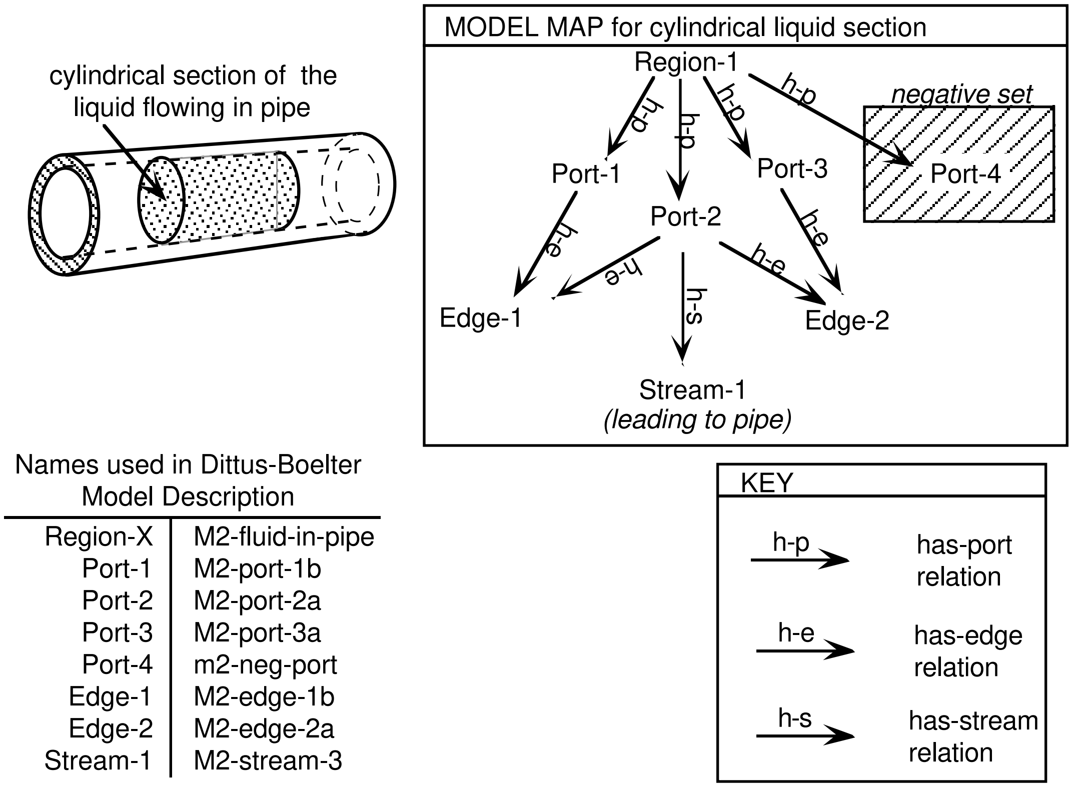
calculate the force from the acceleration, but one cannot generate a force on a mass by somehow accelerating it without applying a force. MDS makes causal variables explicit because diagnosis must both find causes for effects and find effects from causes. Causal variables are specified as a list of pairs (variable-name, attribute), like input and output variables lists. Causal variables are not necessarily output variables because calculations done by the model do not necessarily parallel the causality.

4.3.5 Affected Variables

Affected Variables are the variables being influenced by the phonomenon being addressed by the model. Just as *force* is a causal variable in the example of Section 4.3.4, *acceleration* is an affected variable. Affected variables are specified by a list of pairs analogous to the causal variables, and are used to identify models which may satisfy the goal of finding effects. Section 5.2 describes identifying models to satisfy three different possible goals.

4.3.6 Approximation Conditions

Approximation conditions are restrictions on physical parameters that must be met for the model to accurately describe the phenomenon. The Ideal Gas Law may reasonably describe most real gases if the pressure is less than a few atmospheres. Under this condition, the interactions of the



Names used in Dittus-Boelter Model Description

Region-X	M2-fluid-in-pipe
Port-1	M2-port-1b
Port-2	M2-port-2a
Port-3	M2-port-3a
Port-4	m2-neg-port
Edge-1	M2-edge-1b
Edge-2	M2-edge-2a
Stream-1	M2-stream-3

Figure 4.5: Use of a singleton negative set in the Dittus-Boelter Model Map.

`value` but could also take other attributes such as `time` or `value-precision`. The specification of input variables is a list of pairs, where the first element in each pair indicates a parameter, and the second element indicates the attribute.

4.3.3 Output Variables

MDS also specifies which of the variables in the model map the model generates as output. This specification is also a list of pairs, analogous to the list for input variables. Unlike input variables, output variables do not have to be found to successfully match a model. They are generated by the model matcher when the model is run and are installed into the model map by attaching the new parameter to the correct spatial object with the `DESCRIBES` relation. If the goal for matching is to calculate values for a given physical parameter, the list of output variables is used to determine if the model in question can produce the desired value.

4.3.4 Causal Variables

The model description maintains a third set of variables for each model, called *causal variables*. In the physical world various effects are caused by particular driving forces. For example, a temperature gradient causes heat transfer and a body force acting on a mass causes acceleration. One may the matching method for both entities and variables.

The second difference between model maps and equipment descriptions is that some portions of the semantic net in a model descripton may be specified as *negative.* These negative portions of the net represent entities with their associated relations that must not be present in the equipment description.

Finding: Some models require explicit specification of what is *not* present in addition to specifying entities that are present.

Anything not specified (either negatively or positively) is irrelevant to the model and may be either present or not present in matching equipment descriptions. The negative entities in the semantic net are grouped into sets, indicating how they should be matched. Each set is treated as a separate subnet, and if all of the entities in the set and all the relations on those entities are found in the equipment description, then the matching fails.

One example of negative entities comes from the Dittus-Boelter Model implemented for this thesis. (Appendix C explains this model and description.) The model requires a cylindrical section of fluid inside a pipe and there can be nothing else inside the cylindrical space. In the Dittus-Boelter Model Map, a negative set containing a single port attached to the cylindrical liquid region indicates that the region has exactly the positive ports specified and no additional ports. Additional ports would indicate additional adjacencies of the liquid with something other than the pipe or adjacent liquid. Figure 4.5 shows a picture of the liquid individual with a portion of the model map containing the negative object. The intended meaning of this model map requires that negative sets be matched after all positive entities. Section 5.4 describes the negative matching procedure.

The Pump-NPSH model, which calculates net positive suction head for a centrifugal pump, provides another example of negative sets. (See NPSH Model in Appendix C.) The model requires that the pump be completely filled with liquid water, and we use negative entities to indicate that no other individuals may exist inside the centrifugal pump besides the liquid region. Figure 4.6 contains a portion of this model map showing two negative sets. The sequence of negative objects in the larger set expresses this constraint by disallowing any matches where regions, such as **`fluid`**, adjacent to the water inside the pump also have shared surfaces with the pump. This negative set disallows such adjacent regions at the outlet port, `Port-6` in the figure. Another similar negative set disallows this kind of adjacency at the inlet port (not shown), forcing the outlet and inlet ports of the fluid region to be at the inlet and outlet of the pump. This model description also contains a singleton negative set with `Port-7` attached directly to the `Liquid` individual indicating that it has no other ports and thus there are no individuals "hiding" inside the liquid individual.

4.3.2 Input Variables

For each model, MDS specifies which of the variables that exist in the model map are input to the model. The variables are linked to spatial entities by the `DESCRIBES` relation, providing the spatial extent and location of the variable. Specification of *input variables* includes not only the parameter object from the model map, but also which attribute the model requires. Many models take the instead of `EXEMPLIFIES(region-452, pump)`. The matching and reconfiguration algorithms use the fixed set of relations, but are not specific to the entities addressed by those relations, including the entities in the type hierarchies (with the exception of the `region`, `port`, `edge`, `endpoint`, and `stream` subclasses). This approach allows hierarchies to be changed without changing the algorithms. Thus if a user needs a change that can be expressed through the vocabulary hierarchy, the system can easily accept that change. However, if some models or matching problems arise that require additional relations, the algorithms may require redesign.

The preceeding paragraphs have described 12 of these 13 relations. One additional relation, `ISOMETRIC`, applies to any two spatial entities that occupy the exact same space. Thus when reconfigurations generate new individuals which duplicate a portion of the existing equipment description, the relationship between the new individual and old individuals can be maintained. The `ISOMETRIC` relation is more often used between the ports of two regions rather than the regions themselves, because when merging and splitting generates new regions, some of the ports occupy the exact same space as the old regions' ports. The new region typically occupies more or less space than the old region.

4.3 The Model Description Scheme

The Model Description Scheme (MDS) is a language to describe the characteristics of models required to match and use them in a diagnosis. The model description allows the model to be treated as a black box. MDS provides a method for describing both implicit and explicit characteristics necessary for matching and running the model. The scheme addresses the physical situation required, enabling conditions, approximation conditions, causal parameters, carried parameters, and resources, all of which are implicit characteristics. It also provides a technique to describe the explicit characteristics including input parameters, output parameters, call forms, and returns forms.

4.3.1 Describing the Physical Situation: The Model Map

The model description must make explicit the physical situations in which a model applies. MDS describes physical situations with entities and relations as used in EDS. Thus, each model description contains a small equipment description which we call a *model map*. Model maps have two differences from an equipment description. First, the physical situation captured in the model description is intended to represent all the situations in which that model may apply. The entities act as variables that will be instantiated with entities from the equipment description, which are of the same type or are a subtype. Thus model entities should be the most general type possible. Variables, represented as parameter objects (Section 4.2.4), also are included in the model map and used for matching. Since many models require particular variable attributes, such as `value type` (for example, *real*) or `value dimension` (for example, *2-dimensional vector*), the variables in the model map may have attribute values which will be used during matching. The variable's attributes specified here should be the most general that the model can accept. Section 5.3 describes

Purpose	Relation	Inverse Relation	Where Used
spatial relations	has-port has-stream has-edge has-end-point isometric	port-of-region stream-of-port edge-of-port end-point-of-edge isometric-r	equipment description
type information	exemplifies	exemplified-by	connects equipment description to type hierarchies
aggregation	has-instance has-part has-spatial-portion has-component has-component-phase	instance-of part-of spatial-portion-of component-of component-phase-of	equipment description and type hierarchies
parameters	describes	described-by	equipment description and type hierarchies
subclass	is-a	can-be-a	type hierarchies

Table 4.1: The relations defined in EDS.

`value-interval-type`, `value-precision-type`, and `value-precision`. Parameters sometimes have minimum and maximum values, for example, when they represent values obtained from sensors, and these minima/maxima are described by attribute functions `value-min` and `value-max`.

One of the characteristics of the textbook examples is that the location and spatial extent of a parameter can determine the boundaries of individuals. The location of parameters is indicated through the `DESCRIBES` relation. The `DESCRIBES` relation attaches parameters to regions, ports, streams, edges, or endpoints, identifying their location and extent. It is expected that more specific information regarding location would be useful for certain models and physical parameters and thus the 11 location attributes shown in Figure 4.4 have been included. Since the geometric representation used is not quantitative, however, experimentation with these attributes is left for future work. When the location attributes do not have values (as is the case for all parameters in this work), the parameter is assumed to describe the entire entity which it `DESCRIBES.` Entities in this representation may occupy volume, may be surfaces, lines, or points, so parameters may have a variety of spatial extents.

The `HAS-INSTANCE` relation is included in EDS for representing views of entities over portions of their lifetime as described in Section 4.2.3. The intention is that behaviors not readily represented by parameters, like phase changes, will be represented with multiple `HAS-INSTANCE` relations, but this has not been tested in the implementations.

4.2.5 Relations in EDS

EDS has a fixed set of 13 relations and their inverses. The 13 relations are listed in Table 4.1. The classes in the type hierarchies could have also been relations in EDS, for example, `PUMP(region-452)`

pressure-sensor-readout-137

time: 76443
time-dimension: 1
time-units: sec
time-type: real
time-interval-type: point
time-precision-type: absolute
time-precision: ±1
time-max: nil
time-min: nil

value: 52.0
value-dimension: 1
value-units: bar
value-type: real
value-interval-type: point
value-precision-type: relative
value-precison: ±5%
value-max: 500
value-min: 0

location: nil
location-dimension: nil
location-units: nil
location-value-type: nil
location-interval-type: nil
location-coordinate-system: nil
location-coordinate-system-type: nil
location-precision-type: nil
location-precision: nil
location-max: nil
location-min: nil

Figure 4.4: The attributes of parameter objects in EDS.

4.2.4 Representing Behavior: Parameters and Instances

EDS provides parameters and instances to represent behavior. Parameters are represented as objects (nodes in the semantic net) like spatial entities. EDS provides 29 unary attribute functions to specify characteristics of parameters. These attributes fall into three classes, attributes of the parameter's time, attributes of its value, and attributes of its location. Figure 4.4 shows these attributes with some example values. In this work, the `time-type` and `value-type` may be integer, real, qualitative landmark, or qualitative sign. The `time-dimension` and `value-dimension` take integers indicating whether the corresponding value is a 1-dimensional (scalar), 2-dimensional, or n-dimensional vector. The `time-interval-type` and `value-interval-type` may be either a point, open, or closed interval. `Time-precision-type` takes values `absolute` or `relative`, indicating that the `time` of the parameter is ± `time-precision` or that `time` is ± `time-precision %` of `time`. The value of the parameter, given by the `value` attribute function, is similarly described by `value-type`,

4.2.3 Representing Parts and Pieces

Some models require an individual to have another individual as a part. For example, the model that calculates net positive suction head for a centrifugal pump (See Section C.3.) requires that the pump has a part which is a single suction impeller. This is the standard *part-of* relation used in conventional type hierarchies (like the `functional-type` hierarchy in this work) to represent subcomponents and aggregates. Since EDS uses four type hierarchies instead of one, it also provides four disaggregation relations. These relations are `HAS-PART`, `HAS-SPATIAL-PORTION`, `HAS-COMPONENT`, and `HAS-COMPONENT-PHASE`. The inverse of each relation is also defined for convenience. `HAS-PART` applies to functional parts, for example, a pump may have an impeller, a shaft, and a casing. `HAS-SPATIAL-PORTION` relates entities that are parts defined along geometric lines, for example, the lower half of Pipe-1 is a `SPATIAL-PORTION-OF` Pipe-1. `HAS-COMPONENT` is a disaggregation relation for materials, for example, air has nitrogen and oxygen components. `HAS-COMPONENT-PHASE` breaks a region along phase boundaries, for example a colloidal suspension has a solid phase component and a liquid phase component.

The disaggregation relations are used in the type hierarchies as well as in the equipment description. In type hiearchies, the disaggregations are assumed to be a complete set. For example, if a `centrifugal-pump` in the type hierarchy has 3 `HAS-PART` relations, one each to `impeller`, `shaft`, and `casing`, then this is taken to mean that there are exactly these 3 parts of centrifugual pumps. In the equipment description, parts are not assumed to be complete sets. The disaggregation relations are allowed in the equipment description so the user may describe equipment at multiple levels of detail or mixed levels of detail. EDS allows the user to decide what detail and how much detail to represent for their physical system. The implemented examples of this thesis make extensive use of the `HAS-PART` relation (and its inverse) both in the type hierarchies and in the equipment descriptions. (See the DETAILED-PUMP, CSTR-1, and CSTR-2 equipment descriptions in Appendix B.) Extensive use of the `HAS-SPATIAL-PORTION` relation and its inverse is made in the equipment descriptions when reconfigurations (Chapter 6) are done, since reconfigurations result in multiple representations of some portions of space.

To make it possible to reason about portions of the physical world over temporal slices of their existence, EDS provides the `HAS-INSTANCE` relation. It is intended to represent *temporal subabstractions* as defined by Lenat and Guha [43]. The `HAS-INSTANCE` relation indicates that one entity, A, is a view of another entity, B, during some portion of B's lifetime. A rapid thermal multiprocessor (a reaction chamber for manufacturing semiconductor devices) may have helium in its chamber in one instance of the interior space of that chamber and may have oxygen occupying the same space in another temporal instance. The examples implemented in this thesis do not require reasoning about different instances of entities, even though some involve parameters changing over time, so use of the `HAS-INSTANCE` relation has not been demonstrated.

ber of a particular class. Pipe-1 participates in 4 `EXEMPLIFIES` relations. The functional type `pipe` does not determine other types. Pipes may be made out of various metals or plastics. The pipe will be some kind of a solid phase, but certain metals can exist as single phase solids or 2-phase solids. Pipes also are all approximately hollow cylinders, but at a more detailed level of representation they have different shapes, for example, some of them have threads and others have flanges.

Why are 4 type hierarchies used instead of the standard type hierarchy (which corresponds to the functional type hieachy in this work)? These kinds of characteristics are deemed necessary based on examination of textbook examples, and these characteristics have the following properties.

Useful Hierarchical Classifications. These characteristics have hierarchical classes which have been identified and named over the centuries by scientists and engineers. Properties may describe all members of a class, for example, metals conduct electricity more readily than other materials. Thus it is useful to describe an equipment individual as being `metal`, even if the specific metal is not known.

Independence of Type Hierarchies. An individual, like Pipe-1, has a range of possible types in each of the 4 type categories as described above which are at least partially independent. When regions are merged and split, as they are during reconfiguration, types will apply independently to the new regions formed. For example, if we conceptually cut a piece out of the wall of Pipe-1 to reason about heat transfer, the piece is not a `pipe` but is `stainless-steel-316`.

Correct Matching on Superclasses. In the textbook examples observed and in the models implemented, the individuals required by models can correctly match any equipment individual which is of the same class or is a subclass of the types specified in the model. Thus the class:subclass relations must exist somewhere in the system. Furthermore, certain models require matching on one or several types independent of the other types. For example, a model that calculates the hydraulic horsepower required by any mechanical pump (one of the examples implemented in this thesis) correctly matches centrifugal pumps and positive displacement pumps, whether the material type is `stainless-steel-316` or `plastic`. Similarly a general model describing electrical conduction through metals correctly matches a metal `wire`, `pipe`, or `tank`.

There may be hierarchies beyond the four identified in this work that are necessary for specifying and matching models. This thesis does not claim that there are exactly four hierarchies. However, these four hierarchies are useful for the models and physical situations observed and implemented in this work. It should be noted here that very little use was made of the geometric type hierarchy in this work because of the limitations on the geometric representations. Future work involving quantitative geometric characteristics and more sophisticated spatial reasoning may reveal a diffferent approach to handling geometric properties.

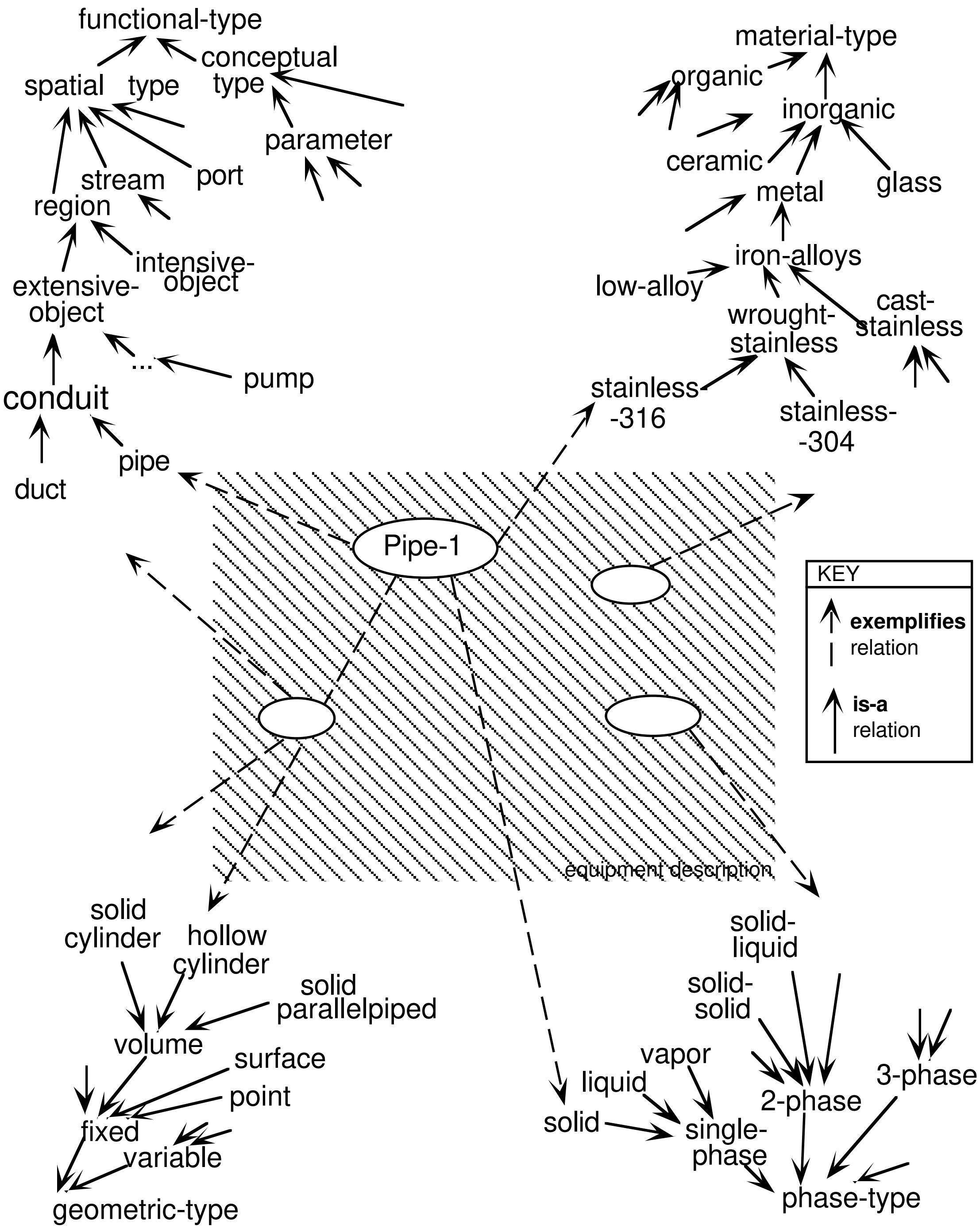


Figure 4.3: EDS uses separate type hierarchies for functional types, material types, phase types, and geometric types.

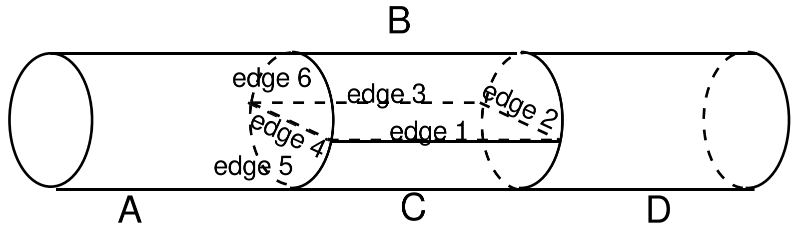


Figure 4.2: A spatial situation that requires edges and endpoints be represented to merge half-cyinders B and C correctly.

the adjacent surfaces of the regions. `Streams` are full-fledged objects in EDS so that subtypes of `stream` may be included in the type hierarchy to distinguish different connections between neighboring individuals. For example, two solid objects may simply abut as in Figure 4.1 or they may be welded together. Since connections are `stream` objects, parameters may also be attached to them to describe characteristics of the connection. The terminology used here (stream, port) is based on Stephanopoulos [72] with the concepts slightly expanded in this work.

Two subclasses of the type `port` are needed to distinguish otherwise ambiguous situations. The use of type hierarchies in EDS (Section 4.2.2) encourages users to divide space along boundaries of phases, materials, and functional component types. However, it is often not convenient to use only those boundaries. (Consider a liquid stream that may flow continuously from beginning to end of a refinery.) The subclass `uniform-port` describes any surface of an individual that does not correspond to a discontinuity. This surface represents an arbitrary division made for convenience of representation. This type of port is necessary to distinguish the situation where, for example, two solid objects which abut are represented as two individuals from the situation where one solid object has been represented as two individuals for convenience. In the first case, there is a discontinuity which may be relevant to some models. The two solid objects have abutting ports that are `nonuniform-ports.`

This spatial representation is one of the limitations in the implementations that will prevent a class of models, those which require quantitative geometry, from being used. Reconfiguration methods that would generate the quantitative geometric individuals also cannot be added to the system as it exists.

4.2.2 Type Hierarchies

The material, phase, functional (or component) type, and geometric characteristics of individuals are used in the textbook examples (Appendix A) to identify individuals for non-component models, along with parameter values. EDS represents these properties by means of four independent type hierarchies. As an example of the independence of these properties, assume that the cylinder in Figure 4.1 is a pipe called Pipe-1. Its functional type is `pipe`. Its material type is `stainless-steel-316`. Its phase type is `solid`, and its geometric type is `hollow-cylinder` as shown in Figure 4.3. The `EXEMPLIFIES` relation indicates that Pipe-1, a physical object, is a mem-

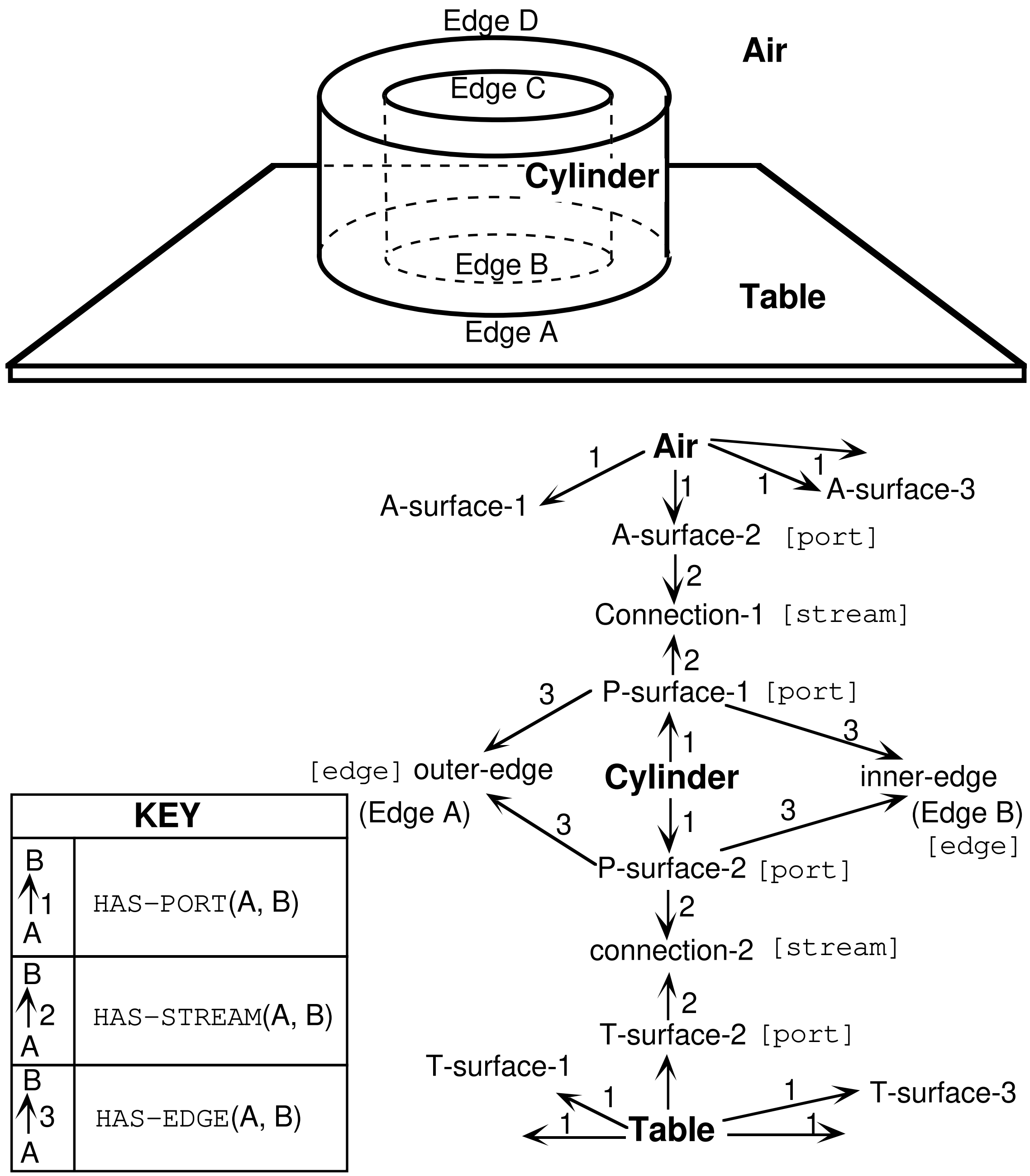


Figure 4.1: The semantic net represention of a cylinder on a table.

4.2.1 Spatial Representation in EDS

The space occupied by the physical equipment is described in EDS by dividing it into pieces or individuals however the user may want to do that. The pieces must fit together to cover the entire space of the equipment. The spatial representation method provided in EDS is limited to the simplest possible representation that would allow the goals of this thesis to be achieved. Since a number of models were found that do not require quantitative geometry and rely only on adjacencies between regions, EDS provides a semantic net representation of spatial adjacencies. Nodes in the net are spatial entities or parameters describing them, and the net is implemented as objects and named directional links in the BB1 system [36,37].

The spatial representation is designed to contain sufficient information to support the merging and splitting operations required for reconfiguration. This requires not only adjacencies between regions, but also adjacencies between surfaces and between edges. Thus, each individual is represented as some number of objects in the net, where the objects correspond to the individual itself, portions of the surface of the individual, edges of those surfaces, and endpoints of the edges. These objects are distinguished by their functional types, `region`, `port` (surface), `edge`, and `endpoint`. (See Section 4.2.2 for discussion of type hierarchies.) Each `port` (surface) object represents an adjacency between this individual and some other individual. The adjacencies between surfaces are curves and are represented as `edge` objects. Edges may have common `endpoints` with other edges. The binary relations between these objects that represent an individual are `HAS-PORT` (relating a `region` to one of its `ports`), `HAS-EDGE` (relating a `port` one of its `edges`), and `HAS-ENDPOINT` (relating a `edge` to one of its `endpoints`).

Portions of surfaces that become `ports` and `edges` in this representation do not correspond to the everyday understanding of surfaces, which are assumed to be smooth, and edges, which are assumed to be discontinuities in the surface direction. Figure 4.1 shows an example of a cylinder sitting on a table top. Such a cylinder is normally thought of as having 4 surfaces, an inner curved surface, an outer curved surface, and the two flat ends. The cylinder in Figure 4.1 has only two `ports`, one representing the portion of the cylinder's surface adjacent to the table, and one representing the portion adjacent to the air. Each of these `ports` have two `edges`, Edge A and Edge B in the figure. The EDS description, also shown in Figure 4.1, has no objects corresponding to Edge C and Edge D.

It may appear that surfaces and the adjacencies between regions would be sufficent to represent space, but when two regions merge, there may be surfaces of the two regions that must be merged to maintain the adjacency semantics. For example, when the two half cylinders, B and C, of Figure 4.2 are merged, the outer curved surfaces must merge into one surface. The adjacencies between these surfaces, as expressed by Edges 1 and 3 in the figure, are required. The end surfaces will also merge into one surface, and the Edges 5 and 6 on the left end of the cylinder must become one edge to maintain the semantics. The endpoints, representing adjacencies between edges, are required to make that edge merge.

The relationships between adjacent regions are expressed as `stream` objects which are linked to

Chapter 4

Representing Equipment and Models

4.1 Summary

The Equipment Description Scheme (EDS) and the Model Description Scheme (MDS) are the representations we developed to embody the characteristics that we observed in 25 textbook examples of *non-component* models. (Appendix A lists these examples.) EDS provides methods for representing distribution of materials, phases, component types, and shapes through a four dimensional space as well as parameters describing portions of that space. Our method uses four extensible type hierarchies to represent *functional, material, phase,* and *geometric* types and 13 relations and thier inverses. We find that at least *functional, material,* and *phase* properties are classified into hierarchies by engineers and that these hierarchies are useful for model-matching. (For example, a model of electrical conduction may apply to a whole class of metals.) We also find that the four types behave differently through reconfigurations and exhibit independence with respect to model-matching. (For example, electricity can conduct equally well through a spoon, a screwdriver, or a wire, if they are made of the same materials. The functional type is irrelevant.) MDS uses the same methods to represent the physical space required by a model, but also describes characteristics specific to models, including classifications of variables (*input, output, causal, affected, carried*), classification of conditions, (*approximation, enabling*), syntactic information (*call forms* and *return forms*), and resources. We also find that models require specifications of entities and relations that are not allowed to be present. MDS specifies these as *negative sets.*

4.2 The Equipment Description Scheme (EDS)

EDS provides a method whereby a user may describe the contiguous space occupied by a physical system. The user may divide the space into individuals at their discretion. For each portion of space, the user may specify the functional type (like `pipe` or `valve`), the material type, the phase type, and the geometric type using 4 independent type hierarchies. These hierarchies form a description vocabulary. Since it is not possible to predict the particular terms or primitives that will be useful for all physical systems, the vocabulary can be varied by the user. The details and motivations for using 4 type hierarchies are given in Section 4.2.2.

textbook examples of model-matching. Several matching mechanisms appear to be required depending on what a model requires in its individuals. The methods implemented can represent and match models where the exact individuals and relations required are specified and where certain other individuals having particular relations are specified to *not* be present. These two mechanisms are called *simple* (positive) and *negative matching* respectively, and are described in detail in Chapter 5. Some models may apply to situations with a variable number of individuals, such as any number of pipes connected together end-to-end. These models require a method for representing patterns in the model descriptions as well as a method for matching patterns to the equipment descripton. This mechanism was not implemented, in the interest of completing the project in a reasonable amount of time.

Similarly, some reconfiguration mechanisms are selected for implementation from those identified in examples. This system can reconfigure individuals that have phase and material specified but no quantitative amount. This system can also reconfigure individuals with a functional type (such as `pump`) from equipment descriptions which do not explicitly represent any object of that type, but do represent the parts (such as `impeller, shaft,` and `casing`) of that component. This system cannot reconfigure to meet quantitative shape and size requirements on regions of specified phase or material, as is required by some models.

The goal of this thesis is to show that the approach of model-matching at diagnosis time could enhance re-usability and extensibility. Since the applicable mechanisms are determined by the characteristics of the model, and since only a limited number of models could be implemented, then only mechanisms required by the models being implemented for demonstrating re-usability and extensibility were included. The other mechanisms, some of which may not yet be identified, are left as future work.

of space. Likewise, it has been assumed that the situation in which a model applies is a continuous portion of space. This space may be broken up into any number of indivduals, but each individual is assumed to be *contiguous* with some other individual in the model description. Component models used in the domain of chemical processing, such as models of pumps and reactors, often meet this assumption. The textbook examples of non-component models (See Appendix A) also required individuals that met this assumption. Chemical processing plants often meet this assumption in that all the processing units are connected. This method would not be appropriate for diagnosis of a whole semiconductor manufacturing facility, because the individual processing steps are often not connected. Wafers are carried from one piece of equipment to another. However, the approach is applicable to each piece of equipment within the fabrication facility.

Macroscopic Properties Assumption. We also assume that no reasoning at the molecular level is required. The system has no means of transitioning from reasoning about a region filled with gas to a region of balls (molecules) moving through empty space. When such a transition occurs, parameters such as temperature and pressure which are macroscopic properties no longer make sense. The reconfiguration methods developed here assume that macroscopic properties apply to the regions created by merging and splitting and thus that the size of the regions required is never smaller than the number of molecules necessary for macrosopic properties to apply.

3.4.3 Limitations due to Implementation

Other limitations of the implementations are a result of the need to limit the length of the overall project, but still be able to show feasibility of this approach. One such limitation lies in the spatial reasoner, the portion of the code that merges and splits adjacent regions. The represention used is a semi-quantitative semantic net which makes explicit adjacencies between neighboring volumes of space, but allows only limited quantitative descriptions of those spaces. Chapter 4 gives more detail on this representation. Models that specify individuals using quantitative geometries cannot be accomodated in this implementation. The system does include models whose individuals have shape implied by the functional type, as the shape of a pipe is implied by its type. The system can also work with models whose individuals' shapes are determined by adjacent components, such as the fluid inside a portion of a pipe.

We investigate limited kinds of changes occuring in the equipment description. In the examples implemented, no adjacencies of individuals either appear or disappear over time, other than through the merging and splitting done by the reconfiguration algorithms. In some examples, the size of the surface representing the adjacency between two regions changes, but does not completely disappear. Clearly these kinds of changes occur in chemical processing systems as well as in situations where mechanical motions cause the adjacencies to change. Further experimentation is needed in this area and could require revision of the representation or the matching and reconfiguration algorithms.

We implement selected matching and reconfiguration algorithms from the set identified in the

Model Matching Implementation (approx. 500 K total, 9,000 lines)	
Value Comparison (for parameter matching)	28%
Spatial Reasoning	38%
Match & Reconfigure	28%
Generate Model Test Set, Check Model Conditions, Execute Model	6%

Table 3.1: The model-matching system's implemented components and sizes.

3.4 Scope of the Solution

3.4.1 Implementations

The algorithms for matching and reconfiguration are implemented in Common LISP within the BB1 system [36,37]. BB1's architecture is a blackboard framework that provides a sophisticated object oriented representation language as well as the means to opportunistically execute operations based on changing conditions within a system. The object oriented capabilities of BB1 include the ability to create and use different kinds of relations on objects (beyond the standard subclass relations) and to selectively inherit attributes and relations. Since diagnosis is very opportunistic, selecting what to do next based on the conclusions derived so far, the BB1 architecture provides an ideal framework for experimenting with the model-matching system built for this thesis in the larger context of diagnosis.

The algorithms for matching and reconfiguration consist of about 9,000 lines of LISP code (approximately 500 K) which use object manipulation functions provided by BB1. Table 3.1 shows the various capabilities implemented and their percentages of the 9,000 lines of code. Each of the 16 test cases made for the demonstrations of re-usability and extensibility involves a model description (as well as the model itself) and an equipment description. Eleven different equipment descriptions and 8 model descriptions were built to make these 16 test cases. The average equipment description is about 115 K and the average model description is about 40 K, adding up to another 1.5 M of code.

3.4.2 Underlying Assumptions

Two assumptions underlie our method, and thus inherently limit the situations in which it is appropriate.

Contiguous Space Assumption. The first assumption regards the space occupied by equipment and by model individuals. We assume that the equipment occupies one continuous volume

3.3.3 Inputs/Outputs of Model Matcher to a Diagnosis System

The inputs to the model matcher developed in this thesis are

1. a physical parameter,

2. its spatial extent,

3. and one of `[causes, effects, values]`
   (the diagnosis system's goal with respect to the physical parameter.)

The physical parameter and its spatial extent are objects in the object-oriented equipment description. They act as a starting point for the model matcher, which is attempting to identify individuals and find the necessary relations between those individuals in the physical context surrounding the input parameter. The goal may be one of three possible goals, find `causes`, find `effects`, or find `values`. The matcher attempts to match only models that present an appropriate causal relation to the input parameter. If the diagnosis system is searching for values, perhaps as input for some other model, then only models that can calculate the physical parameter are considered.

The outputs of the model matcher are

1. the relations on the physical individuals required by the matched model (these are instantiations of the model's relations on the physical entities matched),

2. the calculated values generated by the model embodied as physical parameters in the equipment descripton,

3. and a list of applicability conditions.

The relations returned are the manifestation of the explicit mapping of model individuals, parameters, and relations to their physical analogs. Some of the physical entities specified in the relations may be created by the model matcher as a side effect, if a reconfiguration is required to get a match. However it does not destroy existing individuals in this process. These relations must be satisfied for the results of the model to be valid, and these are the inputs to a truth maintenance component. If the values calclated by this model are later found to conflict with measured values, these relations are used to generate diagnostic hypotheses about what could faulty. The output parameters generated by running the model are returned explicitly as a list, since dependency tracking for these values will be done by the diagnosis system. By side effect, the parameter objects for outputs are created and installed into the physical description at the location corresponding to where model outputs were matched. The last output is a list of conditions that the model requires to be met to ensure applicability, but which could not be tested because of lack of data. Rather than disallowing use of the model when applicability data is unavailable, these conditions are passed to the diagnosis system which may treat them as assumptions.

z at time t. These axioms, when combined with some facts about the physical system such as

$$ANDGATE(A1),$$

$$ANDGATE(A2),$$

and

$$CONN((OUT, 1, A1), (IN, 2, O1)),$$

and a fixed set of constants like $A1$ and $A2$, effectively yield a static network of models. These axioms and facts provide the correspondence between models and constituents by stating, for example, $ANDGATE(A1)$ and then including $ANDGATE(x)$ in the antecedant of the axioms that embody the model for AND-gates. The correspondence of model variables to physical parameters is implied by axioms through the specification IN or OUT and the number of the input or output. The correspondence is not stated explicitly, but assumed to be obvious once the physical constituent, such as $A1$ or $A2$, is stated.

3.3.2 Dynamic Virtual Network of Models and Variables

Our method produces the same three elements used by previous diagnosis systems, the network, the correspondence between models and physical constituents, and the correspondence between model variables and physical parameters. There are two differences between the network produced in this work and the network used in previous work. The first difference is the variability of the physical constituents. Since portions of the equipment that are identified as constituents may change, the network can take on many different forms. The second difference is that the network is virtual and dynamic, in that the only portion of the network that actually exists is the piece being used by the diagnosis system at the current time. Thus the approach is much like that of paging and virtual memory used in operating systems.

One of the functions that a network of models and variables tacitly performed is a representation for the physical equipment showing how components are connected and where the parameters existed. The method developed here uses a representation of four dimensional space to show component and parameter locations and then produces the expected network of models from that representation. It is expected that a diagnosis system would use this representation to find information it needs about the equipment rather than the previously used network of models.

The method developed here explicitly provides both the correspondence between models and constituents and the correspondence between model variables and physical parameters when it produces a new node in the virtual network. A successful matching of a model to the equipment description produces a new node. The relations required for a model to be applicable are explicit in the description used by the matcher and are available to the diagnosis system for dependency tracking and hypothesis generation. The model descriptions in this thesis embody many more relations than were typically associated with models in previous work ($isa(c_i, type_j)$), but the additonal relations may be ignored by a diagnosis system.

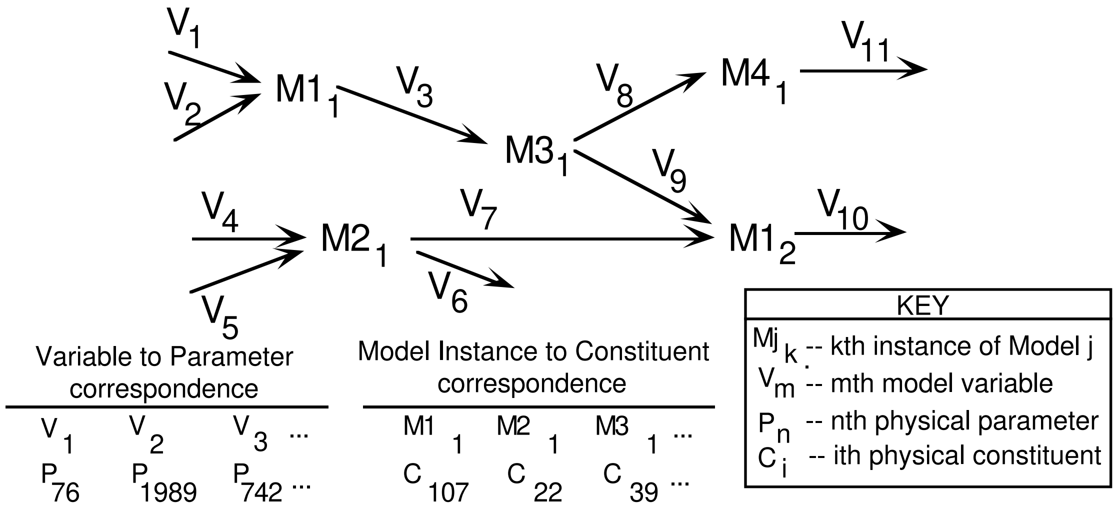


Figure 3.1: Abstract view of a static network of models as is typically used in model-based diagnosis systems.

[13,66]. Figure 3.1 illustrates the network of models and the correspondences of model variables to parameters and of model instances to physical constituents. The values calculated by models are compared to physical parameters to generate discrepancies. Each model that participates in generating discrepant values also contributes some (usually just one) relations which that model requires for its results to be valid. Typically the single relation was that the component corresponding to the model was functioning correctly. This relation is often expressed as either $isa(c_i, type_j)$ or $normal(c_i) \wedge type_j(c_i)$. These relations are used by a diagnosis system to generate diagnostic hypotheses. Each model also may have a causality specification with it, so that models used to generate discrepancies could be selected in portions of the network where the fault might be.

In logic-based approaches [18,31] where theorem provers are used to generate diagnoses, a static network of models was not built explicitly but was still effectively achieved through the form of the logic sentences which embodied the models combined with the fixed set of constants that corresponded to physical constituents. For example, consider some of the axioms in Genesereth's DART system [31].

$$ANDGATE(d) \wedge VALUE((IN, 1, d), t, ON) \wedge VALUE((IN, 2, d), t, ON)$$

$$\longrightarrow VALUE((OUT1d)tON)$$

This axiom states that if d is an AND-gate and the values on its two input lines are both ON at time t, then the value on its output line is also ON at time t.

$$CONN(x, y) \wedge VALUE(x, t, z) \longrightarrow VALUE(y, t, z)$$

This axiom states that if x and y are connected and x has value z at time t, then y also has value

3. The five kinds of characteristics are at least partially independent of each other. (For example, a component such as a pipe can be made of many different possible materials.) The characteristics behave differently through different mergings and splittings of entities, and thus are independently represented and manipulated.

4. The physical individuals and connections between them required by each model are made explicit to allow the individuals to be identified in the equipment description at diagnosis time. (A matching procedure triggers "cutting and pasting" when the match fails, if individuals required by the model can be made from the individuals existing at that location in the equipment description.)

Our approach requires three elements be available in a knowledge base.

- a set of models
- a model description for each model
- equipment description

These elements are described in detail in Chapter 4. The approach uses two kinds of algorithms for identifying individuals for models.

- matching
- reconfiguration

Reconfiguration algorithms are called by matching algorithms when a match between a model description and an equipment description has failed. The reconfiguration algorithms evaluate the individuals present at the point of failure in the equipment description to determine if the desired individuals can be made from them. If so, the individuals are reconfigured, and the matching algorithms continue, attempting to complete the match. Chapter 5 describes matching algorithms, and Chapter 6 describes reconfiguration.

3.3 How Model-Matching Fits into Model-Based Diagnosis

Diagnosis systems have used static networks of models. Our approach is to generate part of that network when it is needed. This dynamic approach allows different networks to be constructed for the same equipment, when the data available warrant (and thus make it possible to use non-component models) while still presenting the same information to the diagnosis system.

3.3.1 Static Network of Models and Variables Previously Used

In previous approaches to model-based diagnosis, models of the constituents (component or process) of a system were linked together via their input and output variables to form a network of models that corresponded to the components and parameters of the physical system to be diagnosed representation and spatial reasoning capability sufficient to handle the spatial requirements of the selected algorithms and models.

3.2 Overview of Solution to the Model-Matching Problem

Our approach is based on characteristics observed in engineering problem solving where non-component models, like the Ideal Gas Law, are used. The focus is on non-component models because they present a bigger challenge for matching and because they have not been used significantly in previous systems. The observations are based on five textbook problems from each of five different areas, heat transfer, fluid mechanics, mass transfer, thermodynamics, and reaction kinetics. These 25 problems are listed in Appendix A. The findings are that the individuals used in these problems were portions of contiguous space identified by combinations of up to 5 characteristics.

- phase
- material
- parameter values or locations
- geometry
- boundaries of components

Identifying the individuals to which such models apply in advance of diagnosis time is not feasible because there are too many possible choices as described in Section 2.4. Since parameter values, which arise at diagnosis time, play a role in identifying the boundaries of individuals, the boundaries are underspecified until those values are available.

Our solution is to describe the space occupied by the equipment and separately describe the individuals required by each model. Descriptions of the equipment space and model individuals include the five characteristics identified above. At diagnosis time, when parameter values appear, matching algorithms use model descriptions and the equipment space to identify appropriate pieces of the space as individuals. The process of identifying individuals within such a descripton of space is refered to as *individuation.*

The following principles are used in this work to make individuation at run-time possible:

1. The equipment representation partitions the 3-dimensional space occupied into entities, and the matter, energy, and connections amongst the entities are described. Since it is a partition without arbitrary holes, it is possible to cut the individuals required out of that space at diagnosis time. If necessary, the equipment entities may be "cut and pasted" into individuals that do not correspond to either parts or aggregates of the original entities.

2. Engineering models typically divide the physical world into entities according to (at least) the five identified characteristics: materials, phases, parameters, geometry, and components. These characteristics are described for each entity and methods are provided for splitting or merging entities to make different individuals, while correctly maintaining characteristics.

Chapter 3

An Approach to Automated Model Matching

3.1 Summary

Our approach to the model-matching problem is to represent the 3-dimensional space occupied by the equipment, the characteristics of the space required by the model, and then match them at run-time, possibly reconfiguring the equipment description to match. Reconfiguring is a process of merging and splitting the individuals in an equipment description, preserving the substance but changing the form. Our method requires explicit descriptions of model requirements, a separate description of the space of the equipment, and algorithms that match and reconfigure. We have isolated the problem of model-matching from the larger problem of diagnosis, to focus our study and resources on this problem. The interface to a diagnosis system consists of the inputs to a matcher, the outputs, and some globally shared representation for the equipment. The inputs are an equipment parameter (which is part of the global equipment description) and a goal (one of *find causes*, *find effects*, or *calculate values*) with respect to that parameter. The outputs are the values calculated by an appropriate model (We are doing the *behavior prediction* step from Figure 2.3.), the relations on equipment entities that must hold for the calculated values to be valid, and any conditions on parameters which could not be tested at the time. The network of models used in previous work had another role, though, besides that of providing the behavior predictions and conditions returned by the model matcher. It was also used as a representation of the structure of the equipment, which is necessary for other tasks within the diagnosis. In our approach, the diagnosis system is assumed to have access to the equipment description to obtain its structural information.

Our methods are limited by their underlying assumptions as well as by the limited implementations and testing. We assume equipment individuals occupy contiguous space and likewise for model individuals. We also assume that macroscopic properties such as temperature and pressure apply to all individuals, and so cannot transition to reasoning about matter as balls (molecules) moving through empty space. We implement matching and reconfiguration algorithms in Common Lisp and build 11 different equipment descriptions and 8 model descriptions to generate 16 different implemented cases of model-matching. The algorithms and various descriptions are about 2 M of source code, 1.5 in the descriptions for the examples and 0.5 in the algorithms themselves. To limit the scope of the work, we select for implementation a subset of the algorithms we identified as necessary to solve the general model-matching problem. We also provide limited geometric

or chemical reaction which are composed to represent the units and ultimate larger portions of a chemical plant. These authors have developed a diagnosis method that uses the process-oriented QPT representation of Forbus [25]. They include QPT processes representing correct behavior as well as faulty behavior. By activating and deactivating the possible process models for a unit to get model predictions that are consistent with observations, faults can be identified. Even though units are not the fundamental individuals used in this work, the use of QPT requires that individuals be defined and process instances assigned in advance, as was discussed in Section 2.5.1.

Saxena and Unruh [65] apply model-based diagnosis directly to semiconductor processing equipment. In their work, they compare the use of detailed quantitative models of the equipment called *response surface models* (RSM) to qualitative simulation models using QSIM [42] and a simple symbolic model that associates the direction of change in wafer state parameters to processing conditions. Their models are built to describe the whole piece of equipment being diagnosed. They do not address the problem of matching models or individuating.

2.5.3 Diagnosis in Chemical Manufacturing

The use of non-component models and thus dynamic individuation is likely to appear in diagnosis of fluid processing and chemical manufacturing. On surveying this literature, we find that most work divided the processing equipment into *units*, the term commonly used by chemical engineers to refer to parts of a processing plant at the level of columns, reactors, and tanks. This view of a processing plant is a component-connection model like that used in the work on digital circuit diagnosis.

The earliest use of model-based diagnosis for fluid processing systems was the work of Scarl *et al* [66] whose system diagnosed failures in the liquid oxygen delivery system for the space shuttle. The system contained a network of functional relations that described the behavior of the various components. By propagating values through this network and comparing calculated values to measured values, faults could be isolated. Kramer [41] used quantitative constraints to model the units in a chemical plant. Associated with each constraint was a set of faults that could cause the constraint to be violated. Because of this association, the approach is not strictly model-based. Rich and Venkatasubramanian [62] built a system called MODEX which used qualitative models for units as well as for fluid flows (streams) connecting them. They later combined rules with the model-based system to improve efficiency [76]. Gallanti *et al* [29] used equation models for steam handling units in a power plant, applying a form of Iwasaki's causal ordering technique [39] to link the equations and identify causes of faults. For such a linking to be possible, the variables in the equations have to be implicitly associated with the real variables (and the locations of those variables) in the physical system. Ng [53] used qualitative simulation models based on QSIM [42] for the components in a proportional-control heater/cooler. By using models that simulate over time, they can detect some kinds of faults not previously diagnosable. However, they explicitly state that their approach rests on the assumption that the continuous system can be modeled adequately by a component-connection model and that each qualitative constraint can be localized to one component. Oyeleye *et al* [54] modeled units in a chemical processing plant using *extended signed directed graphs* as models of the units. This work probably does not meet the definition of *model-based* because the directed graphs were a representation of causality, rather than simulators of behavior. However, their method, like all the others surveyed, relies on an individuation of the system done at system building time along unit boundaries.

The work of Grantham and Ungar [32] is distinguished from the other work in chemical processing because it does not use *units* as the individuals. It relies on processes such as heat flow processing steps are the fundamental individuals to which models may be assigned. Mohammed and Simmons do not represent the 3-dimensional space for the processing equipment, but they do use a simplified spatial representation of the layers on the wafer. Funakoshi and Mizuno [28] develop a similar approach to simulate the effects of process steps on wafers, but their models are sets of rules. Slaughter *et al* [70] developed a system that was inspired by the work of Mohammed and Simmons, but that uses quantitative analytic models for process steps instead of qualitative models. They use an object oriented approach and hand code the dependencies for each model into *methods* (procedures) which record dependencies as the simulator runs. Dishaw and Pan [20] make numerous runs of quantitative simulators to generate data from which diagnosis rules are extracted. In later work, Mohammed develops methods for combining model-based with rule-based reasoning [49]. In that effort, the focus is on carefully selecting the level of abstraction and granularity in the representations so that both methods work synergistically. None of these researchers represent the space occupied by the equipment nor individuate, but they solve a problem that our approach could not solve. We assume that the equipment about which we are reasoning occupies a particular region in space over time. There is no obvious way to map the situation of a wafer being subjected to many process in many locations into our paradigm.

Freeman [27] applies model-based diagnosis to a different problem within semiconductor manufacturing. When wafers are complete, the circuits on them are tested electrically to determine if they have the desired properties. If some measurement is abnormal, the problem is to identify which structure or layer on the wafer is responsible for the problem. Freeman's system incorporates analytic models of the relationships between measured parameters and properties of wafer layers. Freeman builds the network of analytic models required for any particular set of given measurements at run-time, and is thus doing a form of model assignment. The system maintains a symbolic hierarchy of what structures are contained by what other structures. It also contains definitions of variables which are uniquely associated with a kind of structure. Models then use these defined variables. A network of models is built for a particular set of measurements starting from the variables that are associated with the given measurements. All models are included that address these variables, and the network is filled out with models required to relate the measurements to structural parameters. Because one can define in advance the parameters that one will see, these parameters can be uniquely associated with the places where they occur. Freeman does not need any representation of 3-dimensional space. Even though our methods were not envisioned to work in a situation like this, their fundamental assumption of working in a particular region of space is not violated. However, if one can predict the parameters one will use including their locations, then our approach is not useful. It introduces complexity that simply is not necessary. The predictability is probably a result of the kind of diagnosis. A large number of wafers which are very similar are examined. There is a prescribed set of measurements that will are made. The physical system, the wafer, is being examined over a relatively short period of time in a relatively stable environment. These characteristics are significantly different than are encountered when reasoning about a piece of manufacturing equipment, our intended area of application.

[45] treats as a bridging relation, we would treat is a model, the relation between current and charge carrier density. As Liu points out, the two ontologies carve the world up into different granularities. Our view is that no small number of these particular ontologies are sufficient to reason about the real world. Therefore our approach attempts to generalize what these researchers have done for special cases. By providing the full 3-dimensional space (and also time) and by providing extra information with each model about what it needs, we can carve the world as the particular task, goals, and model demand. The carving methods include our type and parameter transformations as well as reconfiguration methods. These appear to be at higher level of generality because, for example, a single reconfiguration method (like our implemented *intensive* reconfiguration or unimplemented *extensive* method) could cut an MC out of liquid or could cut any other necessary size of individual.

2.5.2 Diagnosis in Semiconductor Manufacturing

Some example problems that inspired this thesis were found in the semiconductor manufacturing domain. Semiconductor manufacturing involves a number of processing steps where fluid flows, heating, cooling, and chemical reactions are carried out to form layers on silicon wafers which ultimately become the chips upon which computers are built. Because of the nature of the processes involved, this domain can require the use of non-component models during diagnosis. Much fluid processing is done in this domain as in the chemical manufacturing domain, but semiconductor manufacturing is distinguished by the discrete nature of the steps involved. A wafer is placed in a number of different fluid processing environments through the manufacturing sequence. In the chemical processing domain, fluids tend to be handled in a more continuous manner. The literature and research in these two domains tend to be separate from each other, and thus we consider related work in semiconductor manufacturing as well as the chemical processing domains. We have surveyed the literature directed at diagnosing semiconductor manufacturing to find any information relvant to model-matching and individuating. A number of approaches to computer aided diagnosis have been investigated with respect to semiconductor manufacturing including evidential reasoning (for example belief nets), neural net pattern detection, rule-based approaches, and model-based approaches. We restrict the discussion here to approaches that involve predictive models of the physical equipment.

In semiconductor manufacturing, a wafer which is initially a thin disk of silicon, goes through many different processes that occur in many different pieces of equipment. Mohammed and Simmons [50,68] address the problem of causally linking characteristics of the layers on the wafer to processing steps and conditions in the processes. Their approach is to symbolically simulate the wafer going through the process steps. At each step, dependencies are recorded as to what process and condition is responsible for what portion of the structure developing on the wafer. The causal history that is built up can then be used for finding causes for abnormalities. Even though this approach would not be classified as model-based diagnosis, it uses models to simulate process steps. The models are matched to the equipment by the type of the process step involve, similar to the way in which models are assigned in the work on diagnosis of digital circuits. Here the ways. Collins and Forbus [7] introduced the Molecular Collection (MC) ontology which is based on Hayes' [35] notion of *piece of stuff*. An MC is a piece of stuff that is small enough to always be in one "place", but large enough to have macroscropic properties such as temperature. The MC is important for reasoning about thermodynamic cycles such as refrigeration cycles where one analyzes what happens to a portion of the refrigerant as it flows around its cyclic path. MC is built on top of a QPT representation of substances (liquids and gases) in containers. Since MC is always in one "place", it is never partially in one container and partially in another. Rules attached to each process (the *models* within QPT) generate properties for MC and tell how MC is affected by the process, for example that MC will move into the next container.

Rajamoney and Koo [60] develop a method, also supported by QPT, that can explain evaporation of alcohol at the surface of a liquid into the surrounding air. Their approach separates from the liquid a small number of particles which are representative of the larger group, predicts effects of interactions of the particles, then extends the results to the whole liquid. They provide specific relations for moving back and forth as well as specifying what additional quantity types and individuals (as used in QPT terminology) appear and disappear. Our methods could not handle the particular transition made to reasoning about molecules as balls that move around in empty space, because our methods asume that macroscopic properties such as temperature are meaningful. Bylander and Chandrasekaran [5] predict behavior of composites from the components of a system. They use structural primitives for both connections and containments. Substances move from one containment to the next. The containers can be joined into composite containers, and causal patterns specify behavior for composites from their components.

Liu and Farley [46] and later Liu [45] develop the Charge Carrier (CC) ontology that describes charged particles flowing through wires. CC is an alternate ontology for the analog circuit components such as capacitors and resistors. It is very similar in intent to the MC ontology, but resides in the domain of electrical flow. Different primitives, both quantity types and individuals, are defined for the CC and component ontologies. Bridging relations convert from one representation to the other. For example *current* in the component ontology converts to *charge-flow* in the CC ontology. For CC, the notion of a *region* is defined as a structural unit between two poles, through which charge carriers can flow. They can be connected in series or parallel and any number can be defined within the space of a component. They may also be hierarchically defined, with regions having subregions. The regions are all cylinders described by length and cross sectional area. Unlike our approach, though, these regions are defined and not cut out of some complete 3-dimensional space. These regions provide the foundation for the bridging between the two ontologies. Liu [45] states that there is not a one-to-one mapping between the terms in the two ontologies because they carve the world in different granularities.

In all of these approaches, the individuals are defined in advance either through the use of QPT or through the specific terms provided in the ontology pairs. However, in all cases, space is being used to relate the two ontologies to each other. MC has its "places", the other fluid ontologies [5,60] have their containers, and CC has its regions. In each case, the representation of space is just strong enough to meet the needs of the two ontologies. Another noteworthy point is that what Liu

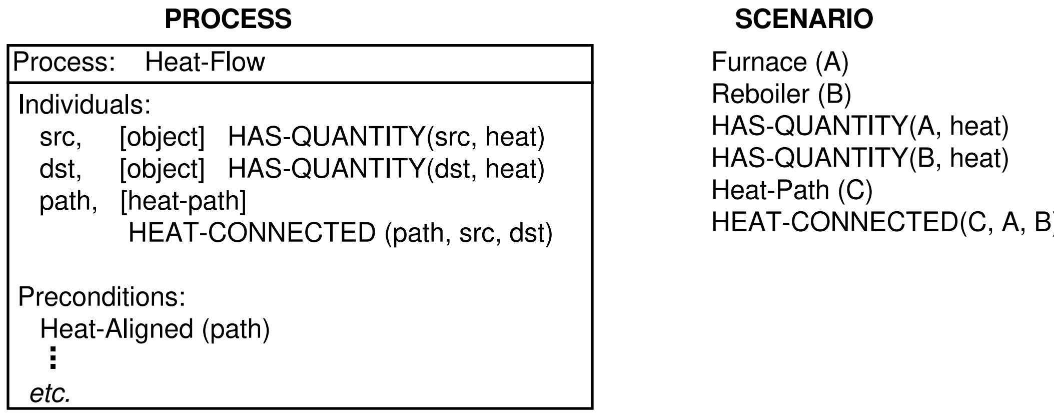


Figure 2.5: An example of a *process* and a *scenario* represented in Forbus' QPT [25].

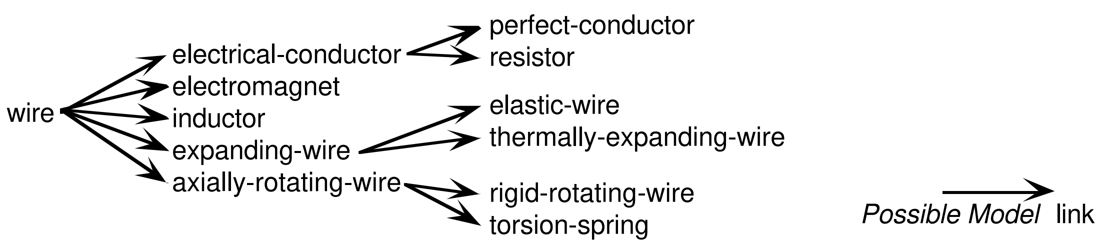


Figure 2.6: An example of a primitive component (wire) and some of its possible models as defined by Nayak [52].

which represent all the ways a component may be modeled. An example is shown in Figure 2.6 for the possible models of a wire. This organization according to components results in an a priori definition of individuals as in digital circuit diagnosis.

Several researchers have extended the results of Nayak. Rickel and Porter [63] present methods for system boundary definition and time scale selection when answering queries about plant physiology. Iwasaki and Levy [38] extend Nayak's method to cases where simulation over multiple states is required to answer the query. Forbus and Whalley [26] built a tutoring system that combines compositional modeling, qualitative representations, truth maintenance, and constraint propagation to explain thermodynamic cycles. All of these researchers use either QPT or a very similar underlying representation and are thus forced to define individuals a priori as in Figure 2.5.

Besides defining a fixed set of individuals, these approaches differ from our approach because they require specific internal representations and calculation methods within the models. The models that are composed all contribute equations (of one or several specified forms) to a simulator. Thus only models that are written in the proper form for the simulator can be added.

Shifting Ontologies

The body of work on shifting ontologies is the closest to our own. This work is not directed towards matching models for diagnosis but is addressing the problem of carving the world up in different extend the use of the graph of models by using time constraints to select models and using belief nets to enable model selected even when none of the models fully meet requirements for accuracy or time. Weld [77] develops methods for comparing models with respect to their predictions for a physical system *over time* and then later [79] incorporates those techniques into a graph of models. Other efforts have been directed at producing various models dynamically. Weld and Addanki [78] collaborated on methods to generate approximate models by choosing appropriate simplifications for the various components being modeled. A number of researchers [57,47,12,40] have investigated order of magnitude reasoning to simplify models under various circumstances.

The efforts directed toward model selection all have in common that they deal with one entity. In [1] the graph of models is applied to a gear train transmission. Every model in the graph addresses the same portion of the physical system, the whole gear train transmission. The problem of identifying individuals and matching models to physical systems is not addressed here.

Compositional Modeling

Recent research addresses the problem of modeling large systems by composing models of parts of the equipment. When modeling large systems, complexity of the model can become prohibitive, so the efforts in the area of compositional modeling attempt to use the simplest model possible for each part of the physical system. Falkenhainer and Forbus [23,24] investigate compositional modeling within the context Qualitative Process Theory (QPT) [25]. QPT provides a process-oriented qualitative representation for representing physical systems. Falkenhainer and Forbus' system composes models in response to queries. To determine how much of the system to model for any query, they rely on defined *operating blocks* which identify parts of the system (a steam power plant) that must be represented at a uniform level of detail. They also require that enough parts be modeled, so that the equations contributed by the models for the parts can answer the query. In the later work [24] they define systems and subsystems to indicate which parts of the physical system must be modeled as units. Before analysis can begin, they "fully instantiate" their set of models (called *domain model*) on a description of the equipment (called *scenario*). This process is a matching of each model to every possible place that the model can be used within the equipment. Figure 2.5 shows an example of a partial model (called *process*) of heat-flow with a scenario. A process can be mapped onto the scenario at every place where it has individuals satisfying the predicates (such as `HAS-QUANTITY`) specified under `Individuals` in the process. The mapping is not as straight forward as in the component centered approaches, but there is still a finite mapping of processes onto the scenario. Even though models are in some sense being matched to equipment, all the individuals are defined in advance. `A`, `B`, and `C` in the scenario of Figure 2.5 are the individuals. This contrasts with our approach.

Nayak [51,52] has also investigated compositional modeling. Nayak's work introduces the use of context, both structural and behavioral, as well as the expected behavior to select appropriate models (called *model fragments*) for each part of a physical system. Attempting to answer queries, like Falkenhainer and Forbus, Nayak defines criteria for generating the *simplest adequate* model to answer the query. Nayak organizes the model fragments into trees of *context dependent behaviors* as well as fault behavior for components in analog circuits. Models are tied to the type of analog component, so these types predetermine a single individuation. Struss [74] develops a theory of relations between models and how a model switch affects a diagnosis. The models are grouped accordling to the type of component to which they apply, such as breakers or lines in Struss's power transmission network domain. The relations between various models of the same component are defined and used to switch models under various circumstances. This work explicitly states that a fixed set of components is assumed to be given, but the component-wise classification of models also only requires an individuation according to component type. Bublin and Kashyap [4] directed energies toward building a multi-level model of a carburetor which allows focusing and control within a diagnosis. This single model is used as the representation for a physical carburetor, and the individuation and matching are done implicitly at system building time like the other approaches described. Bonarini and Sassaroli [3] introduce processes into a system that diagnoses trash burning plants, but these processes are defined over a fixed set of component types.

All of these papers, with exceptions in [34] and [13], make the assumption that a component view of the world and the single individuation which that entails is sufficient. Hamscher [34] and Davis [13] both use some representation of the physical location of a component, that it is on a particular board, DIP, and pin. Unlike our work, they do not retain the 3-dimensional space within their representation, nor redefine individuals at run-time.

Model Selection

Weld [79] defines 4 independent dimensions within the model selection problem.

1. Scope (for example, the refrigerator vs. the refrigerator's compressor)
2. Domain of Applicability (the ranges of inputs and outputs over which a model is valid)
3. Precision (for example, qualitative ranges vs. integer values)
4. Accuracy (how closely the model reflects physical reality)

Of these four dimensions, investigation of scope appears to be the most likely area in which the problem of individuation would have arisen. However, we find no previous work that directly addresses scope. Scope is addressed, usually as a side issue, in the work on compositional modeling discussed in the next section. The work addressing model selection is generally focused on accuracy.

The *Graph of Models* formalism of Addanki *et al* [1] is a seminal work in the area of selecting models for accuracy. Relations between models correspond to changes in assumptions between the models. For each relation, the changes in parameter values that would result if one were to use one model instead of the other are also specified. Given some starting point in the graph of models and some observed parameter values, one can compare the values to those predicted by the given model, and then identify the most accurate model using the relations between models. In a graph of models, all the models describe the same entity, such as the gear train transmission in [1]. In selecting among physiologic models of the heart and lungs, Rutledge and Shachter [64] diagnosis with more than one suspendable relation per individual (or per model) are left for future work.

2.5 Related Work

In surveying the literature, we find that the problem of individuating and matching models to some physical system has not been addressed before. We investigate several areas of the research literature where the matching problem might have appeared, including artificial intelligence, computer-aided diagnosis in semiconductor manufacturing, and in chemical manufacturing. The diagnosis task requires that a system be able to deal with unanticipated problems, and thus would be the most likely task in which researchers would address individuating or reconfiguring on the fly, but we also investigated other tasks involving modeling of physical systems. The research nearest to ours with respect to making individuals were some efforts addressing shifting ontologies for physical system modeling. These researchers use some aspects of a 3-dimensional space to map between (usually) 2 specific ways of carving up the world. Our work generalizes these approaches by using a whole 3-dimensional space and letting the situation (parameter values available, diagnostic goals, models available) determine how to carve the world into individuals. Our view is that there are *many* right ways to carve up the world, but the appropriate one for a problem cannot be determined in advance.

2.5.1 Artificial Intelligence

Multi-Model Diagnosis

Within the diagnosis literature, we focus on diagnosis systems where more than one model could be used at a given point in the equipment. A disproportionate amount of this work addresses digital circuit diagnosis. Davis [13] uses models of the behavior of circuits and gates as well as models of their physical locations, but instances of the models are linked together at system building time to represent the equipment. Genesereth [31] works with multiple levels of detail, reasoning about adders and mulitpliers as well as individual gates, but all the levels must be in place before diagnosis starts. Hamscher [34] uses both functional and physical representations of circuits, but defines primitives in both representations. The equipment is represented using these primitives, which have been designed for efficient troubleshooting and repair. The primitives limit what may be an individual. Struss and Dressler [75], de Kleer and Williams [19], and Raiman *et al* [59] expand the theory of diagnosis to use *fault* models, models which simulate behavior of a particular malfunction in a component. Each type of component has a model of correct behavior as well as some number of fault models. The types and locations of all components are identified in advance. The type of the component defines the set of models applicable. The types act as primitives, defining in advance what can be an individual.

Some multiple model diagnosis has been done outside of the digital circuit domain. The DEDALE system of Dague, Raiman, and Devès [10] uses multiple models for correct behavior

Definition 2.2 (The Extended Diagnosis Problem) *Given:*

1. *4SPACE, a body of contiguous three-dimensional space defined over time interval that is occupied by the physical system (It may be moving and may be varying in size or shape over time.)*
2. *ED, the equipment description, is a representation of the arrangement of matter and energy occupying 4SPACE via spatio-temporal relations on portions of that space*
3. *M, a set of models with specifications of the matter, energy, space, and time characteristics of the individuals required by those models*
4. *OBS, a set of observations, which are values for parameters having particular space-time locations and extents in ED,*

identify the relations within ED that must be suspended for OBS to be consistent with values generated by M.

This definition follows the definitions used in consistency-based approaches as described by Console and Torasso [8] where models of correct behavior are used. Definition 2.2 could be modified to include abductive approaches to diagnosis where fault models are used, but since this thesis makes no commitment to one approach or the other, those modifications are left out for brevity.

The Extended Diagnosis Problem as defined may appear to vary from the *The Diagnosis Problem* in the goal of identifying relations rather than identifying a subset of COMPS. However, in systems using the old definition, there is one relation for each constant c_i in COMPS that says $isa(c_i, type_j)$. (The relation may also be stated in the form $normal(c_i) \wedge type_j(c_i)$.) This relation then allows models describing the correct behavior for a $type_j$ object to be used to simulate c_i and derive conclusions. Since these relations are the only relations that may be suspended, and there is one relation for each element of COMPS, identifying a subset of COMPS is equivalent to identifying relations to suspend [73]. The relations have been explicitly included in *The Extended Diagnosis Problem* rather than identifying some set of individuals, because the models encountered required that a number of relations hold on individuals for models to apply. For example, the Dittus-Boelter model as implemented for this thesis (See Section C.2) requires that the individual within the pipe be liquid. Thus if the values generated by the model were discrepant with observed values, it could be because the relation requiring that the individual be a liquid was violated. The liquid individual also must have a particular shape, that is a cylinder that completely fills the interior of the pipe. This is a spatial relation that may be suspended if the model's values are discrepant with observations. The new definition presented here makes it possible to diagnose (suspend relations) situations other than a particular component being faulty, such as material contamination or spatial relations between components changing. A physical system may exhibit faulty behavior when components no longer have correct spatial relations to each other, even though all the pieces of the system are normal. These additional relations make it possible to diagnose additional faults, but they also make diagnosis more complex. A diagnosis system could ignore these relations, considering only relations of more likely faults, similar to the method developed by Davis [13]. Investigations of

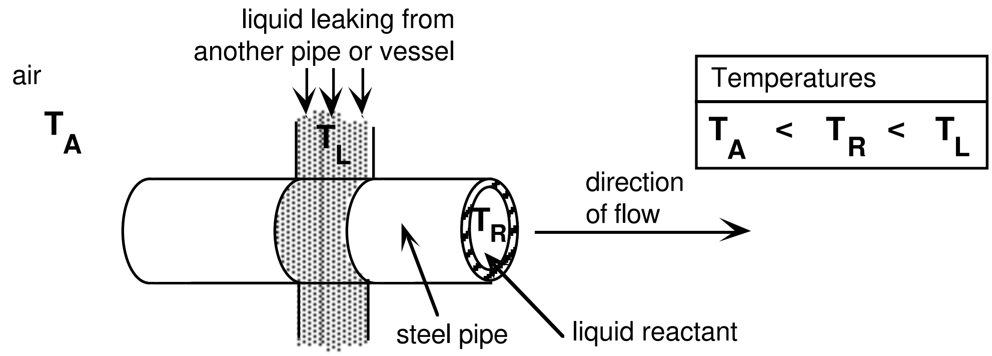


Figure 2.4: A hot liquid leaking from some other vessel flows over the outside of a pipe.

reactant liquid that is warmer than the outside air. Normally this liquid will be losing heat to the environment as it passes through the pipe. In the figure, a hotter liquid stream is shown flowing over the outside of the pipe, as might occur if some nearby pipe or vessel sprung a leak. In the portion of the pipe with the hotter fluid flowing across the outer surface, heat transfers through the pipe wall heating the reactant inside. To calculate what temperature may be achieved, one would apply some models of heat transfer including transfer at the surface of liquids to the adjacent pipe surface and conduction through the pipe wall. To calculate the liquid-solid heat transfer from the pipe to the reactant, one could use a heat transfer coefficient such as that calculated by the Dittus-Boelter equation. (The Dittus-Boelter equation is an empirical correlation which is described in many heat transfer textbooks, such as [33,56].) This engineering model applies to any length of the cylindrical space inside the pipe through which the fluid is flowing. (This model is used in some of the implemented examples in this thesis and is explained in detail in Section C.2.) For the situation shown in the figure, one would select the space inside the pipe that corresponds to the area over which the hot fluid is flowing on the outside. Note that the particular cylindrical space could not have been identified in advance of the hot liquid leaking. Nor could one identify all such possible cylindrical spaces, because there are too many such spaces.

An investigation of textbooks in the areas of heat transfer, reaction kinetics, fluid flow, thermodynamics, and mass transfer yielded a number of examples of non-component principles being used to solve problems. (These textbook examples are listed in Appendix A.) In these examples, as for the example of Figure 2.4, individuals were being identified by carving them out of the four-demensional space (3 spatial dimensions and 1 time dimension) occupied by some physical system. Clearly this space must be available at diagnosis time, if these principles are to be used. Definition 2.1 does not specify how the constants in COMPS relate to the space of physical equipment. They typically are chosen to represent pieces of the equipment that do not spatially overlap, but do not cover the four dimensional space of the equipment.

To extend the definition of *The Diagnosis Problem* to include diagnoses where non-component models are used, the four dimensional space must be made available, and the requirement of identifying all the individuals (*constants* in Definition 2.1) in advance must be removed.

structural components, they present an additional challenge for automated model-matching over component-oriented models.

2.4 Extending the Definition of "The Diagnosis Problem"

During the course of this research, we found that the definition of *The Diagnosis Problem* currently used in model-based diagnosis research excludes problems that are called *diagnosis problems* in the domain of fluid processing and chemical manufacture. Three recent articles [8,16,73] examine the model-based diagnosis literature and identify, among other characteristics, the variations in the definition of *The Diagnosis Problem* used by previous researchers. The relevant feature of the definition is what is assumed to be given. The paper by de Kleer, Mackworth, and Reiter [16] states that a *system* is given, and that a system has the following definition.

Definition 2.1 (System) *A system is a triple (SD, COMPS, OBS) where:*

1. *SD, the system description, is a set of first-order sentences.*
2. *COMPS, the system components, is a finite set of constants.*
3. *OBS, a set of observations, is a set of first-order sentences.*

The physical equipment is assumed to consist of components, COMPS, which form the set of constants described by the first-order sentences in SD and OBS. SD contains the models that predict the behavior of components, and OBS contains the values observed for physical parameters. To solve *The Diagnosis Problem*, one identifies the subset of COMPS which is behaving abnormally. The other two papers [8,73] use slightly different terminology, but indicate that some set of components is identified in advance and remains fixed through all diagnoses.

The problem with assuming that such a system is given is that it restricts the portions of the physical system to which models can be applied. The constants in the set COMPS are associated with fixed pieces of the physical system prior to any diagnosis problem arising. For some situations in the chemical processing domain where non-component models such as the Ideal Gas Law (Section 2.3.2) must be used, this restriction prohibits diagnosis. There are so many possible places that these models can apply, that it is prohibitive to identify all those places before the data arise to suggest the appropriate one. Furthermore, the appropriate place for model application in one diagnosis may overlap the place that is appropriate in another diagnosis, preventing identification of any single set of constants where each constant represents separate portions of the physical system, as the constants in COMPS are assumed to do. Throughout the rest of this thesis, the term *individual* is used to mean a portion of the physical equipment which is identified as an entity for the purpose of applying models. For example, for each application of the Ideal Gas Law, one must identify a particular volume of gas molecules. Components such as OR-gates, pumps, or valves also can be individuals.

As an example of a particular case where one must identify individuals at diagnosis time, consider a situation that can arise in fluid processing equipment as depected in Figure 2.4. A pipe carries a

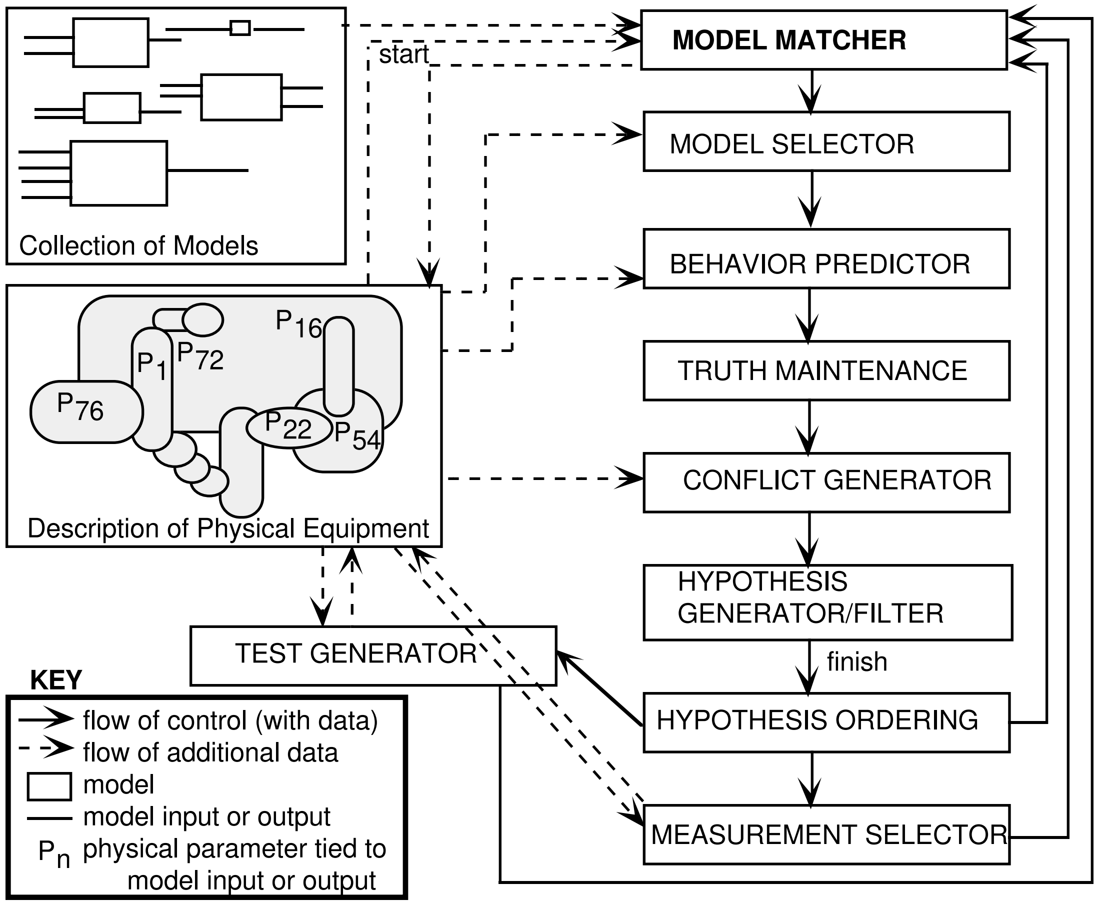


Figure 2.3: The architecture of a diagnosis system with a matcher.

chemical manufacturing equipment.

We implement cases involving fluid flow and heat transfer in pipes, pumps, and a stirred-tank reactor. The models included in the demonstrations could be used by engineers in chemical plants, semiconductor manufacturing facilities, electric power plants, pharmaceutical manufacturing, and refineries. This particular domain presents a challenge for automated model-matching because some of the models used do not correspond to structural components. For example, consider the Ideal Gas Law, $PV = nRT$. This model applies to any group of gas molecules that reside in a contiguous portion of space, assuming that the gas otherwise behaves nearly ideally. The portion of a physical system to which it is applied during diagnosis may be determined by what other parameters have been either calculated or measured. The region for model application may be determined by the values of parameters. The Ideal Gas Law requires that the temperature be uniform across the region of application. Other examples of such models include conservation of energy and mass balance. These kinds of models are called *non-component* models throughout this thesis. The examples of non-component models implemented in this thesis apply to portions of liquid flow through pipes. Because these models may be applied to portions of a physical system that do not correspond to

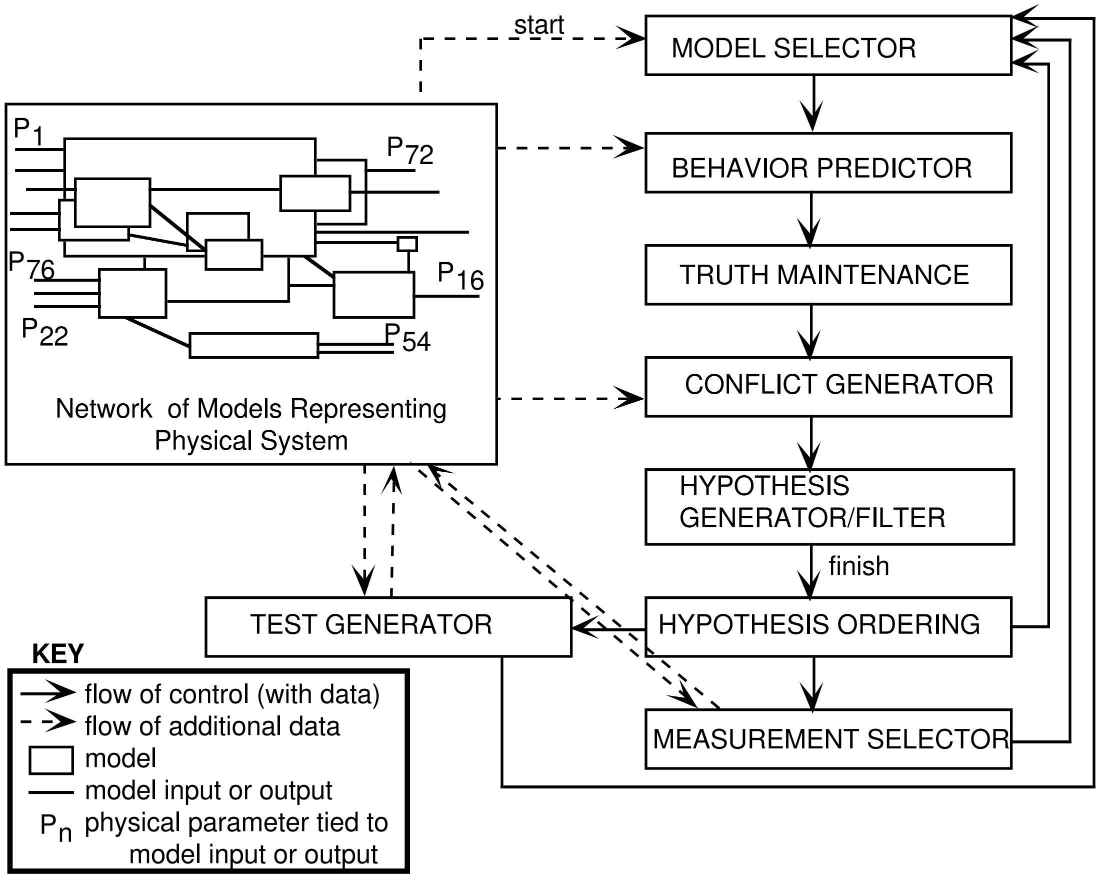


Figure 2.2: The architecture of a diagnosis system without a matcher.

systems for the purpose of attaching models, we assume that a description of a physical system is given. We also assume that some models are given with descriptions of the physical situations in which they apply. We investigate what characteristics must be included in those descriptions as well as how to match the models onto the physical system descriptions. In the example of Figure 2.1 where models describe components of digital circuits, matching models to an equipment description is relatively easy. However, in the domain of our investigation, fluid processing and chemical manufacturing equipment, matching is complicated by the types of models used.

2.3.2 The Domain

The models and equipment descriptions used in this work are taken from the chemical manufacturing domain. Our underlying assumptions restrict models to be computer programs that generate values for physical parameters. We also assume that the equipment occupies some continuous region in space and the models apply to pieces of continuous space. The space may be divided up into individuals, but the individuals must be contiguous. These assumptions appear to allow us to work in many domains where reasoning about some portion of the physical world is done. However, we test the methods only on some models and situations that are encountered in fluid flow and hypotheses when models of faulty behaviors are included with models of correct behaviors.

Other tasks arise when more than one hypothesis is generated. These include *ordering hypotheses*, *identifying observation points*, and *test generation*. Hypotheses may be ordered so that the most likely candidates can be generated first as in the work of Davis [13]. de Kleer [15] has shown that this is a useful strategy for avoiding the combinatorial explosion that results when all hypotheses are generated at once. Additional observations may discriminate between several competing hypotheses, perhaps eliminating some possibilities. Identifying the observations which will provide the most discriminating power has been investigated in [17]. In some physical systems, actions may be taken that change parameter values. The changed parameters may provide more discriminating power. Identifying which parameters to change and what values to give them is called *test generation.* Singh [69] and Shirley and Davis [67] have investigated this problem for digital circuits diagnosis.

Another task that arises when more than one model is available for any constituent of the physical system is *model selection.* A model may describe a constituent only over some portion of its operating range, and thus multiple models are required to cover the full operating range. A model may describe the behavior in a simplified form, so that the model itself is cheaper and faster to use, but it may not accurately describe the physical constituent under some circumstances. When multiple models are available, some criteria must be used to select the most appropriate model for the situation at hand. Some of the researchers addressing this task include Addanki *et al* [1], Struss [74], Dague *et al* [10], and Nayak *et al* [52].

Figure 2.2 summarizes the tasks within diagnosis. These systems contain a representation of the physical systems that is made up of a network of the models. These models' inputs and outputs are linked to form the network. The model inputs and outputs are also tied to physical parameters, which are represented as P_n in the figure. The diagnosis system has access to both the physical parameter values, P_n, as well as values calculated by the models. The diagnosis system also uses the network as a representation of structure in the physical equipment, indicating what components are connected. The figure shows flow of control among the various diagnosis tasks, indicated by the solid arrows. Information will flow along these paths and also along the paths represented by the dashed arrows. The process starts with some physical parameter appearing that is abnormal and ends when all but one of the diagnostic hypothesis have been ruled out.

A couple of the tasks appearing in the figure were not previously explained. After the selector chooses among the multiple models which may apply to the point of interest in the equipment, a *behavior predictor* runs the selected model to generate output values. A *conflict generator* compares calculated values to observed physical values or to other calculated values to detect conflicts. Our approach separates the models from the equipment as shown in Figure 2.3. We insert the model matcher which will map models onto the physical equipment as needed.

In each of the tasks described, instances of models were assumed to be attached to portions of physical systems with the models' variables correctly attached to physical parameters. In this thesis, we attempt to automate *model-matching,* the task of attaching models and model variables to the appropriate portions of physical systems. Since computer systems cannot readily observe physical

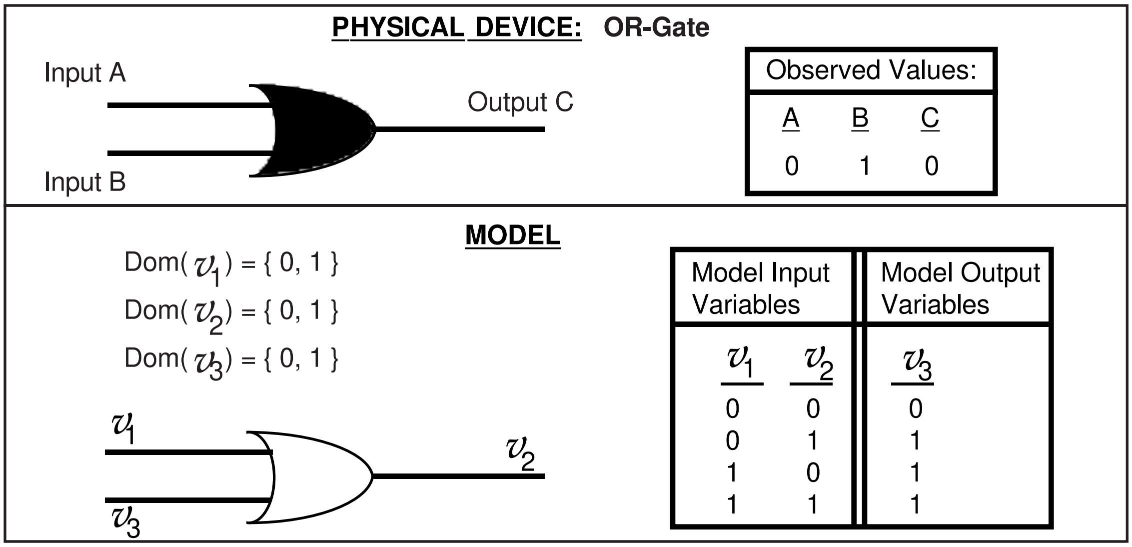


Figure 2.1: The model-based diagnosis paradigm compares parameter values predicted by models to observed values.

3. the model is not appropriate for the physical situation.

An example is shown in Figure 2.1. Here the physical system is an OR-gate and the model calculates the output of the OR-gate given both inputs. The variables are digital: they can only take values of zero or one. Using the values observed at Input A and Input B on the physical device as v_1 and v_2 for the model, the model predicts that the value observed at Output C should be 1. This is discrepant with the actual value of 0. Thus one can conclude that this particular OR-gate is faulty, that one or several of the observations of the physical parameters are incorrect, or that the model is not the correct model for the situation at hand.

When model-based diagnosis is applied to physical systems, there are typically many components such as the OR-gate above that are being simulated via models. The goal is to identify the location of the fault in these larger systems. In this context, other tasks must be carried out to successfully locate faults. One such task is *dependency tracking*. The calculated values which are used to derive discrepancies may depend on a number of components behaving correctly as well as the input values to the models being correct. In the example of Figure 2.1, only one component and one model is involved. In a larger system, the discrepancy may be derived from values generated by models of several components, each of which is a potential source of faults. Truth maintenance methods [14,21] accomplish this task of dependency tracking.

Once a discrepancy is found, *diagnostic hypothesis generation* must be done. The diagnostic hypothesis can be generated from the dependencies of the discrepant parameters, which are typically tracked by a truth maintenance system. The hypotheses generated may also depend on whether the discrepancy is assumed to be the result of a single fault. Some research in the area of hypothesis generation [18,16,58,61] has addressed the problem of identifying all possible hypotheses and representing that hypothesis space efficiently. Other research [19,75] has addressed how to generate systems, predicting behaviors in terms of a particular a set of variables $\mathbf{v_C} = (v_1, v_2, \ldots v_n)$ for that consituent, C. Each variable takes values from a domain $DOM(v_i)$, and then the cross product of these domains, $DOM(\mathbf{v_C}) = DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$, is the space of behaviors that can be represented. The model then specifies a relation R as a subset of $DOM(\mathbf{v_C})$ that gives the predicted behavior of the constituent.

However, our definition departs from Struss's definition [74] in that we assume models to be implemented computer programs. Inputs to the program form a subset of the variables $\mathbf{v_C}$, and program outputs form another subset, disjoint from the first. Other researchers such as Scarl *et al* [66] and Davis [13] have assumed that models are equations that can be manipulated symbolically (either by hand or by computer programs) to produce desired variables as outputs. This kind of manipulation is excluded by the definition used here. We also assume that models are run via function calls, for convenience of implemention. No assumption is made about the forms inside the models, though. The models may employ any data structures or algorithms that run on the computer platform being used. This aspect of the definition is less restrictive than that used by other researchers modeling physical systems such as Falkenhainer and Forbus [24] and Nayak [51]. They assume that models provide equations in a few specified forms that can be composed, simulated, or solved by programs in their systems. For this composition to be possible, the models must use internal representations that are compatible with the simulation program, thus restricting the class of models that can be included in such a system.

Another departure from Struss [74] is in the type of constituents to which models apply. Struss informally states that constituents may be, for example, components or processes. In this work, constituents will refer to any spatially contiguous portion of the physical system. Physical objects typically referred to as components, such as pumps, valves, or AND-gates, occupy spatially contiguous portions of a physical system and thus are constituents according to this definition. Processes, such as those used in Qualitative Process Theory [25], typically refer to spatially contiguous regions within a physical system. However, in this work, constituents may correspond to portions of processes or components. The models may apply to constituents of any size. No distinction is made between models describing portions of processes or components and models describing whole components or aggregates of components that together form a larger physical system.

2.3 What is the Model-Matching Problem?

2.3.1 Model-Matching: A Task in Model-Based Diagnosis

In model-based diagnosis, one uses models of physical systems to predict behavior. The values of parameters predicted by models are compared to measured values or other predictions to find discrepancies. If a model predicts values that are discrepant with what is occuring in the physical system, then either

1. the portion of the physical system being modeled is faulty,

2. the physical system's parameters were incorrectly observed, or

Chapter 2

The Model-Matching Problem

2.1 Summary

The model-matching problem must be solved for any system to proceed with model-based diagnosis or other reasoning about physical systems. It is a distinct task from other component model-based reasoning tasks, such as model selection, prediction, truth maintenance, conflict generation, hypothesis generation, hypothesis ordering, measurement selection, and test generation. All of these tasks are done (in some form) in an iteration to accomplish diagnosis. Model-matching typically has been done by hand. We automate model-matching and insert it as a task into a diagnosis iteration. Since our goal is to improve extensibility, our definition of *model* is different than previous definitions: we require that models be implemented computer programs and we allow them to use any internal representations and calculation methods, rather than a prescribed representation. Through our investigation of model-matching in the chemical processing domain, we found certain models, like the Ideal Gas Law, which apply to a very large number of locations within a physical system, and thus *cannot* be matched in advance of problem solving. The traditional definition of *The Model-Based Diagnosis Problem* assumes that individuals are fixed in advance and given as an input, and thus excludes some real world diagnosis problems where models such as the Ideal Gas Law are necessary. We suggest that the definition of *The Model-Based Diagnosis Problem* be revised. Our survey of the related work indicates that the model-matching problem has not been recognized as a separate problem. Some previous work on shifting ontologies, though, does carve the world up in different ways. This previous work generally maps back and forth between two different ways of carving up some kind of physical system using some partial representation of space. Our approach generalizes these methods by providing the whole space and letting the model and data suggest the appropriate individuals.

2.2 Models

We are concerned with models of engineered physical systems, specifically models that may be used in model-based diagnosis, where parameter values predicted by models are compared to measured values and to other predicted values. Thus the definition of *model* used in this research is similar to that used by other model-based diagnosis researchers. The terminology used here follows Struss [74]. Struss states that models describe constituents (components or processes) of physical

- It identifies several matching methods and demonstrates some of those methods in an implemented matching system. An important finding was that models sometimes specify entities and relations that cannot be present in addition to those that must be present. A matching method is developed to handle handle those specifications.

- It identifies several reconfiguration methods and demonstrates some of those methods in an implemented matching system. A particularly important finding with regard to reconfiguration is that only reconfigurations local to the point of failure in a match need be considered, if model entities are matched in a particular order.

Specific findings are listed in Section 8.2.

both. In these demonstrations, we use *non-component* models, models which apply to individuals that cannot be identified in advance, as well as *component* models. To demonstrate re-usability, we provided a series of equipment descriptions to the system, each one replacing the old one as if the system were being applied to some new piece of equipment. Each equipment description in a series contains the right things to match a target model, the same model for each equipment description in the series, but in each case the equipment description has a different set of individuals. We test two such series, one for a non-component model and 7 different equipment descriptions and one for a component model and 3 equipment descriptions. To show extensibility we make two series of model additions, one involving 3 non-component models and one involving 4 component models. A single equipment description is used for each of these two addition series. Because extensibility is affected by what kinds of models the system allows to be added, we incorporate several different arbitrarily implemented models into the system. However, to more strongly demonstrate that the system admits models which were not designed by us to work in this system, we include a model built by other researchers that uses the QSIM qualitative simulation program [42] to do its calculations. This is a large program, 1.2 M of Common Lisp code, which was correctly matched and executed by the model matcher. Since we were able to demonstrate this enhanced re-usability and extensibility, we conclude that we have identified some of the characteristics and developed some of the algorithms necessary to automate model-matching.

1.6 Overview of Contributions

This thesis makes several contributions to the understanding and solution the model-matching problem:

- It indentifies a class of models, non-component models, which *require* run-time individuation. This finding has implications for model based diagnosis research. The traditional definition of model-based reasoning, including the most studied instance, *the Model-Based Diagnosis Problem,* assumes that individuals are given and fixed, and thus excludes problems where non-component models are required.

- It identifies characteristics of equipment and models that must be represented explicitly for run-time model-matching. The single most important characteristic that has not been included in previous model-based reasoning systems is the 3-dimensional space occupied by equipment. This thesis verifies the identified characteristics in an implemented system that correctly matches and reconfigures in two series of extensibility tests and two series of re-usability tests.

- It has identified property transformation methods, ways of calculating parameters for merged or split regions from initial regions' parameters, as necessary methods for an automated model matcher. These transforms represent background engineering knowledge. Some of this knowledge is implemented and demonstrated.

1.5 Solution Overview

Our solution to the automated model-matching problem revolves around re-defining the portions of the equipment that are individuals at matching time, if necessary. The equipment description provided to a system must contain enough information to make that possible, and it must retain the 3-dimensional space in its representation. Our approach allows users to divide the equipment into individuals in whatever way they find convenient, as long as the individuals represent the whole body of space occupied by their equipment. The space may be carved up in any manner, but no pieces may be left out of the representation. With these representational constraints, making a new individual from such a representation is matter of *merging* and *splitting* adjacent individuals.

The physical individuals required by a model are described using the same underlying principles as for the equipment. These individuals also partition a 3-dimensional space. Since the engineering models that we observed require entities that exemplify not only particular functional types (such as `pump` or `reactor`), but also particular materials and phases (such as gas or liquid), we provide means whereby these properties can be described for individuals both in equipment and models. All the models require parameters, which have particular locations and spatial extents within the individuals. The spatial representation method allows the spatial extents of parameters to be made explicit. We also find that certain other characteristics of models, such as restrictions on their parameter values and classification of parameters, are necessary to correctly match and use them.

We develop a matching method which, given a particular equipment parameter (having a particular location and spatial extent in the equipment), proceeds through a model description, attempting to match all the individuals and characteristics to those in the equipment. If a match cannot be made, a reconfiguration may be attempted. We develop several reconfiguration methods which can change the form of the individuals by merging and splitting. One algorithm is called when a model requires a region that has only material and/or phase specified, but no functional type. We can generate different equipment individuals by merging adjacent regions of the appropriate phase and material or by splitting a region of that phase and material. Another algorithm is triggered when the model requires an individual of a particular functional type (such as `pump`) and the equipment has individuals that are subparts of a pump.

For such a reconfiguring system to produce new individuals that are useful reasoning tasks, it must be able to generate the properties of the individuals. For example, if we merge two adjacent regions of liquid water, the resulting region also will be liquid water. The resulting region has a mass that is the sum of the masses of the two initial regions. This is background knowledge for engineers. How a parameter is transformed through a merge or split depends on the type of the parameter, its spatial extent, the type of reconfiguration (merge or split), and the type of region involved (phase, material, functional type). We identify several classifications for parameter types based on how they behave through merges and splits and implement transformation algorithms to calculate the new parameters for a merged or split spatial entitiy from the initial entities and their parameters.

Since our goal is to make systems more re-usable and extensible, we provide demonstrations of apply and about physical equipment must be made explicit than is used in systems where model-matching is done by humans. Some method for matching the descriptions of models to physical equipment also had to be developed. Thus the goals of this research were to:

1. identify characteristics required to describe models,

2. identify characteristics required to describe equipment,

3. and develop methods to match models to equipment

for selected representative engineering models in representative physical situations. By examining some models and some situations, we identify a subset of the additional characteristics required to solve the general model-matching problem and develop a subset of the matching methods required, given those additional characteristics.

Since the motivation for automating model-matching is to gain re-usability and extensibility, the test of the methods developed is to show that they do have the independence of model descriptions from equipment descriptions described in Section 1.2. If a model can be matched to a number of equipment descriptions where the level of detail varied from that used in the model description, then the matching methods provide the desired independence for re-usability, at least for the model tested. Likewise, if a number of different models can be added to the model matcher's model collection where the level of detail differed from that in the system's current equipment description, and if those models can be correctly matched and used to calculate new information, then the methods provide the desired independence for extensibility.

This investigation of model-matching is a first step. Clearly efficiency of model-matching is important for practical use as for all computer programs. However, given that this work is the first attempt at model-matching, we focused on functionality more than performance. Although optimization of implementations for efficiency purposes is left for future work, some analysis of the algorithms as well as monitoring of implemented examples are done to assess feasibility.

1.4 Focus on Model-Based Diagnosis

Even though we believe that the problems and solutions studied are relevant to all model-based reasoning tasks, we have focused on the model-based *diagnosis* task. For other tasks, such as analysis or explanation, perhaps only correct behaviors are important. This class of behaviors may be enumerable, and if it is, individuals can also be identified in advance. To diagnose a physical system, one must be able to reason about correct behaviors as well as incorrect behaviors, even when those incorrect behaviors cannot be enumerated in advance. The incentive for automating model-matching is greater in diagnosis than in other model-based reasoning tasks, and we place our work in the context of model-based diagnosis.

different models applicable to a centrifugal pump are present, and each model requires a different level of detail, the equipment description should not have to be built at those three specific levels of detail in order for models to be matched. Of course, the equipment description must contain as much or more detail than a model requires for that model to successfully match.

Extensibility is measured by the ease of adding a new model to a reasoning system and using it to calculate something new. If model-matching is not automated, one must identify every possible place in a physical system where the new model applies and identify the correct matching of model variables to physical paramters. If model-matching is automated, adding a model requires only describing the physical situation in which the model applies. The level of detail provided in the description of the model's physical situation should be independent of the current equipment description. One should not have to find all possible places in the equipment where the model may apply, and provide descriptions with the model at all the levels of detail used in those possible places. Furthermore, the model itself should not have to be re-implemented to use a representation scheme prescribed by the model matcher or diagnosis system. Many models of engineering phenomena already exist; having to re-write them makes model addition more difficult and expensive and thus reduces extensibility. In fact, Given the variety of mathematical forms and calculation methods used in engineering models, it is likely that requiring a predetermined internal representation scheme for models would prevent many models from being included because they could not be written in the prescribed form.

During the course of this research, we discovered a third, very important reason for automating model-matching. Run-time model-matching is *required* for certain kinds of models, like the Ideal Gas Law, because of the very large (or infinite) number of portions of a physical system to which these models may apply. We refer to models with this characteristic as *non-component* models. The particular location in which these models apply is ultimately determined by parameter values. Values will typically not be available until reasoning time, since many values are calculated allong the way. Non-component models are essential to engineering reasoning, but assigning them to every possible location in advance of any task would not be feasible. If models are assigned at run-time, then the values available allow a system to match only relevant models at relevant locations. This finding is elaborated in Section 2.4.

1.3 Goals for this Research

This research is the first attempt at solving the general model-matching problem. Defined as a problem of matching engineering models implemented as computer programs to descriptions of real physical systems, the general problem is very large. The number of situations that real physical systems can present is very large. The number of enginnering models, either implemented as programs or not, is also very large. It is our opinion that attempting to match any one of these models to any of the physical situations could point out deficiencies in an automated model-matching system. (See Section 2.5 for a discussion of related work.)

To enable model-matching, more information about the physical situations in which models

Chapter 1

Introduction

1.1 Problem Overview

In all model-based reasoning systems, models must be assigned to portions of the physical world. Typically this is done by humans at the time the reasoning system is built. Instances of appropriate models are linked together to form a representation of the physical world. A few systems perform model assignment automatically. They allow the equipment to be described in terms used in the models and then do the assignment using a straightforward matching of model terms to equipment terms [23,24]. Before these systems start their tasks, though, all models must be assigned to all possible places in the equipment to which they could later be applied. This thesis proposes a more general solution to the problem of automating model-matching.

The model-matching problem is somewhat more difficult than the term *matching* or *assigning* might suggest. Models generally apply to particular *individuals*, portions of the physical world that have been identified as separate entities. To describe equipment so that a model can apply, one must choose the right portions of the physical world to be individuals. However, if the set of models is not fixed at the time the equipment description is built, one cannot know which individuals to identify. In this work, we do *not* assume that the set of models is fixed, but that the set may change and grow over the lifetime of the system. Thus the matching problem includes the problem of *individuating*, defining portions of the physical equipment that should be treated as separate entities.

1.2 Motivation for Automated Model-Matching

Our original motivations for automating model-matching were to improve *re-usability* and *extensibility* of model-based reasoning systems. In the course of conducting the research, we discovered a third motivation: Automated model-matching is *required* for certain kinds of models. *Re-usability* is measured by the ease of applying a reasoning system and its models to different physical equipment. Without automation, one must identify every possible place in the new equipment where each model applies. If model-matching is automated, one should only have to describe the new physical system and install that description into a reasoning system that contains the model matcher. Furthermore, describing new equipment should be accomplished independently of the models in the system. Otherwise, the description task grows with the size and number of models. So, for example, if three

List of Figures

List of Tables

Contents

Acknowledgements

I would like to thank my adviser, Barbara Hayes-Roth, for direction, advice, and support throughout the research and writing of this dissertation. I would also like to thank the other members of my committee, Jean-Claude Latombe and Krishna Saraswat, for their critical analysis and questions that helped me clarify the research issues and shape the thesis. Thanks also are due to the people at the Knowledge Systems Laboratory for providing a stimulating research environment. I would like to thank Edward Feigenbaum, Director of the Knowledge Systems Lab, for sponsoring this work. I am indebted to Rich Acuff, Lee Brownston, Torsten Heycke, and Jim Rice for various forms of system support, especially firefighting system problems that occur immediately prior to deadlines. I would also like to thank the people who keep the Knowledge Systems Lab running, Grace Smith, Michelle Perrie, and Peche Turner, who have helped me with numerous administrative problems during my stay here. At the Center for Integrated Systems, I want to acknowledge Len Booth and Don Gardner for helping me understand processes and problems in integrated circuit manufacturing.

I am grateful to Xerox Corporation for providing support for this work in the form of a Xerox Fellowship. I especially want to thank Carolyn Tajnai, Director of the Computer Forum, for making the fellowship program a reality here at Stanford and for providing advice and encouragement through my stay. I want to acknowledge Texas Instruments, which provided some of the support for this research through the Microelectronics Manufacturing Science and Technology (MMST) Program (P.O. No. 7554900).

I would like to thank friends and colleagues who have helped me at various stages in the Ph.D. program. I am very grateful to Wayne Montoya and Dave McBride for assistance in formatting and figure drawing for the dissertation. Henny Sipma and John Mohammed provided many helpful discussions of research issues. Karen Pieper, Geoff Phipps, and Steve Nowick provided advice through the early stages of the Ph.D. program. I especially want to thank Liz Wolf and Marcia Derr for advice and moral support through the entire Ph.D. experience.

Abstract

In model-based systems that reason about the physical world, models must be matched to portions of the physical system. To make model-based reasoning and diagnosis systems more readily extensible and re-usable, this thesis explores automating model-matching. If matching is automated, one can add a model without specifying every place in the physical equipment where it can be used. One can apply the system to new equipment without identifying every place that every model may be used. However, models address particular *individuals*, portions of the physical world identified as separate entities. If the set of models is not fixed, one cannot carve the physical system into a fixed set of individuals. Our goals are to develop methods for individuating and matching models and to identify characteristics of physical equipment that must made explicit for those methods.

Our investigation involves three steps. First we explore examples of engineering models applied to physical systems found in textbooks or in manufacturing equipment to identify relevant characteristics. Second, we implement matching methods using the characteristics. Third, we test re-usability and extensibility. If the system can correctly define individuals and match some models, even when models call for individuals not previously defined, then we can conclude that we have identified some subset of the characteristics required to automate model-matching.

The first step of the investigation revealed that a number of models used in the domain of fluid processing and chemical manufacturing do not correspond to *components* such as valves, tanks, or pumps. Many principles apply to regions containing particular materials or phases, or having particular parameter values. An example is the Ideal Gas Law, which applies to any volume of space occupied by molecules in the gas phase. Individuals for these kinds of models cannot be identified in advance because there are too many possible individuals. Previous model-based diagnosis work assumes the set of individuals is *given* and *fixed.* This assumption excludes real world diagnosis problems where models like the Ideal Gas Law are required. Identification of this class of models also shows that run-time individuation is required to solve certain kinds of problems.

We develop two matching and two reconfiguration algorithms which use descriptions of the space occupied by the equipment and the space required by models to reconfigure individuals at run-time. Two series of equipment description replacements demonstrate re-usability. Each equipment description in a series has content to match the same model, but had represented as different individuals. Two series of model additions demonstrate extensibility. In each series, the equipment description remains constant, and the added models' individuals vary. The system correctly reconfigures and matches in all cases. We conclude that the 3-dimensional space occupied by the equipment and required by the models along with the distribution of phases, materials, and functional components within that space are required for model-matching. The locations and spatial extents of parameters are also required.

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Barbara Hayes-Roth
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe
(Co-Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Krishna Saraswat

Approved for the University Committee on Graduate Studies:

MODEL-MATCHING AND INDIVIDUATION FOR MODEL-BASED DIAGNOSIS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Janet L. Murdock
January 1995