Model Based Knowledge Organization

Wylie, R.; Kamel, M.

3. DeCoste, D. 'Dynamic across–time measurement interpretation'. In *Proceedings AAAI–90*, pp. 373–379. MIT Press, 1990.

4. Gallanti, M., Gilardoni, L., Guida, G., and Stefanini, A. 'Exploiting physical and design knowledge in the diagnosis of complex industrial systems'. In eds. Du Boulay, B., Hogg, D., and Steels, L., *Advances in Artificial Intelligence – II*, pp. 481–495. Elsevier Science Publishers B. V., 1987.

5. Knoblock, C. A. 'A theory of abstraction for hierarchical planning'. In ed. Benjamin, P. D., *Change of representation and inductive bias*, pp. 81–104. Kluwer, 1990.

6. Kuipers, B. 'Qualitative simulation'. *Artificial Intelligence*, vol. 29, pp. 289–388, 1986.

7. Lakshmanan, R. and Stephanopoulos, G. 'Synthesis of operating procedures for complete chemical plants – 1. hierarchical, structured modelling for nonlinear planning'. *Computing in Chemical Engineering*, vol. 12, #9/10, pp. 985–1002, 1988.

8. Lakshmanan, R. and Stephanopoulos, G. 'Synthesis of operating procedures for complete chemical plants – 2. a nonlinear planning methodology'. *Computing in Chemical Engineering*, vol. 12, #9/10, pp. 1003–1021, 1988.

9. Meystel, A. 'Intelligent control: A sketch of the theory'. *Journal of Intellingent and Robotic Systems*, vol. 2, pp. 97–107, 1989.

10. Sykes, D. J. and Cochran, J. K. 'Development of diagnostic expert systems using qualitative simulation'. In *AI and Simulation 1988*, pp. 32–38. The Society for Computer Simulation, 1988.

11. Venkatasubramanian, V. and Rich, S. H. 'An object–oriented two–tier architecture for integrating compiled and deep–level knowledge for process diagnosis'. *Computing in Chemical Engineering*, vol. 12, #9/10, pp. 903–921, 1988.

12. Wylie, R. and Kamel, M. 'Model based knowledge organization: A framework for constructing high level control systems'. *Expert Systems With Applications*, vol. 4, #3, 1992. Accepted for publication.

13. Yang, Q. *Abtweak user's manual.* University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1990.

14. Zeigler, B. P. *Multifacetted modelling and discrete event simulation.* Academic Press, 1984.

DISCUSSION

The paper presents our efforts to date on defining a domain independent language and set of operators for representing and manipulating models and behaviour. We have shown that a certain domain independent operator (BPP) can be used to automatically identify appropriately sized models for analysis of sensor data, for diagnosis, and for planning operating mode changes. We have also shown how the domain independent representation supports reasoning from multiple models. Two types of interaction are shown: sharing of knowledge about the system (planning and QP model interacting to solve operating mode change problem); and passing of control (diagnosis task spawned by sensor data analysis task).

A critical issue in this research is the derivation and interpretation of dependency information from domain specific models. This issue is reasonably well understood in hierarchical and non–linear planning where the strips assumption severely restricts the manner in which variables can interact.

The same cannot be said for QP and discrete event simulation. In these domains there seems to be a complex interaction between causal relationships in the system being modelled, knowledge about system state provided in the question, the specific reasoning strategy used, and the overall goals of the reasoning tasks. In the work presented here, dependency information was restricted to knowledge about system state derived from the question and causal arguments derived from the system.

One important focus of this research in the immediate future will be to extend and formalize the derivation of dependency knowledge.

A second immediate task to be undertaken is to extend the example to include monitoring of plan execution, and to demonstrate the method on more complex interpretation, diagnosis and planning problems.

A third task is to reimplement the MBP representation facility and to extend its functionality by adding an interface to a discrete event simulation tool.

Finally, the longer term research plan is to define and test other behaviour preserving transformation operators which simplify models by aggregation rather than pruning, and to consider operators of both types (pruning and aggregation) which aren't strictly behaviour preserving but for which the divergence in behaviour must be estimated empirically.

REFERENCES

1. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The design and analysis of algorithms.* Addison–Wesley, 1976.

2. Chapman, D. 'Planning for conjunctive goals'. *Artificial Intelligence*, vol. 32, pp. 333–377, 1987.

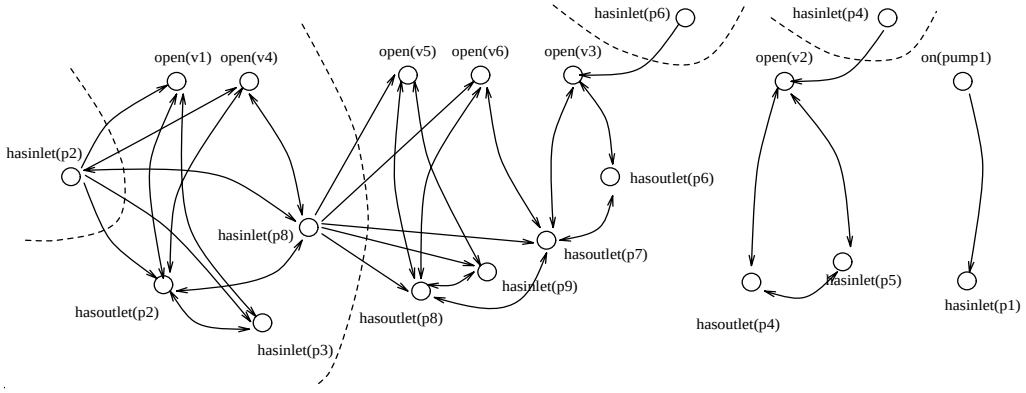

Figure 5: Dependency Graph and Decomposition for Planning Model

such a graph would be unmanageably large. For this reason in the planning literature, only that portion of the graph required for a given question is constructed (see Knoblock [5]).

In the decomposed planning model, as in the decomposed QP model, only those strongly connected components which contain literals mentioned in the problem or which are "upstream" of such a component need be considered. As well, the reduced planning task can be profitably solved hierarchically, starting with the most dependent submodel and working upstream (for a detailed discussion of this please see Knoblock [5]).

In the current problem, pruning allows us to eliminate all references to valve–2, pipe–1, pipe–4, pipe–5, and pump–1. Solving the reduced planning problem results in a set of four ordinal constraints between three operators and the initial and goal states:

$$\begin{aligned} \text{Initial–state before close–valve(valve1)} \\ \text{close-valve(valve1) before open–valve(valve4)} \\ \text{open-valve(valve4) before open-valve(valve5)} \\ \text{open-valve(valve5) before goal–state} \end{aligned}$$

Both this reduced problem and the full problem were solved using a non–linear planner called Abtweak (Yang [13]). The same solution was produced in both cases while performance improved by 31% to 40% (in both number of nodes expanded and in processing time). While these results must be taken as preliminary, they do support the argument that Behaviour Preserving Pruning of the problem prior to solution is worthwhile.

In this simple problem, the resulting non–linear plan admits only one ordering of the operators. This will not generally be the case, and we may exploit other system models to provide further ordering constraints and to check the plan for feasibility.

operator name	preconditions	postconditions
open–valve(v, src, snk)	valve(v), pipe(src), pipe(snk), between(v,src,snk), ¬has-outlet(src), ¬has-inlet(snk), has-inlet(src), ¬open(v)	open(v), has-outlet(src), has-inlet(snk)
close–valve(v, src, snk)	valve(v), pipe(src), pipe(snk), between(v,src,snk), open(v)	¬open(v), ¬has-outlet(src), ¬has-inlet(snk)
start–pump(p, src, snk)	pump(p), pipe(src), pipe(snk), between(v,src,snk), ¬on(p), has–inlet(src)	on(p)
stop–pump(p)	pump(p), on(p)	¬on(p)

Table 5: Planning Operator Definitions

The drain–and–isolate problem's goal state may be expressed as an incomplete behaviour in the QP domain

$$\text{amount–A} = ((0,0), std)$$
$$\text{outlow–B} = ((0, fmax), std)$$

Completion of the resulting MBP yields flow rates through the various pipes in the extended model. These flow rates in turn determine valve settings consistent with the goal:

$$\neg\text{open(valve–1)}$$
$$\text{open(valve–3, valve–4, valve–5)}$$
$$\text{on(pump–1)}$$

Note that the goal state does not stipulate the state of valves 2 and 6 since their status is irrelevant to satisfaction of the goals.

Using BPP in Planning To solve the planning problem, the Behaviour Preserving Pruning operator may once again be applied. As with QP models, the domain independent dependency graph structure is determined by component interconnections. In Figure 5 the full graph is shown. For space reasons, names have been reduced so that *pi* refers to pipe–i, *vj* refers to valve-j, and type relations (ie. valve(?x), pipe(?x), pump(?x)) are not shown. The edges in this graph are oriented from preconditions to effects, and all the effects of a single operator are forced into a single strongly connected component by linking them with bi–directional edges.

Note that it is not normally possible to exhaustively construct this graph. In domains without a sparse topology such as our pipe network

synthesizing operating mode changes, we approach the problem in three phases: 1) define initial, goal and possibly intermediate (planning island) states using a detailed model of the process; 2) plan a (non–linear) sequence of valve and pump operations to solve each subproblem identified above and, 3) derive additional temporal constraints for these subproblems from detailed models of the process (and possibly from heuristics, predefined operating procedures etc.) Such hybrid approaches to planning are often required because of the representation language used in classical planning often too restrictive (Chapman [2]).

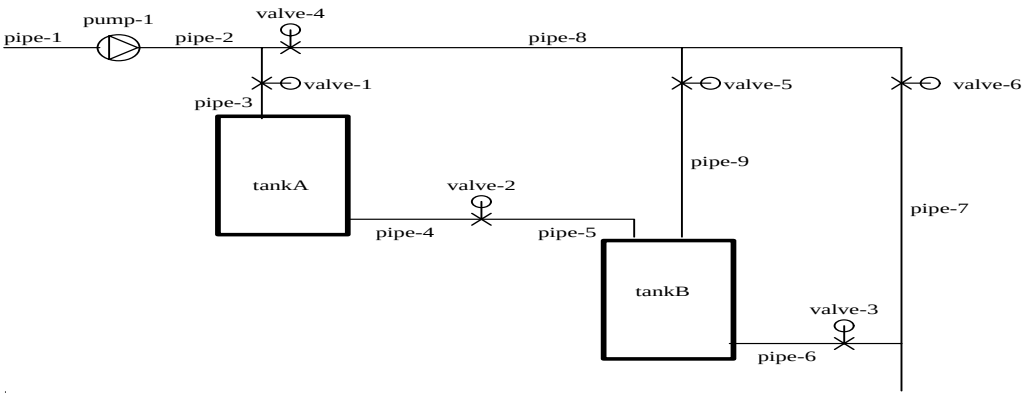

Figure 4: Extended Two Tank System

In this presentation, the two tank system discussed above will be extended by addition of pumps, valves and by–pass piping. The resulting augmented system is shown in Figure 4.

A simple axiomatization of the pump–and–valve domain will be used which consists of four operators: open–valve(), close–valve(), start–pump(), and stop–pump(). System state is represented by eight relations: open(), on(), has-outlet(), has-inlet(), between(), valve(), pipe(), and pump()). Table 5 shows the operators.

To continue with the sensor failure problem. We consider here the problem of planning a sequence of valve and pump operations which drains tank A while maintaining flow to tank B. This is the first stage of a repair procedure which involves isolating and draining the tank with the failed sensor, replacing the sensor and then bringing the tank back on–line.

Defining initial and goal states The initial state for the planning problem is:

$$\neg\text{open}(\text{valve–4, valve–5, valve–6})$$
$$\text{open}(\text{valve–1, valve–2, valve–3})$$
$$\text{on}(\text{pump–1})$$

and is assumed to be available from knowledge of previous control actions.

Diagnosis

Diagnosing sensor failures, equipment failures and process upsets are closely related problems. In all cases, the sensor data interpretation task will have encountered an inconsistency between the observed behaviour and the behaviour predicted by the no–fault system models. When the diagnosis task starts, the fault has already been isolated to the submodel exhibiting the inconsistency and those submodels upon which it depends. In the case of our level sensor failure, submodel–1 from Table 3 (corresponding to Tank–A) contains the fault.

The most straight forward approach to model based diagnosis involves search through the space of possible fault hypotheses for the one which most plausibly explains the observed behaviour. Sensor fault hypotheses may be tested by disregarding individual sensors to determine whether the remaining sensors are consistent. In the case of equipment diagnosis, the fault hypotheses require libraries of component models corresponding to the known failure modes for the components. Heuristics based on fault frequency may be used to guide the search.

In this problem, assume that sensor failures are equally likely and that they are more common than equipment faults. Thus the search of the fault space begins with a set of questions intended to test sensor failure hypotheses. The partial behaviour in each of these questions disregards data from one sensor but contains verified behavioural data from adjacent (consistent) submodels. In Table 4 are the results of completing these behaviours. Test–1 disregards tank level sensor data while test–2 disregards inlet flow sensor data. Because the model is consistent with the inlet flow data but not with the tank level data, the level sensor is identified as being at fault.

	Time	interval 1		interval 2		interval 3	
Test	Variable	amt	dir	amt	dir	amt	dir
test-	Flow-in-A	fnorm	std	fnorm	std	fnorm	std
1	Flow-A-B	(0,fmax)	+	(0,fmax)	+	(0,fmax)	+
	Netflow-A	(0,nmax)	+	(0,nmax)	+	(0,nmax)	+
	Pressure-A	(0,pmax)	+	(0,pmax)	+	(0,pmax)	+
test-	Level-A	(0,lmax)	+	(0,lmax)	std	0	std
2	Flow-A-B	(0,fmax)	+	(0,fmax)	std	(0,fmax)	?
	Netflow-A	(0,nmax)	?	0	?	inconsistent	
	Pressure-A	(0,pmax)	+	(0,pmax)	std	inconsistent	

Table 4: Results of Diagnostic Tests

Were the sensors to have passed these tests, search of equipment failure modes would follow.

Planning

The final reasoning task to be discussed in this paper is planning operating mode changes. Following Lakshmanan and Stephanopolous [7] work on

An added benefit is that this approach isolates inconsistencies to submodels. When inconsistencies are encountered in the completion of one of the MBPs the problem is known to be either in that MBP or in one of the MBP's previously completed. Thus the resulting diagnosis task may be directly stated as a question composed of only these MBPs.

Time	interval 1		interval 2		interval 3	
Variable	amt	dir	amt	dir	amt	dir
Flow-in-A	fnorm	std	fnorm	std	fnorm	std
Level-A	(0,lmax)	+	(0,lmax)	std	0	std
Level-B	(0,lmax)	+	(0,lmax)	+	(0,lmax)	+
Flow-out-B	(o,fmax)	+	(0,fmax)	+	(0,fmax)	+

Table 2: Sensor Data Set

Time Interval		interval 1		interval 2		interval 3	
model	Variable	amt	dir	amt	dir	amt	dir
sub-	Netflow-A	(0,nmax)	+	(0,nmax)	std	inconsistent	
model	Pressure-A	(0,pmax)	+	(0,pmax)	std	inconsistent	
1	Flow-A-B	(0,fmax)	+	(0,fmax)	std	inconsistent	
sub-	Netflow-B	(0,nmax)	?	(0,nmax)	?	(0,nmax)	?
Model	Pressure-B	(0,pmax)	+	(0,pmax)	+	(0,pmax)	+
2	Flow-A-B	(0,fmax)	+	(0,fmax)	?	(0,fmax)	?
full	Flow-A-B	(0,fmax)	+	(0,fmax)	std	inconsistent	

Table 3: Results of MBP completions

To make this more clear, consider once again the two tank system of Figure 3. In Table 2 is preprocessed sensor data in QP form (amt,dir). The dependency graph and resulting decomposition of the QP model based on this sensor set is just that shown in Figure 3.

Table 3 shows the results of completing the MBPs in order of their dependency (so that Submodel–1 refers to the SCCs associated with Tank–A and Submodel–2 refers to the SCCs associated with Tank–B). In the first two intervals, no inconsistencies are encountered and the sensor data interpretation task succeeds. In the third interval, an inconsistency is encountered in the first submodel.

Anticipating the next section, assume that a diagnostic task has determined that it is the level sensor in tank A has failed. This fact forces a redefinition of the sensor diagnosis task. The question defining this new task uses the same model but the incomplete behaviour will no longer contain information about the level of tank A. The new MBP is decomposed to yield a new hierarchy of models and sensor data analysis continues.

Many issues of sensor data analysis have been disregarded in this presentation. For a discussion of these issues in the context of QP see De Coste [3].

Model bases

The final concept to formalize is that of a collection of models. We define a model base to be a triple:

$$MB =< \{M_i\}, R_v, R_c >$$

where:
- M_i are models,
- R_v is a binary relation $\{< v_{i,k}, v_{j,l} >\}$ each tuple of which relates a pair of variables from different models,
- R_c is a binary relation $\{< c_{i,k}, c_{j,l} >\}$ each tuple of which relates a pair of components from different models

Where models from different domains overlap with respect to the physical system being represented they will contain common components. Where models from the same domain overlap, they must correspond both with respect to components and with respect to variables.

EXAMPLES OF REASONING

This section continues from the tank example used above to describe the BPP operator. While simple, this example is adequate for showing the use of the language and operators in a range of reasoning tasks and for showing how these reasoning tasks can share knowledge and interact. The specific reasoning tasks which will be considered are: interpreting sensor data; diagnosing equipment and sensor faults; and planning operating mode changes.

Sensor data interpretation

Sensor data interpretation can be defined as completion of MBPs in which partial behaviours contain only sensor data. Sensor data interpretation stops short of diagnosing equipment and sensor failures. When it cannot reconcile the sensor data with its model of the process, it passes the problem on to another task dedicated to diagnosis.

The behaviour preserving pruning operation may be used to automatically decompose a large model into appropriate submodels for interpretation of sensor data. When changes occur to either the process model or the available sensor data (eg. due to sensor or equipment failures), these submodels may be automatically redecomposed.

Given a sensor suite, the BPP operator identifies a dependency hierarchy of submodels. Sensor data interpretation involves completion of an MBP for each of these sub–models in the order of their dependency. The completed behaviours from the submodels contain the desired system state information. This approach reduces computational costs by reducing the average model size.

Notice that many of the edges between variables are bidirectional. This is a characteristic of the reasoning methods in a particular modelling domain (Kuipers QSIM [6]). However, even in such domains, certain edges, such as those associated with irreversible processes can be given a unique direction. The meaning of these directed edges is that knowing the value of the variable at the origin of the edge can *change* the value of the variable at the edge's destination. Knowledge of the value of the destination variable can only *restrict* the value of the origin variable.

Now consider a question $Q = \{M, B_Q, \epsilon\}$. Say $\epsilon_i = \pm\infty$ for every variable except pressure-A. Following the definition of questions, this indicates that for this reasoning task only the pressure in tank A is of interest. Say also that the partial behaviour B_Q provides values for four variables: flows into tank–A and out of tank–B, and fluid levels in both tanks. Hence all edges incident on the node representing tank level will point away. The resulting problem specific dependency diagram is shown in Figure 3.

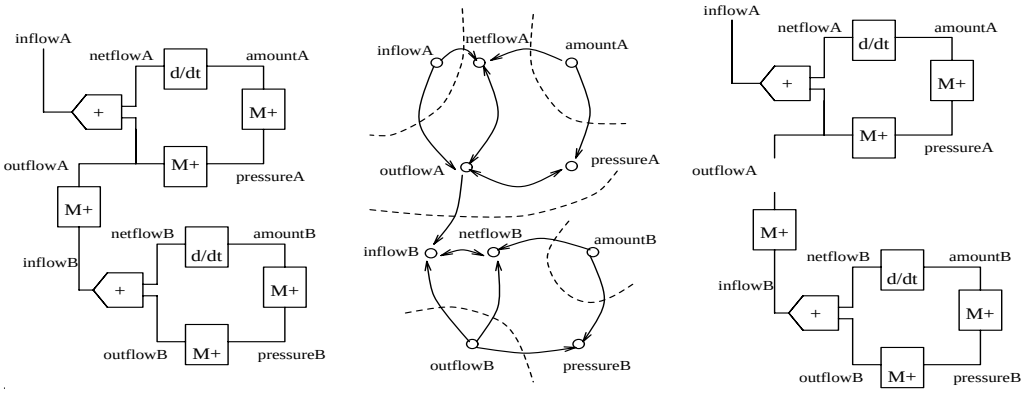

Figure 3: Dependency graph and decomposition of QP model

From this dependency graph it should be clear that the behaviour of the down–stream tank is irrelevant to the question at hand. Our Behaviour Preserving Pruning (BPP) operator identifies this fact by partitioning the variables into sets (called strongly connected components or SCCs) where any pair of variables in the same SCC are mutually dependent and any pair of variables selected from different SCCs are either unidirectionally dependent or are mutually independent. This partitioning of the dependency graph is shown by the dotted lines in Figure 3. The submodels resulting from this decomposition are also shown in Figure 3 (note that the uninteresting SCCs associated with single variables have been collapsed).

Algorithms for decomposing directed graphs into strongly connected components can be found in any good algorithms text (eg. Aho Hopcroft and Ullman [1]).

While the example given involves decomposing a QP model, BPP is equally applicable to any other domain where dependency relations can be identified between variables. Examples include planning (Knoblock [5]) and discrete event simulation (Zeigler [14]).

interesting variables are given the first tolerance and uninteresting variables are given the second. A question, Q, is defined as the 3-tuple:

$$Q =< M, B, \epsilon >$$

where: M is the (possibly incomplete) model
B is the (possibly incomplete) behaviour
ϵ is a vector of error terms, one per output variable

Behaviour preserving operations on MBPs

The most obvious behaviour preserving model transformations (BPT) are the domain dependent operators. Examples of these operators are simulators and planners which take well formed domain specific models and derive behaviour from them. By definition, if these operators are implemented correctly, then they are behaviour preserving.

Less obvious are BPTs applicable to many different types of models. Such operators trade off predictive power for generality. They are of use in specializing models to particular questions and in decomposing large problems into computationally tractable subproblems.

Domain independent BPTs may be characterized by the type of operations they apply to models, and the types of guarantees of behavioural fidelity they provide. In this paper we discuss only BPTs which use delete operations (the other major class uses aggregate operations). Also, we are concerned with operations which provide formal guarantees of behavioural equivalence (instead of empirical guarantees).

Behaviour Preserving Pruning The Behaviour Preserving Pruning (BPP) operator is implemented in two parts. The first part specializes the dependency graph of the model to the particular question being addressed, and the second part decomposes this graph into strongly connected components.

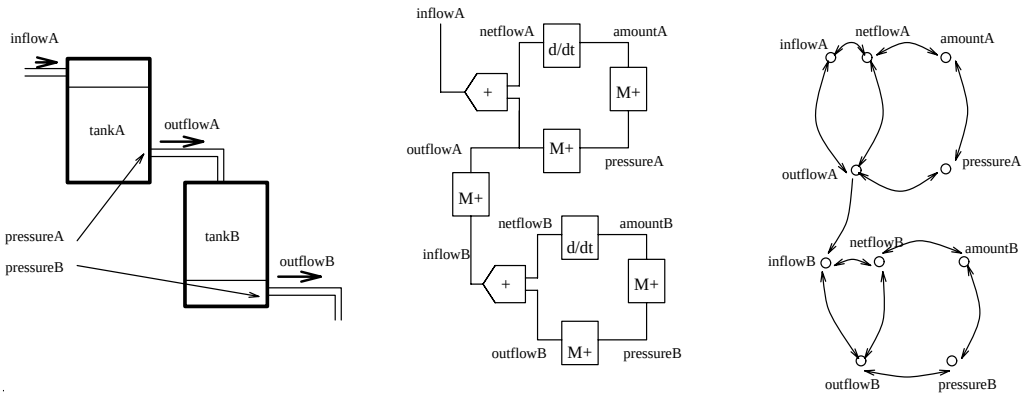

Figure 2: Two Tank System

Consider the model M of a pair of tanks each with one outlet and one inlet. The schematic and component representations are shown in Figure 2.

produces a single behaviour over the state space defined by adjoin(V^1, V^2) (where $adjoin(\{a,b,c\},\{d,b,e\}) = \{a,b,c,d,e\}$. Each pair of tuples (b_i^1, b_j^2) which agree on all shared variables $(V^1 \cap V^2)$ including time, produces a tuple in B^3. This operator allows us to express both simple concatenation of behaviour sequences (where the time points have no overlap and the variable sets are identical) as well as the reconstruction of partial behaviours (where the time sets do overlap and the variable sets are distinct).

The variable–domain transformation operator If V^1 is the variable set in B^1 with variable–domains D^1 and V^2 is the variable set in B^2 with variable–domains D^2, and a mapping $f_D : D^1 \rightarrow D^2$ exists from D^1 to D^2, then

$$B^2 = \mathrm{Dom}(f_D, B^1)$$

will yield behaviour in the new state space.

Other operators on behaviour such as aggregation operators which serve to collapse sets of variables to single variables (eg. spatial averaging) or to collapse sequences to single quantities (eg. time averaging) may also be defined.

Model–Behaviour Pairs

Behavioural and modelling knowledge are linked through their common references to variables. This relationship is made explicit by grouping models and behaviours into pairs (Model–behaviour pairs or MBPs).

$$MBP =< M, B >$$

where: M is a model of some system
B is behaviour syntactically consistent with M

Minimally, these pairs must correspond with respect to number and type of variables. However since M is a weakened representation of some domain specific model, B must also be consistent with this more restrictive modelling knowledge. One of the central ideas in this research is that domain independent model transformations can be defined which do not disturb this deeper relationship.

The notion that many types of reasoning can be expressed as operations on MBPs was presented briefly in the section on reasoning with models. In this view, a problem or question is simply an incomplete MBP, and problem solving involves using behaviour and model transformations to complete it.

We must extend the definition of a MBP in order to capture the ideas that questions are incomplete MBPs and that some of the variables might be uninteresting in the context of a specific question. The extension takes the form of a MBP augmented with a vector of error bounds. The error bounds specify the tolerance to which we wish to know each variable. For the purposes of this paper, tolerances are restricted to $\{\pm 0, \pm\infty\}$. Where

Time	interval 1		interval 2		interval 3	
Variable	amt	dir	amt	dir	amt	dir
Inflow	fnorm	std	fnorm	std	0	std
Netflow	(0,nmax)	+	(0,nmax)	std	(nmin,0)	+
Outflow	(0,omax)	+	(0,omax)	std	(0,omax)	-
Amount	(0,lmax)	+	(0,lmax)	std	(0,lmax)	-
Pressure	(0,pmax)	+	(0,pmax)	std	(0,pmax)	-

Table 1: Behaviour for a Single Tank

As an example of behavioural information Table 1 contains three state vectors for the single tank example presented in Figure 1. In the table, each tuple corresponds to a pair of columns where "amt" defines an interval representing quantity, and dir is direction or time derivative represented using the sign algebra ($\{-, \text{std}, +, ?\}$). The boundaries of the intervals (0, lmax, fmax, pmax...) are drawn from sets of "landmark" values which define the variable–domains of the variables in QP models (where an amount is represented by a single landmark its value is considered to be exactly equal to that landmark, so $fnorm = (fnorm, fnorm)$). The three intervals shown correspond to filling, steady state (inflow matches outflow) and draining (where the fixed flowrate source has been turned off).

Operations on Behaviours The basic domain independent operations on models consist of the relational operators join, project and select along with a mapping operator to convert behaviour conforming to one models schema into behaviour conforming to another models schema.

The Project Operator If V^1 is the variable set of the source behaviour B^1 and $V^2 \subset V^1$ is the variable set of the destination behaviour, then the operation

$$B^2 = \text{Proj}(V^2, B^1)$$

will drop those variables not in V^2 and will collapse adjacent states when they are identical to produce the destination behaviour B^2

The Select Operator If $O(V^1)$ is a function which maps state vectors from V^1 to $\{true, false\}$ then the select operator

$$B^2 = \text{Sel}(O, B^1)$$

will select only those tuples from B^1 for which O is true.

The Join Operator Finally, we may define a "join" operator to construct a single behaviour from two behaviours. If B^1 and B^2 are behaviours which share the same timeset then

$$B^3 = \text{Join}(B^1, B^2)$$

Notice that the dependencies in this model are all bi–directional. This is a characteristic of the particular domain (the qualitative physics (QP) of Kuipers [6]): as with arithmetic equations, there is no directionality implied by a QP constraint. More specific dependency information arises from causality arguments and from problem specific knowledge.

Operations on Models The basic operations on models involve adding and deleting model elements. Elements may be variables or constraints. By composing them in the obvious way, components may be added or removed.

The Cut Operator If $M^1 = (C^1, R^1)$ is a model and $c_i \in C^1$ then

$$M^2 = (C^2, R^2) = \text{cut}(M^1, c_i)$$

yields a model identical to M^1 but with component c_i removed from C^2 and all variable bindings in R^1 referring to the variables in c_i removed from R^2.

The Add Operator If $M^1 = (C^1, R^1)$ is a model then a component c may be added to it by specifying a relation of variable bindings $R' = \{(v_i, v_j)\}$ such that each tuple in R' contains at least one variable from c

$$\begin{aligned} M^2 &= \text{add}(M^1, c, R') \\ &= (C^1 \cup c, R^1 \cup R') \end{aligned}$$

Note that not all variables of c need to be mentioned in R' since they may be internal to c or may be input or output variables not connected to any other component.

Behaviour
Unlike models, the domain independent representation of behavioural knowledge is quite complete. This results from the fundamental role of behavioural information in reasoning about dynamic systems. We have to be able to at least compare behaviours at the domain independent level if we wish to select appropriate models, control reasoning or express goals with respect to system state. Fortunately, we can define quite general operators for editing, transforming and comparing behavioural knowledge so this need to represent behaviour completely is not too onerous.

We adopt a relational representation for behaviour in which each 2-tuple contains a timestamp (key) and a vector of values drawn from the variable–domains of the variables in the model to which the behaviour is attached. That is the behaviour B is represented as:

$$B = \{< S, t >\}$$

where:
- S is a vector of values drawn from the variable–domains of the variables v_i in V (alternatively, a token drawn from the state set X),
- t is drawn from the time set.

variable and an edge for each unique dependency. This is represented by the 3-tuple:

$$M' =< T, V, E >$$

where:
- T is the time set,
- V is the set of unique variables $\{v_{i'}\}$ one for each set of equivalent variables found in R,
- E is a binary relation $\{< v_{i'}, v_{j'} >\}$ representing all the unique dependencies between variables expressed in the components' edge sets S_i

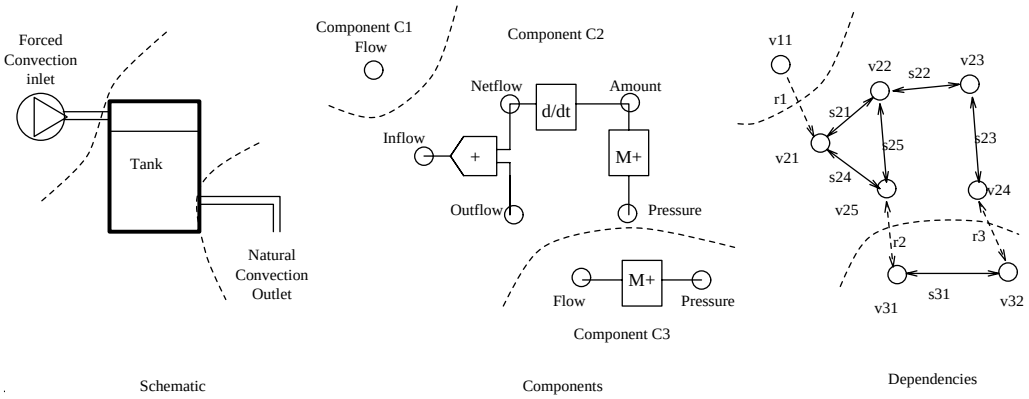

Figure 1: Single tank with two ports

In Figure 1 is a 3 component model of a simple hydraulic system. The schematic depiction of this model shows the three components: a pump which provides a constant inlet flow rate; a tank; and an outlet whos flow rate depends upon pressure.

In the component oriented depiction of the model the specific variables and constraints associated with each component are shown: the pump is represented by a single variable with no dependencies; the tank is represented by a mass balance constraint between the three flowrates, a derivative relationship between amount (level) and netflow, and a monotonic ($M+$) relationship between amount and pressure at the outlet; finally, the outlet component is a monotonic relationship between flowrate and pressure.

In the domain independent representation the constraints have been replaced with edges representing dependencies and the edges and variables have been labeled according to the scheme presented above. Thus $v_{i,j}$ is the j-th variable in component i, r_i is an equivalence edge between variables in different components, and $s_{i,j}$ is the j-th dependency in component i.

The variable oriented representation (not shown) would appear very similar to the component oriented dependency graph. The only differences would be that those sets of variables connected by equivalence edges (r_i) would collapse to single variables.

Models

Domain independence takes the form of a weak representation of constraints between variables. Where domain specific model representation languages express specialized constraints between variables, our language can express only dependency relations between variables. As a result, this language isn't expressive enough to support the derivation of behaviour from models. One must return to the domain dependent representation to perform tasks such as simulation or planning. However, the dependency based representation proves to be quite useful in guiding model transformations.

Another important feature of the domain independent representation is the ability to represent subsystems or components. This is especially important in the current context because we are interested in the relationships between multiple models of a single system in different modelling domains. Identification of common components is often the most natural way to express these relationships.

A third feature is representation of variables and the variable–domains from which values may be taken. Together these features allow variables to defined and interconnected by dependency relationships which may in turn be clustered into components. This yields two related representations: one emphasizing the component structure and the other emphasizing the variable/dependency structure.

For the component oriented view, a model M is defined as a 3-tuple:

$$M =< T, C, R >$$

where:
- T is the time set,
- C is the set of components in the model. Each component $c_i =< V_i, S_i >$ is a 2-tuple composed of a set of variables, $V_i = \{v_{i,j}\}$ and a set of directed edges between these variables $S_i = \{< v_{i,j}, v_{i,k} >\}$ representing dependencies.
  Each variable has a type which depends upon the domain of the underlying model $v_{i,j} : D$. A variable's variable–domain may be further restricted if such information is available,
- R is a set, $\{< v_{i,j}, v_{k,l} >\}$, of 2-tuples which captures the manner in which components are interconnected. Each tuple in R is composed of a pair of variables drawn from the components' variable sets and serves to equate them.

The variable oriented view may be derived from the component oriented view. The variables in the new representation include all the variables from the components in M but with the equivalent variables (defined in R) represented only once. All dependency edges are inherited, but redundancies may be removed. This yields a graph with a node for each unique

first question. Since the final form of the knowledge will be model centered, it follows that the development process must involve model construction, modification and verification.

In conclusion, implementation and maintenance of high level control systems can benefit from organizing knowledge into models. For each specialized model type, specialized representation and reasoning tools will be required. In order to integrate these specialized representations, we need a domain independent framework for keeping track of relationships between models, for selecting models for specific reasoning tasks, and for moving knowledge from one model to another.

MODEL BASED KNOWLEDGE ORGANIZATION

Model Based Knowledge Organization (MBKO) is an attempt to resolve certain problems surrounding the development and maintenance of KBS's by coercing knowledge into a representational framework reminiscent of that commonly used in dynamic, quantitative physical systems modelling.

A detailed knowledge representation scheme based on MBKO has been developed and presented by the authors elsewhere (Wylie and Kamel [12]). The current paper avoids most of the detail of this representation and instead focuses on the role of a domain independent model representation language and transformation operators to improve efficiency and flexibility in model based problem solving.

Reasoning with models

In the context of high level control many reasoning tasks may be expressed as either identification of a model which explains behaviour or derivation of behaviour implied by a model. Examples of this are:

- sensor data interpretation. Given a model of the system and a partial behavioural description (from sensors), derive a complete description of the system's state.
- diagnosis. Given a model of some system and behavioural information which doesn't correspond to the model, find the most plausible variant of that model which does explain the observations.
- planning. Given an incomplete description of the behaviour of the system (ie current state and goal state) and a model for the system, complete the behaviour using only feasible control actions.

LANGUAGE

This section presents a domain independent language for representing models, behaviour, and questions about dynamic physical systems.

Model selection and construction is supported by a set of model transformation operators. A special type of operator called a behaviour preserving pruning operator (BPP) is presented. It has the desirable property of simplifying a model without compromising its predictive power in the context of a specific question.

The rest of the paper is structured into 5 sections. The first presents an overview of the problem domain: high level control of large industrial systems. The second discusses the benefits of adopting a model centered view of knowledge in this (and other) application areas. The third presents the proposed language and operators. The fourth presents some examples of their use. The paper is concluded by a discussion of the work to date and the direction in which this research will continue.

HIGH LEVEL CONTROL

In the management and control of large industrial systems, there are a number of significant problems which are not amenable to solution by either conventional control techniques or MIS/DSS (Management Information System/ Decision Support System) software. Examples of these problems are: equipment and sensor diagnosis; process tuning (optimization); and planning changes in mode of operation of the plant (eg. Wylie and Kamel [12], Meystel [9]). These tasks are usually left to operators and plant engineers. Recently, an assortment of solutions to these types of problems have been demonstrated which exploit reasoning strategies derived from research in Artificial Intelligence (eg. Venkatasubramanian and Rich [11], Lakshmanan and Stephanopolous [7, 8], Sykes and Cochran [10], Gallanti et. al. [4]). While these systems have achieved reasonable technical success, the cost, risk, and expertise required for their development make them not yet commercially viable.

To reduce the cost, risk and expertise required to build and maintain such systems two related questions might be asked: what organizing principles underlie the sort of KBS's which solve these problems; and how are these high level control systems to be built efficiently and reliably.

The first of these questions can be partially answered by referring to the literature for examples. This will show that realistic solutions to these problems usually entail reasoning from multiple representations of the system being controlled. Examples of this include planners which exploit both a STRIPS–operator model and a detailed simulation model of the situation (Lakshmanan and Stephanopolous [7, 8]). Other examples include diagnostic systems which exploit both model and classification based representations (Venkatasubramanian and Rich [11]). Not only do individual reasoning tasks often require multiple models of the situation, but because of the integrated character of the high level control, distinct tasks must also be able to share modeling knowledge and pass control.

The answer to the second question, that of the methodology to be used in constructing such KBS's, follows directly from our answer to the

Model Based Knowledge Organization (NRC # 9999)

Rob Wylie† and Mohamed Kamel
Pattern Analysis and Machine Intelligence Lab, Department of Systems Design, University of Waterloo, Waterloo, Ontario, Canada, N2L-3G1
†*On educational leave from the Knowledge Systems Lab, Institute for Information Technology, National Research Council of Canada, Ottawa, Canada, K1A-0R6*

ABSTRACT

Reasoning about the behaviour of large industrial processes is a difficult and complex computational task. To perform it efficiently requires the use of numerous specialized representations of the system. Any successful information processing system in such an environment must have effective ways to manage these representations, the individual reasoning tasks which use them, and the flow of information between them. This paper addresses these issues by proposing a model centered view of knowledge and presents a domain independent representation language along with operators for managing multiple overlapping models of dynamic physical systems. Examples are given of its use in sensor interpretation, diagnosis and planning in a simple pipe–and–tank network.

INTRODUCTION

This paper presents a domain independent scheme for representing and manipulating models of dynamic physical systems. It is intended for applications in which one physical system is the focus of a diverse set of reasoning tasks. To be useful in such an environment, the representation must support dynamic model selection and construction as well as communication between tasks using different types of models.

Communication between tasks across modelling domains is directly supported by a uniform representation for models because it allows matching of components and variables across domains. This in turn provides a foundation for communicating behavioural, structural and control information across domain boundaries.