Structured Modeling Language for Automated Modeling in Causal Networks

Yousri El Fattah
Rockwell Science Center
1049 Camino Dos Rios
Thousand Oaks, CA 91360
yousri@rsc.rockwell.com

Abstract

The paper presents a structured modeling language (SML) and a relational database framework for specification and automated generation of causal models. The framework describes a relational database scheme for encoding a library of causal network templates modeling the basic components in a modeling domain. SML provides a formal language for specifying models as structured components that can be composed from the basic components. The language enables specification of models as parameterized relational queries that can be instantiated for specific model instances. The paper describes an algorithm that, given a library and a specification, computes a causal model in time and space linear in the number of basic components. The algorithm enables model reuse by combining model fragments from the template library to compose new models. The present automated modeling approach has been implemented using the structured query language (SQL) and a relational database environment. The approach has been successfully used for modeling an automated work-cell in a real-life digital manufacturing application.

1 Introduction

Automated modeling is a key for developing automated reasoning applications in domains such as industrial automation and digital manufacturing. The models can be used to support reasoning tasks for two main objectives. One, to reduce the commissioning time by simulating, verifying and validating the intended function of a system before it is built. Two, to reduce downtime after a system is built by diagnosing and identifying faults. Building adequate models is typically a time consuming process involving much iteration and requiring in-depth knowledge of the particular domain.

The models considered here are in the form of causal networks [Pearl, 1988; Dechter and Pearl, 1991; Darwiche and Pearl, 1994]. Causal networks represent cause-and-effect relationships in a system by a directed acyclic graph in which nodes are the variables and the edges represent direct causal influence. In addition to the graph, causal network models include quantification of the causal influence. Bayesian networks [Pearl, 1988] quantify the causal influence by conditional probabilities that are attached to each cluster of parents-child nodes in the network. Here we adopt deterministic quantification in the language of directed constraint networks [Dechter and Pearl, 1991].

The significance of the problem of automated modeling in causal networks is well-known [Laskey and Mahoney, 1996]. Recently, a number of approaches have been proposed based on object-oriented languages, which allow complex domains to be described in terms of interrelated objects [Koller and Pfeffer, 1997; Laskey and Mahoney, 1997]. The focus of those works is on the representation of probabilistic knowledge as network fragments and not on algorithms for constructing models from the knowledge base. The contributions of this paper are a relational framework for encoding the causal relationships in a modeling domain; a language for model specification and an algorithm for constructing causal models.

The paper is organized as follows. Section 2 describes SML relational framework and the concept of a template library. Section 3 gives the model specification language and a detailed example. Section 4 describes the algorithm for constructing causal models. Sections 5 presents discussion and related work and Section 6 concludes.

2 Structured Modeling Language

SML enables specification of models using a library of basic components. Basic components are the building blocks in a modeling domain. Examples of basic components in the industrial automation domain are a proximity switch, a cylinder, a valve, and a solenoid actuator. We start with a set of definitions then formalize our concept of model library and then give an example.

Definition 1 [Relational Databases] [Maier, 1983] Let $U = \{U_1, ..., U_n\}$ be a set of attributes, each with an associated domain. A *relational database scheme* $\mathbf{R}$ over U is a collection of relation schemes $\{R_1, R_2, \ldots, R_p\}$, where $\bigcup_{i=1}^{p} R_i = U$, and $R_i \neq R_j$, if $i \neq j$. A *re-*

lational database d on database scheme **R** is a collection of relations $\{r_1, r_2, \ldots, r_p\}$. Each *relation* r_i on relation scheme R_i, written $r_i(R_i)$, is a set of value tuples $\{t_1, t_2, \ldots, t_p\}$ for the attributes in R_i from their respective domains, and we write $t.X$ to denote the value assigned by a tuple t to a subset $X \subset R_i$. A *key* of a relation $r_i(R_i)$ is a minimal subset K_i of R_i that uniquely identifies a tuple within the relation. A *foreign key* is a set of attributes within one relation that matches the key of some (possibly the same) relation. A relation scheme may contain more than one key and the keys explicitly listed within a relation scheme are called *designated keys*. We distinguish one of the designated keys as the *primary key*. To denote the primary key of a relation, we underline the attribute names in the key. Let X and Y be subsets of R_i; we say that a relation $r_i(R_i)$ satisfies the functional dependency (FD) $X \rightarrow Y$ if for every X-value x, there corresponds only one value for Y, i.e., the set $\{t.Y \mid t \in r_i \wedge t.X = x\}$ has at most one tuple. A relation scheme R_i embodies the FD $K_i \rightarrow R_i$ if K_i is a designated key for R_i. A database scheme **R** *represents* the set of FDs $G = \{X \rightarrow Y \mid$ *some* $R_i \in$ **R** *embodies* $X \rightarrow Y\}$. **R** *completely characterizes* a set of FDs F if $F \equiv G$.

Definition 2 [Causal Networks] Let $U = \{U_1, ..., U_n\}$ be a set of attributes, each with an associated domain. A *causal network* is a pair (G, d) where G is a directed acyclic graph whose nodes are U and $d = \{r_1(R_1), \ldots, r_p(R_p)\}$ is a relational database such that: (1) each scheme R_i corresponds to a family in G, i.e., a set consisting of a child node X_i and all its parent nodes $pa(X_i)$, (2) d completely characterizes the set of FDs, $\{pa(X_i) \rightarrow X_i \mid i = 1, \ldots, p\}$[1].

The above database definition of causal networks maps directly to that of directed constraint networks [Dechter and Pearl, 1991]. The mapping from the database representation to that in propositional logic [Darwiche and Pearl, 1994] is also straightforward. For each relation r_i on the scheme $R_i = \{X_i\} \cup pa(X_i)$ do: for each tuple $t \in r_i$ output the propositional sentence:

$$\left(\bigwedge_{Y \in pa(X_i)} (Y = t.Y) \right) \supset (X_i = t.X_i)$$

Since each r_i embodies the FD: $pa(X_i) \rightarrow X_i$, the propositional sentences above are ensured to be consistent.

2.1 Template Model Library

The library describes the structure and function of the basic components in a given modeling domain. Basic components are modeled using templates of causal network fragments encoded in a relational database. The structure of a causal network is described by a set of families in a directed graph. Each family may have a distinguished parent called assumption. A basic template consists of a directed acyclic graph (dag) G, and a set of functional relations $\{r_1(R_1), \ldots, r_n(R_n)\}$ defined on the dag's families $R_1, .., R_n$ for all the non-root nodes $X_1, .., X_n$; $R_i = \{X_i\} \cup pa(X_i)$. The structure of a basic template of type B is defined by its relational scheme,

$$scheme(B) = \{R_i \mid i = 1, \ldots, n; \\ R_i = (X_i, pa(X_i)); pa(X_i) \neq \emptyset\} \quad (1)$$

We now describe the template library encoding of the causal relationships. The encoding consists of two relational tables: *struc* and *func*. The *struc* scheme is:

$$struc(\underline{Sid}, Type, AssumDomId, \\ InVar_1, InVar_1_DomId, \ldots, InVar_n, \\ InVar_n_DomId, OutVar, OutVar_DomId) \quad (2)$$

Each tuple in the struc relation corresponds to a family in a causal network fragment. It is identified by a structure id, *Sid*, and is associated with a basic component type, *Type*. The family consists of the output variable, *OutVar* which is the child, and the input variables, *InVar_1*, ...,, *InVar*$_n$, which are the parents. The domains of those variables are indexed by the integer attributes: *InVar_1_DomId*, ..., *InVar_n_DomId*, *OutVar_DomId*. The number of input variables, n, is determined by the size of the largest family in the modeling domain. Families with m parents, $m < n$, will have null values for all inputs with index greater than m. *AssumDomID* is an index identifying the assumption domain. If a family includes no assumption then *AssumDomID* is null. The *func* relation scheme is:

$$func(Sid, Assum, InVal_1, \ldots, InVal_n, OutVal) \quad (3)$$

Sid is a foreign key referencing the structure id in the *struc* relation.

The library defines the value domains of the various attributes in a table having the scheme,

$$dom(\underline{DomId}, Val_1, \ldots, Val_n, Cost_1, \ldots, Cost_n) \quad (4)$$

DomId is the domain identifier, and the allowed values in the domain are: *Val_1*, ..., *Val_k*. Here, k is a parameter determined by the maximum number of values in a domain. Each value *Val_i* has an associated cost, which is a non-negative integer. Only the domains of the assumption attributes have associated costs; the costs for all other domains have the default value zero. The domains for the assumption attributes provide the fault information required for model-based diagnosis [El Fattah and Dechter, 1995].

The causal relation r on the scheme (*Assum*, *InVal_1*, ..., *InVal_n*, *OutVal*) of a sub-component $t \in struc$ is determined by the set of tuples $u \in func$ having the foreign key equal $t.Sid$. That is, $r = \{u \in func \mid u.Sid = t.Sid\}$. The relation r satisfies the functional dependency,

$$Assum, InVar_1, \ldots, InVar_n \rightarrow OutVar \quad (5)$$

The relation provides the value of the output variable as a function of the values of the assumption and the input variables.

[1] The tuples in a relation r_i do not have to cover all value instantiations of the parent variables. This means that the quantification of the causal relation can be incomplete. The more complete the quantification the more specific the predictions that can be made using the model.

Table 1: *struc* relation for cylinder.

SID	Comp_ Type	Assum_ Dom	In_ Var1	In_ Dom1	In_ Var2	In_ Dom2	Out_ Var	Out_ Dom
11	Cylinder	4	Ext_P	1	Ret_P	1	Pos	8
12	Cylinder		Pos	8			Retd	1
13	Cylinder		Pos	8			Extd	1

Table 2: *dom* relation for cylinder.

DomID	Val1	Val2	Val3	Cost1	Cost2	Cost3
1	on	off				
4	ok	stuck_extd	stuck_retd	0	1	1
8	extd	retd	neut			

2.2 Example

Figure 1 shows an example of a basic template in the industrial automation domain. The figure depicts the schematic and the dag representation of the template's structure for a basic component of type cylinder. A cylinder has two inputs (root nodes) corresponding to the extend and retract pressures: **EXT_P** and **RET_P**. There is one assumption variable, **ASSUM** (also a root node), which represents the mode of operation of the cylinder. The cylinder's piston rod position, **POS** causally depends on the input pressures and the assumption. The cylinder has two outputs (the leaf nodes), which are the extended **EXTD** and retracted **RETD** positions of the piston. Table 1 shows the tuples in the *struc* relation for cylinder. The values and the costs for the various domain indexes are given in the *dom* table shown in Figure 2. Table 3 shows a partial set of tuples for cylinder in the *func* relation. The first tuple says that if the cylinder is *ok* and both pressure inputs are *off* then the piston position is neutral.

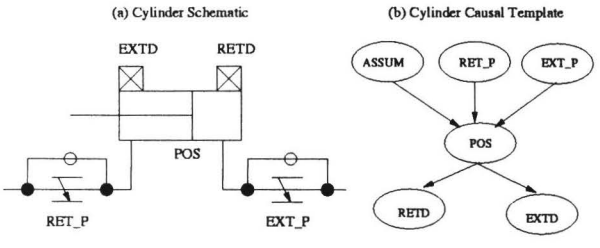

Figure 1: Basic template for a cylinder: (a) schematic, (b) dag of causal network fragment.

3 Model Specification Language

SML specification of models is based on the concept of structured template. Structured templates represent components that are composed of basic components. Examples of decomposable components in the industrial automation domain are work cells, fixtures, clamps, pins and dumps. We start by defining a structured template, then formalize the specification language and give an example.

Table 3: Partial *func* relation for cylinder.

SID	Assum	In_ Val1	In_ Val2	Out_ Val
11	ok	off	off	neut
11	ok	off	on	retd
11	ok	on	off	extd
11	ok	on	on	neut
13		extd		on
13		neut		off
13		retd		off

3.1 Structured Template

A structured template is specified by the basic templates of its sub-components and by the connections between those sub-components. Formally, a structured template composed of m basic templates can be specified as a pair: (S, C). S is an ordered tuple of the families of the m basic templates,

$$S = \langle (X_{11}, pa(X_{11}), \ldots, (X_{1n_1}, pa(X_{1n_1}), \ldots, (X_{m1}, pa(X_{m1}), \ldots, (X_{mn_m}, pa(X_{mn_m})\rangle \quad (6)$$

C is a set of pairs representing the structural connections between the basic templates:

$$C = \{(X_i, X_{k_j}) \mid i < j, X_{k_j} \in pa(X_j)\} \quad (7)$$

Each element in C is a connection from a node X_i which is the child in some family $(X_i, pa(X_i))$ to a parent node X_{k_j} in another family $(X_j, pa(X_j))$ in S such that i precedes j.

3.2 Specification Language

Figure 2 gives the grammar of SML model specification language in EBNF (Extended Bacchus Naur Form). Terminals are enclosed in double quotes, non-terminals are in italic, anything between square brackets are optional. The grammar says that a structured component is specified by one or more sub-components. Each

structured_template ::= *sub_comp_list*

sub_comp_list ::= *sub_comp_entry* | *sub_comp_entry* *sub_comp_list*

sub_comp_entry ::= *sub_comp_out* "=" "sub_comp" "(" *struc_comp_label* "," *basic_comp_label* "," *struc_id* ["," *connection*] ")"

connection ::= *from* | *from* "," *connection*

from ::= *sub_comp_out*

sub_comp_out ::= *string_value*

struc_comp_label ::= *string_value*

basic_comp_label ::= *string_value*

struc_id ::= *integer_value*

Figure 2: SML specification grammar

sub-component is specified by the function, *sub_comp*. The parameters of that function are: the structured component label, *struc_comp_label*; the basic component label, *basic_comp_label*; the basic template structure id, *struc_id*; and the connections from other sub-components. A connection consists of one or more references *from* the outputs of parent sub-components. The *sub_comp* function returns a reference to the output of the defined sub-component which can then be used as the "from" attribute to another child sub-component. The connection in a sub-component definition is optional. If included then the outputs of the parent sub-components are used to replace the inputs in the defined sub-component. Else, the inputs to the defined sub-component are determined by the *struc* relation tuple identified by *struc_id*.

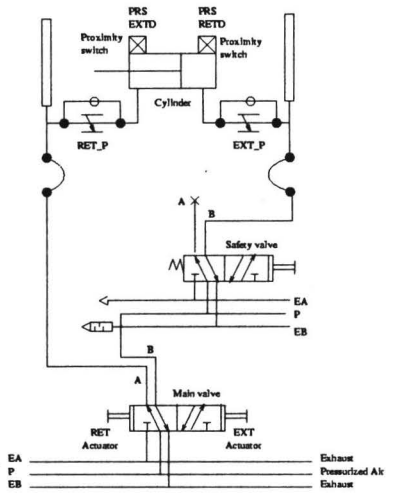

Figure 3: Clamp pneumatic circuit

Model specification is encoded in a relational table with the following scheme,

$$spec(\underline{SCid}, SCLabel, SubCompLabel, Sid, From_1, \ldots, From_n) \quad (8)$$

SCid denotes a structured component id and is the primary key. *SCLabel* is the structured component label. *SubCompLabel* is the label of a sub-component whose type is identified by the structure id, *Sid*. *From_1*, ..., *From_n* are foreign keys consisting of the SCid's of the parent sub-components whose outputs are connected to the respective inputs of the basic structure template identified by *Sid*. The function *sub_comp* in the SML specification has the side effect of adding a tuple representing the sub-component entry to the *spec* table. The function then returns the *SCid* of that tuple which can be used to encode the "From" connections as explained earlier.

3.3 Example

Figure 3 shows the pneumatic circuit for an industrial clamp. The function of a clamp is to hold a machine part in place on a fixture during operation on the part, e.g., a robot welding operation. The clamp has two landmark positions; namely the extended and retracted positions. The extended (retracted) position is produced by the control valves that connect the extend (retract) port of the cylinder to pressurized air and the retract (extend) port to the exhaust. The circuit shows two pneumatic valves connected in series. The main valve has two solenoid actuators: one for extend and the other for retract. The safety valve is a spring-return valve with only one actuator for extend. For the clamp to extend (retract), the extend actuators of both valves must be on (off) and the retract actuator of the main valve must be off (on). The extend and retract position of the clamp is sensed by proximity switches that sense the piston rod at the extended and retracted positions.

Figure 4 shows the structured template specification of a clamp set. The subcomponents of a clamp set include power, air, actuators, valves, cylinders and proximity switches. Each clamp in the set is associated with a cylinder and all cylinders are actuated by the same valves. The specification is stated as an SQL procedure whose parameters are the clamp set label, *Comp*, and a variant array of cylinder labels. Each sub-component entry in the specification references a structure id, *SID*. For example the first sub-component, Power, has $SID = 15$, which references a basic component of type power in the *struc* table. Naturally, the SID values are implementation dependent. The next subcomponent entries are: air, the extend and retract actuators of the main valve, and the extend actuator for the safety valve. Note that those components have no *from* arguments which means that their inputs are exogenous variables coming from outside the clamp set. The next three subcomponent entries specify the main valve and the next two specify the safety valve. The main valve position, *Vlv1_Pos*, has two inputs coming from the main extend and retract actuators. The main valve pressure lines A and B both take input from the the main valve position. The next two entries for the safety valve are similar. The equation for the safety valve B output *Vlv2_B* takes an input from the main valve B output *Vlv1_B*, which implies in-series

```
Sub Create_ClampSet(Comp As String,
            ParamArray Cylinders() As Variant)
 Power = SubComp(Comp, "Pow", 15)
 Air = SubComp(Comp, "Air", 16)
 Act_Ext_1 = SubComp(Comp, "Ext", 14)
 Act_Ret = SubComp(Comp, "Ret", 14)
 Act_Ext_2 = SubComp(Comp, "Safety_Ext", 14)
 Vlv1_Pos = SubComp(Comp, "Vlv", 5,
            Act_Ext_1, Act_Ret)
 Vlv1_A = SubComp(Comp, "Vlv", 6, Vlv1_Pos, Air)
 Vlv1_B = SubComp(Comp, "Vlv", 7, Vlv1_Pos, Air)
 Vlv2_Pos = SubComp(Comp, "Saf_Vlv", 8, Act_Ext_2)
 Vlv2_B = SubComp(Comp, "Saf_Vlv", 10,
            Vlv2_Pos, Vlv1_B)
 For intI = 0 To UBound(Cylinders())
   Cyl = "Cyl_" & Cylinders(intI)²
   Cyl_Pos = SubComp(Comp, Cyl, 11, Vlv2_B, Vlv1_A)
   Cyl_Extd = SubComp(Comp, Cyl, 13, Cyl_Pos)
   Cyl_Retd = SubComp(Comp, Cyl, 12, Cyl_Pos)
   Prox_Extd = SubComp(Comp, "PRS_Extd_" & Cyl,
            1, Cyl_Extd, Power)
   Prox_Retd = SubComp(Comp, "PRS_Retd_" & Cyl,
            1, Cyl_Retd, Power)
 Next intI
End Sub
```

Figure 4: Specification of a clamp set

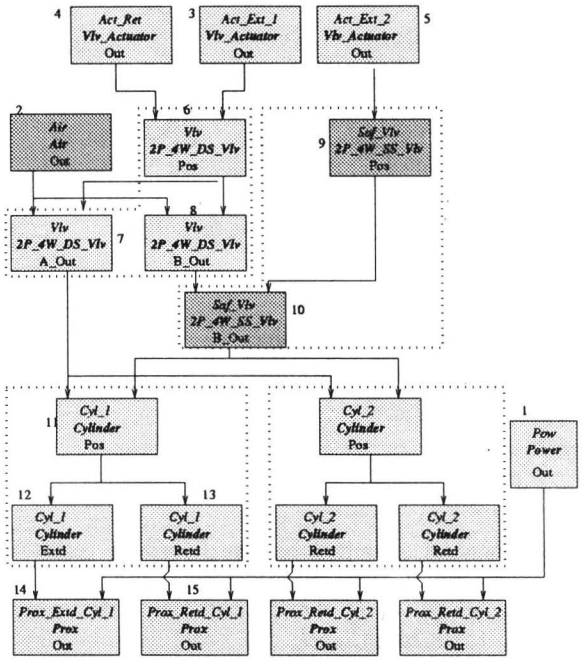

Figure 5: Clamp structured template

connection as indicated by the schematic in Figure 3. The "for" loop in the procedure creates for each element in the cylinders list five subcomponent entries: Three entries for the cylinder and two for the cylinder-extended and retracted proximity switches. The air inputs to each cylinder come from the main valve A line and the safety valve B line as in the schematic in Figure 3.

The subcomponent entries in the specification are ordered such that all connections to the inputs of a subcomponent come from the outputs of sub-components that are earlier in the ordering. For example, *Vlv1_Pos* has its inputs coming from the earlier sub-component outputs: *Act_Ext_1*, *Act_Ret*. Figure 5 depicts the specification structure for a clamp set instance having 2 cylinders: *cyl_1*, *cyl_2*. The figure shows the ordering of the sub-components according to the structured template specification of Figure 4. Each sub-component is represented as a box; the name of the sub-component is shown in italics; the type is in italics and bold; and the output is shown at the bottom of the box. The figure shows that a basic component (e.g., a cylinder) consists of multiple sub-components, which are the causal network families.

4 Model Generation

Model generation requires two information sources: (a) the template library namely the *struc* and *func* relations, and (b) the model specification in the form of a *spec* relation. Causal models can be generated in some predefined format such as symbolic or relational. Relational format represents the causal model by a relational database where each relation corresponds to a causal network family, $(X_i, pa(X_i))$. The name of the relation is the child variable, X_i, and the attributes of the relation are the family variables, $\{X_i\} \cup pa(X_i)$. The symbolic format represents the equations for each family as a set of propositional logic sentences [Darwiche and Pearl, 1994].

The model generation algorithm is given in Figure 6. The algorithm takes as input a library of basic components, specification of a structured component instance, *SCLabel*, an output model format, *OutFormat*, and an optional assumption list, *AssumList*. The algorithm outputs the causal model in the required format. The assumption list consists of basic component types whose assumptions need to be included in the model. If no list is given then the model will be based on the default assumption *ok* for all basic component types. The algorithm selects the specification tuples for the structured component instance (step 1) and computes for each tuple s a causal relation r which is then output in the assigned format (steps 2-6). Step 3 selects the structure template tuple $t \in struc$ having same structure id, *Sid* as tuple s. Step 4 determines the scheme for the causal relation. The step prefixes the sub-component label to the assumption, input and output variables. It caches the name of the output variable in the temporary *out* table. If there are connections then the input variables are replaced by the proper connections from the *out* table. An assumption is included if the structure templates mentions an assumption and the sub-component type is a member of *AssumList*. Step 5 populates the causal relation with the function template tuples having same structure id, *Sid* as tuple t. The time and space com-

plexity of the algorithm is $O(n \cdot k)$, where n is the number of sub-components and k is the maximum number of tuples in the functional relation of the causal families.

Algorithm Create Causal Model

Input library relations: *struc*, *func*; specification *spec*; model data: *sc_label*, *out_format*, *assum_list*.

Output Causal model.

Initialization $out(\underline{SCid}, OutVar) \leftarrow \emptyset$

1. Select specification tuples $S \leftarrow \{s \in spec \mid s.SCLabel = sc_label\}$
2. For each $s \in \sigma_{sc_label}\ spec$ do
3. Get the basic component structure tuple, $t \in struc$ whose key is $s.Sid$
4. Compute causal relation scheme, $(X, pa(X))$:
   (a) $X \leftarrow s.SubCompLabel\ \&\ t.OutVar$
   (b) Add the tuple $(s.SCid, X)$ to the *out* relation.
   (c) if $t.Type \in assum_list \wedge \neg\ IsNull(t.AssumDomId)$
   then **/* add assumption */**
   $pa(X) \leftarrow pa(X) \cup s.SubCompLabel\ \&$ "_assum"
   else **/* no assumption */**
   $pa(X) \leftarrow \emptyset$
   (d) For $i = 1$ to n do
   if $IsNull(t.from_i)$
   then **/* no connection */**
   $pa(X) \leftarrow pa(X) \cup \{s.SubCompLabel\ \&\ t.InVar_i\}$
   else **/*enforce connection*/**
   $p \leftarrow \{p \in out \mid p.SCid = s.SCid\}$
   $pa(X) \leftarrow pa(X) \cup \{p.OutVar\}$
5. Get the causal relation r on the scheme $(X, pa(X))$, $r \leftarrow \{u \in func \mid u.Sid = t.Sid\}$
6. Output r in the representation, *out_format*.
7. End for.

Figure 6: Algorithm Create Causal Model

5 Discussion and Related Work

The present automated modeling approach has been implemented using the structured query language (SQL) and a relational database environment. The approach is used to model the clamping fixtures in a real-life robot-welding workcell. The fixtures consist of multiple sets of pins, dumps, locators, and clamps having pneumatic circuits similar to the one in Figure 3. One feature of our modeling approach is the explicit representation of assumption attributes. This is essential for building models for diagnosis [El Fattah and Dechter, 1995]. The models are used to generate model-based diagnostic code to run on the programmable logic controller that controls the workcell [Provan *et al.*, 1998].

Previous work [Falkenhainer, 1991] proposed *compositional modeling*, a framework for constructing adequate device models from a model fragment library which explicitly represents modeling assumptions. The framework takes a domain theory, a structural description of a specific system, and a query about the system's behavior and searches the space of possible models to compose the right model for the query. Recent work on causal explanation [Nayak, 1994] proposed the notion of causal approximation and showed that compositional modeling becomes tractable when all model fragment approximations are causal approximations. Our approach does not represent modeling assumptions and each component type is represented by exactly one model. This is not an inherent limitation in our approach and we plan to extend it by: (1) including modeling assumptions in the relational schema of the fragments, (2) incorporating compositional modeling constraints in the library in the form of SQL stored procedures. We can assign costs to various model fragments and modify our model construction algorithm to compose minimum cost models.

Specifying the model template library as a relational database enhances the maintenance and reconfiguration of models. The database encoding enables adaptability and scalability to new and unforeseen requirements and applications. New fields and record types can be added to the library database without affecting current automated modeling programs. Our database approach enhances the reliability of the modeling data through the use of proper integrity constraints, e.g., key integrity, domain integrity and referential integrity.

The focus of previous work on automated modeling in bayesian networks has been primarily on the representation of probabilistic knowledge as network fragments and not on algorithms for constructing models from the knowledge base [Laskey and Mahoney, 1997]. A main contribution of our approach is the description of an efficient algorithm for automated generation of causal models. The algorithm can be stated declaratively in SQL, which enhances the expressiveness and naturalness of model-building operations. The ubiquity of database environments and industry standard SQL servers are two factors in favor of our database approach to automated modeling.

An advantage of our approach is the ability to describe models with a generalized number of sub-components. For example, our clamp set model includes a variable number of clamps in the template which can then be instantiated for generating specific models. See Figure 4. The OOBN language [Koller and Pfeffer, 1997] cannot represent models with varying number. For example, the OOBN approach cannot represent a model of a car accident with a variable number of passengers; instead it will represent a distinct model for each possible number of passengers [Koller and Pfeffer, 1997]. Also, the OOBN language is poorly equipped to represent global constraints on the sub-components. For

example, it cannot represent that a car contains three passengers, at least one of whom is a child [Koller and Pfeffer, 1997]. Our relational approach can express such global constraints as global consistency queries. The key to the expressive power of our approach is that structured model templates are represented as parameterized relational queries that can be instantiated to generate specific model instances.

The present modeling framework can be extended to represent temporal causal networks [El Fattah and Provan, 1997]. This can be done by classifying causal relations in two classes: instantaneous and delayed and by including temporal data for delayed relations. The value of the output of a causal relation at any time t is determined by the values of the inputs at t if the relation is instantaneous and at $t - d$ if the relation is delayed, where $d > 0$ is the time delay.

Our structured template specification requires a causal ordering of the subcomponents, similar to those used to describe the operation of physical devices [Kuipers, 1984; Iwasaki and Simon, 1986; de Kleer and Brown, 1986]. Previous work on bond graphs has also developed methods for causal ordering from a compositional modeling perspective [El Fattah, 1996]. Causal ordering is orthogonal to our modeling approach. We assume that all basic components consist of causal constraints [Dechter and Pearl, 1991] with pre-defined input-output directionality. The causal structure of our structured components is determined by the directed connections between its subcomponenets. If the connections and the basic constraints are non-directional then a causal ordering algorithm can be employed and the directional constraints are then cached in our database library.

6 Conclusions

The paper presents a structured modeling language for automated modeling in causal networks. The language provides a formal method for specification of models based on a library of basic components. The library is represented by two relations: *struc* and *func*. The *struc* relation describes the templates for the structure of basic components, which determines the causal relation scheme and the associated causal network families. The *func* relation contains the value tuples for the causal relations of the basic components. The specification of models is done by instantiating structured component templates defined in SML. The information required for instantiating the templates can be acquired automatically from the schematic representation, e.g. the wiring diagrams, of the modeled system. The modeling knowledge is captured and maintained in the basic component library, which is then reused to generate causal models. The paper describes an efficient algorithm for the automated generation of causal models from specification. The approach has been implemented and successfully used to model an automated work-cell in a real-life digital manufacturing application.

References

[Darwiche and Pearl, 1994] A. Darwiche and J. Pearl. Symbolic causal networks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 238–244. AAAI Press/ The MIT Press, 1994.

[de Kleer and Brown, 1986] J. de Kleer and J.S. Brown. Theories of causal ordering. *Artificial Intelligence*, 29:33–62, 1986.

[Dechter and Pearl, 1991] R. Dechter and J. Pearl. Directed constraint networks: A relational framework for causal modeling. In *Proceedings, IJCAI-91*, pages 1164–1170, 1991.

[El Fattah and Dechter, 1995] Y. El Fattah and R. Dechter. Diagnosing tree-decomposable circuits. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Quebec, Canada, 1995.

[El Fattah and Provan, 1997] Y. El Fattah and G. Provan. Modeling temporal behavior in model-based diagnosis of discrete-event systems (a preliminary note). In *Working Notes of the Eighth International Workshop on Principles of Diagnosis*, pages 43–50, Le Mont-Saint-Michel, France, 1997.

[El Fattah, 1996] Y. El Fattah. Constraint logic programming for structure-based reasoning about dynamic physical system. *Artificial Intelligence in Engineering*, 10(3):253–264, 1996.

[Falkenhainer, 1991] B. Falkenhainer. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:59–143, 1991.

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

[Koller and Pfeffer, 1997] D. Koller and A. Pfeffer. Object-oriented bayesian networks. In *Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313, 1997.

[Kuipers, 1984] B. Kuipers. Common sense reasoning about causality: Deriving behavior from structure. *Artificial Intelligence*, 24:169–203, 1984.

[Laskey and Mahoney, 1996] K. B. Laskey and S. M. Mahoney. Network engineering for complex belief networks. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 389–396, 1996.

[Laskey and Mahoney, 1997] K. B. Laskey and S. M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *Uncertainty in Artificial Intelligence (UAI-97)*, pages 334–341, 1997.

[Maier, 1983] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, MD, 1983.

[Nayak, 1994] P. Pandurang Nayak. Causal approximations. *Artificial Intelligence*, 70:277–334, 1994.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, CA, 1988.

[Provan *et al.*, 1998] G. Provan, Y. El Fattah, and A. Darwiche. Using database specifications to automate the diagnosis of factory automation systems. In *Proceedings of the International Automative Manufacturing (IAM) Conference*, pages 323–328, 1998.