Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

A RELATIONAL ALGEBRAIC FRAMEWORK FOR MODELS MANAGEMENT

Hyu Chan Park
Wan Bok Lee
Tag Gon Kim

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-Dong Yusong-Ku, Taejon 305-701, KOREA
(*e-mail: hcpark@coregate.kaist.ac.kr*)

ABSTRACT

Models management with a large number of models has placed demands on a highly structured and rigorous framework. This paper proposes a new framework, called the RASES (Relational Algebraic System Entity Structure), for the models management. It is based on the concepts of system entity structure (SES) and relational algebra (RA). SES is accepted as a conceptual basis to organize a family of hierarchical structures of models. Under the RASES, SES itself is represented in the form of relations which may be stored as tables in a relational database. Furthermore, several operations on SES can be formulated in terms of relational algebra which can be coded in a standard query language.

1 INTRODUCTION

Modeling of complex systems and models management with a large number of models have placed demands on highly structured and rigorous framework. To control the complexity of the modeling and models management, the structures of models must be separately managed from the behaviors of models. Furthermore, models must be specified in a modular form so that a composite model can be hierarchically constructed just by coupling input/output ports of the models (Zeigler 1984). The structure of such a hierarchical model is termed as a *model structure*. Hierarchical management of such models and unified representation of the model structures are essential to efficiently construct models of complex systems. Therefore, our *models management problem* concerns how to manage models in a library of models, how to organize a family of model structures in a unified form, and how to construct new models from existing models and model structures.

This paper proposes a new formalism, called *relational algebraic system entity structure* (RASES), to cope with the models management problem. It is based on the *system entity structure* (SES) formalism (Kim et al 1990, Sevinc and Zeigler 1988, Zeigler 1984, Zhang and Zeigler 1989) and the *relational algebra* (RA) formalism (Codd 1970). The SES formalism is adapted as a conceptual basis to organize a family of model structures in a unified representation. The RA formalism is accepted as a vehicle to manage the models and model structures within a relational database. Several applications in different domains (Kim 1990, Sevinc and Zeigler 1988) have proved that SES can be successfully employed for the models management. It provides a unified representation scheme within which one may systematically integrate a family of model structures and extract new model structures from it. SES itself, in turn, can be formulated in terms of relations under the RA formalism. Furthermore, several operations on SES, such as the pruning operation, can be defined in relational algebra.

The relationally formalized SES (RASES) can exploit the power of *relational database*. Relational database has been applied successfully to several applications, and accepted as a method to overcome limits of traditional approaches. In the models management problem, there may be a large number of models and model structures which must be stored to provide sharable repository for many modelers. This problem can be alleviated by the database approach because *relational database management system* (RDBMS) provides rich facilities to efficiently manage large amounts of data. As a consequence, the RDBMS is an attractive platform for implementing a framework for the models management.

In the work described here, we are developing a RASES framework for the models management on a general purpose RDBMS, the INFORMIX. Using the facilities of the framework, modelers can create models which meet their modeling objectives, and conduct appropriate experiments on the models. Furthermore, the framework is sufficiently general enough that it can be extended to manage the knowledge of other domains, and to serve somewhat different needs of users.

This paper is organized as follows. Section 2 briefly reviews SES with a more formalized form of SES. In Section 3, we propose a relational formalization of SES with a relational algebraic pruning algorithm. Section 4 discusses an implementation of the RASES framework on a RDBMS, the INFORMIX, and section 5 illustrates a simple example with buffer-processor models. Finally, some concluding remarks are in Section 6.

2 SYSTEM ENTITY STRUCTURE

SES, proposed by Zeigler, is a declarative knowledge representation scheme which systematically organizes a family of possible alternatives for a system's structure. Such a family characterizes decomposition, coupling, and taxonomic relationships among entities. The entity represents a real world object. The decomposition concerns how an entity may be broken down into sub-entities, and the concept of coupling is to specify how these sub-entities may be combined to reconstitute the entity. The taxonomic relationship concerns admissible variants of an entity.

As shown in Figure 1, SES is represented as a labeled tree with attached attributes which satisfies the following axioms: uniformity, strict hierarchy, alternating mode, valid brother, and attached variables (Zeigler 1984). There are three types of nodes in the tree. *Entity* node, like A, represents a real world object, and may have several aspects and/or specializations. There are two types of entity, namely composite entity and atomic entity. Composite entity is defined in terms of other entities (which may be either atomic or composite), while atomic entity can not broken down into sub-entities. An entity may be attached with, and characterized by, *variables*. *Aspect* node, connected by single vertical line from an entity, like *A-dec*, represents one decomposition of the entity. The children of an aspect node are entities, distinct components of the decomposition. An aspect has *coupling* specifications associated with it. *Specialization* node, connected by a double vertical line from an entity, like *B-spec*, defines the taxonomy of the entity. It represents a way in which general entities can be categorized into specialized entities. *Selection rules* may be associated with specialization node, and guide the way in which specialized entities are selected in the pruning process. *Pruning* process extracts a specific system structure from a SES. Such an extracted structure is called *pruned entity structure* (PES), in which every entity has a single aspect, and no specialization. *Selection constraint* concept (Zeigler et al 1991), depicted as dotted arrow in the Figure 1, means that not all specialized entities may be selected independently. Once an entity of specialization is chosen, some entities of other specializations are rejected or selected.

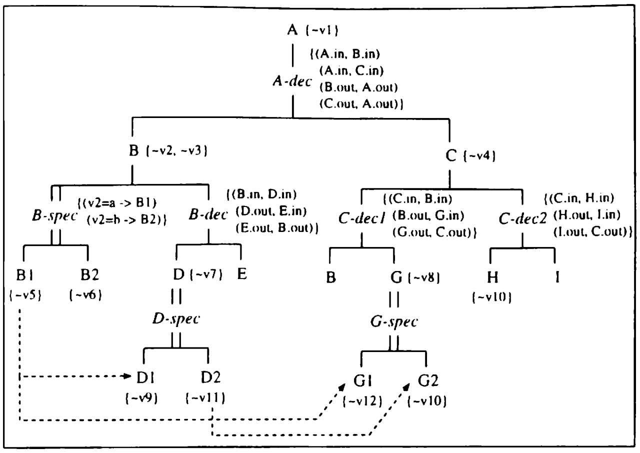


Figure 1: Simple Example of SES

We propose more formalized form of SES. It can be formalized by items, the relations among them, attributes attached to them, and the selection constraints as follows.

SES = <ITEM, REL, ATT, Gsel>

ITEM = E ∪ A ∪ S : *set of items*
E : set of entities,
A : set of aspects,
S : set of specializations;

REL = Asp ∪ Spec : *relations among items*
$\text{Asp} \subset E \times A \times 2^{E}$: aspect relation,
$\text{Spec} \subset E \times S \times 2^{E}$: specialization relation;

ATT = Evar ∪ Acoup ∪ Srule : *attributes of items*
$\text{Evar} : E \rightarrow 2^{V}$: attached variables,
$\text{Acoup} : A \rightarrow 2^{(E.IO \times E.IO)}$: attached couplings,
$\text{Ssel} : S \rightarrow 2^{R}$: attached selection rules;

$\text{Gsel} : (S \times E) \rightarrow 2^{(S \times E)}$: *global selection constraints.*

where V represents a set of variables,
IO represents a set of input/output ports of entities,
R is a set of selection rules in the form of $(cond \rightarrow E)$.

Items, which are depicted as nodes in the treelike representation, are composed of *entities*, *aspects*, and *specializations*. *Relations* among items define how each item is related with each others, and are depicted as edges in the treelike representation. *Aspect relation* relates each entity with sub-entities through an aspect, and *specialization relation* relates each entity with specialized entities through a specialization. Each item may be augmented by attached attributes. *Variables* are attached to, and characterize each entity. *Couplings* are attached to each aspect, and can be described in terms of entities with their input/output ports. *Selection rules* are also attached to each specialization, and help the selection process of the pruning. We also consider global *selection constraints* among specializations and their specialized entities. For example, the entity structure in Figure 1 is defined as follows.

E = {A, B, C, B1, B2, D, E, G, H, I, D1, D2, G1, G2}; A = {A-dec, B-dec, C-decl, C-dec2}; S = {B-spec, D-spec, G-spec}; Asp = {(A, A-dec, {B,C}), (B, B-dec, {D,E}), (C, C-decl, {B,G}), (C, C-dec2, {H,I})}; Spec = {(B, B-spec, {B1,B2}), (D, D-spec, {D1,D2}), (G, G-spec, {G1,G2})}; Evar = {(A, {v1}), (B, {v2,v3}), (C, {v4}), (B1, {v5}), (B2, {v6}), (D, {v7}), (G, {v8}), (H, {v10}), (D1, {v9}), (D2, {v11}), (G1, {v12}), (G1, {v10})}; Acoup = {(A-dec, {(A.in, B.in), (A.in, C.in), (B.out, A.out), (C.out, A.out)}), (B-dec, {(B.in, D.in), (D.out, E.in), (E.out, B.out)}), (C-decl, {(C.in, B.in), (B.out, G.in), (G.out, C.out)}), (C-dec2, {(C.in, H.in), (H.out, I.in), (I.out, C.out)})}; Ssel = {(B-spec, {(v2=a → B1), (v2=b → B2)})}; Gsel = {((B-spec, B1), {(D-spec, D1), (G-spec, G1)}), ((D-spec, D2), {(G-spec, G2)})}.

3 RASES: RELATIONAL ALGEBRAIC FORMALIZATION OF SES

This section first briefly reviews the concepts of relational algebra. Then we propose a relational formalization of SES, called RASES, and a pruning algorithm on the relationally formalized SES by using relational algebra.

3.1 Relational Algebra in Brief

The cartesian product of domains $D_1, \cdots, D_n$, written $D_1 \times \cdots \times D_n$, is a set of n-tuples $< v_1, \cdots, v_n >$ such that v_1 is in D_1, v_2 is in D_2, and so on. *Relation* is any subset of the cartesian product of one or more domains, and each element of the relation is called *tuple*. Relation can be represented as a table where each row is tuple and each column has a distinct name called attribute. Each attribute has an associated domain. A relation R with a set of attributes $A = \{A_1, \cdots, A_n\}$ is denoted by $R[A]$ or $R[A_1 \cdots A_n]$. Let t be a tuple in $R[A]$. Then the part of t corresponding to a set of attributes $X \subseteq A$ is denoted by $t[X]$. We also use the relation name itself, for example $t[R]$, to indicate all attributes of the relation.

There is a family of operations usually associated with relations. They can be coded by using algebraic notations, called *relational algebra*. Fundmental operations in relational algebra are *union*($\cup$), *difference*($-$), *cartesian product*($\times$), *projection*(π), and *selection*(σ). In addition to the five fundmental operations, there are some other useful operations that can be defined in terms of the operations above. They are *intersection*($\cap$), *natural join*($\bowtie$), and *theta join*($\bowtie_\theta$). Details of such operations can be found in (Codd 1970, Codd 1979).

In addition to the above operations, we defined a new operation, called *replacement*(Θ), which will be used intensively for a pruning algorithm, as follows.

Replacement ($\Theta^{A_2 \Leftarrow B_2}_{A_1 = B_1}$) operation:
Let $R[X]$, $S[Y]$, and $R'[X]$ be relations,
and attributes $A_1, A_2 \in X$ and $B_1, B_2 \in Y$.
Then each tuple $r' \in R'$ for $R' = R\ \Theta^{A_2 \Leftarrow B_2}_{A_1 = B_1} S$
is obtained as follows.
For each tuple $r \in R$,
if there exists a tuple $s \in S$ such that $r[A_1] = s[B_1]$
then $r'[X - A_2] = r[X - A_2]$ and $r'[A_2] = s[B_2]$
else $r' = r$.

That is, a replacement operation of $R\ \Theta^{A_2 \Leftarrow B_2}_{A_1 = B_1} S$ means that each value of attribute A_2 is replaced with the value of attribute B_2 if the condition $A_1 = B_1$ is satisfied as shown in the following example.

R:

A	B	C
a	1	x
b	2	y
b	3	y

S:

D	E
u	1
v	2
w	4

→ $R\Theta^{A \Leftarrow D}_{B=E} S$:

A	B	C
u	1	x
v	2	y
b	3	y

The replacement operation can also be coded by using other operations as follows. Here, $\delta_{A_2 \leftarrow B_2}$ means that attribute name B_2 is changed into A_2.

$$R\ \Theta^{A_2 \Leftarrow B_2}_{A_1 = B_1} S = \delta_{A_2 \leftarrow B_2}(\pi_{B_2, X - A_2}(R \bowtie_{A_1 = B_1} S)) \cup (R - \pi_X(R \bowtie_{A_1 = B_1} S))$$

3.2 Relational Formalization of SES

Although SES has been visually represented as a tree-like structure, it can be transformed into other forms which can coherently convey the information it bears (Zhang and Zeigler 1989). In this section, we present a relational formalization of SES, in which the constituents of SES are represented as relations, and the operations on them are defined by relational algebra. The relational formalization of SES can be easily represented as relational tables . Each relational table is defined in terms of relation(table) name, attribute(column) names, and some kinds of constraint on them. *Relational algebraic system entity structure* (RAES) is defined by a 6-tuple.

RASES = <ASP,SPEC,EVAR,ACOUP,SSEL,GSEL>

1) *ASP* [*ent*, *asp*, *subent*] : contains the aspect relation. *ent* is an entity, *asp* is an aspect, and *subent* is a sub-entity of the entity *ent*. This means that the *subent* is a sub-entity of *ent* in the aspect *asp*.

2) *SPEC* [*ent*, *spec*, *specent*] : contains the specialization relation. *ent* is an entity, *spec* is a specialization, and *specent* is a specialized entity of the entity *ent*. This means that *specent* is a specialized entity of the general entity *ent* in the specialization *spec*.

3) *EVAR* [*ent*, *variable*, *value*] : contains the variables attached to the entities. *ent* is an entity to which a *variable* is attached, and *value* is a value of the variable *variable*. This means that *variable*, whose value is *value*, is attached to the entity *ent*.

4) *ACOUP* [*asp*, *ent*1, *port*1, *ent*2, *port*2] : contains the couplings attached to the aspects. *asp* is an aspect, *ent*1 is an entity, *port*1 is a port of the entity *ent*1, *ent*2 is another entity, and *port*2 is a port of the entity *ent*2. This means that *port*1 of the entity *ent*1 is connected with *port*2 of the entity *ent*2 for the aspect *asp*.

5) *SSEL* [*spec*, *cond*, *specent*] : contains the selection rules attached to the specializations. *spec* is a specialization, *cond* is a condition, and *specent* is a specialized entity. This means that if the condition *cond* is satisfied then *specent* is selected for the specialization *spec*.

6) *GSEL* [*spec*1, *specent*1, *spec*2, *specent*2] : contains the global selection constraints. *spec*1 and *spec*2 are specializations, *specent*1 is a specialized entity of *spec*1, and *specent*2 is a specialized entity of *spec*2. This means that if *specent*1 is selected for the specialization *spec*1 then *specent*2 must be selected for the specialization *spec*2.

All of the relations are independent of a specific SES to be represented. Thus, creating a SES for a family of model structures is a matter of entering data into predefined relations rather than a matter of defining new relations. This choice of having all relations predefined has advantages. Since the database structure is known prior, an implementation of the models management framework is easier. Furthermore, since all the SESs built in this framework look alike, they are easier to understand, to combine, and to interchange. Figure 2 shows the relational tables for the example SES depicted in Figure 1.

Several operations and functions on SES may be defined in relational algebra. The relational algebra, in turn, can be easily coded in a standard query language such as the SQL. This helps users to analyze the characteristics of a SES stored in the SES base. To clarify the simplicity of relational operations, only two of them (one for predefined operations and the other for user coded query) are presented here.

ASP :

ent	*asp*	*subent*
A	A-dec	B
A	A-dec	C
B	B-dec	D
B	B-dec	E
C	C-dec1	B
C	C-dec1	G
C	C-dec2	H
C	C-dec2	I

SPEC :

ent	*spec*	*specent*
B	B-spec	B1
B	B-spec	B2
D	D-spec	D1
D	D-spec	D2
G	G-spec	G1
G	G-spec	G2

EVAR :

ent	*variable*
A	v1
B	v2
B	v3
C	v4
B1	v5
B2	v6
D	v7
G	v8
H	v10
D1	v9
D2	v11
G1	v12
G2	v10

ACOUP :

asp	*ent1*	*port1*	*ent2*	*port2*
A-dec	A	in	B	in
A-dec	A	in	C	in
A-dec	B	out	A	out
A-dec	C	out	A	out
B-dec	B	in	D	in
B-dec	D	out	E	in
B-dec	E	out	B	out
C-dec1	C	in	B	in
C-dec1	B	out	G	in
C-dec1	G	out	C	out
C-dec2	C	in	H	in
C-dec2	H	out	I	in
C-dec2	I	out	C	out

SSEL :

spec	*cond*	*specent*
B-spec	v2 = a	B1
B-spec	v2 = b	B2

GSEL :

spec1	*specent1*	*spec2*	*specent2*
B-spec	B1	D-spec	D1
B-spec	B1	G-spec	G1
D-spec	D2	G-spec	G2

Figure 2: Relational Tables Representation of Example SES

1) *spec_of*: The operation $spec\_of(x)$ returns a part of specialization relation ($SPEC$) which has x as the value of *ent* column. It is coded as the following relational algebra.

$$spec\_of(x) = \sigma_{ent=x}(SPEC)$$

For example,

$spec\_of(B) =$

ent	*spec*	*specent*
B	B-spec	B1
B	B-spec	B2

2) *user coded SQL query* : Although users may retrieve much structural information on a SES by using the predefined operations, they can also use the SQL for their own purpose without the help of predefined operations. For example, the list of entities, which have variable 'v7' or 'v10' as their attached variables, can be obtained by the following SQL query.

```
SELECT ent
FROM   EVAR
WHERE  variable = 'v7' OR
       variable = 'v10';
```

3.3 Relational Algebraic Pruning of SES

A SES specifies a family of model structures, in which every entity is organized by aspects and specializations. The pruning process is to extract a pruned entity structure, also called pure entity structure, from an original SES. *Pruned entity structure*

Algorithm: prune(SES)
/* input : SES (ASP, SPEC, EVAR, ACOUP)
output : PES (ASPpruned, EVARpruned, ACOUPpruned) */

step1: prune_select(SES) ;

step2: ASPpruned $\leftarrow$ (ASPsel $\Theta_{subent=ent}^{subent\Leftarrow specent}$ SPECsel) $\Theta_{ent=ent}^{ent\Leftarrow specent}$ SPECsel ;

step3: ACOUPsel $\leftarrow \pi_{ACOUP}$(ACOUP $\bowtie_{asp=asp}$ ASPsel) ;

ACOUPpruned $\leftarrow$ (ACOUPsel $\Theta_{ent1=ent}^{ent1\Leftarrow specent}$ SPECsel) $\Theta_{ent2=ent}^{ent2\Leftarrow specent}$ SPECsel ;

step4: EVARsel $\leftarrow \pi_{EVAR}$(EVAR $\bowtie_{ent=ent \vee ent=subent}$ ASPsel)

$\cup\ \pi_{EVAR}$(EVAR $\bowtie_{ent=ent \vee ent=specent}$ SPECsel) ;

EVARpruned $\leftarrow$ EVARsel $\Theta_{ent=ent}^{ent\Leftarrow specent}$ SPECsel ;

Figure 3: Relational Algebraic Pruning Algorithm

(PES) is a SES in which every entity has either a single aspect or no aspect. In a PES, leaf entities have no aspect, and every non-leaf entity has only one aspect which represents the unique decomposition of that entity. Pruning process is mainly composed of two parts. First, one aspect and/or one specialized entity are selected from the alternatives hanging from each entity. The selection may be guided by the modeling objectives (Zeigler 1984). Next, each selected specialized entity inherits all (sub-structures, variables, couplings) of its general entity.

We propose a relational algebraic pruning algorithm as shown in the Figure 3. Step1 selects only one aspect and/or one specialized entity from its alternatives hang from each nonleaf entity (details are in *prune_select*()). Note that an entity may have several aspects and/or specialized entities. In step2, each specialized entity, which is selected in step1, replaces its general entity from the aspect table. In this way, the specialized entity inherits the sub-structure of its general entity. In step3, coupling specifications of each aspect is appropriately modified by replacing general entities in the couplings with its specialized entities. In step4, each general entity attached with variables is also replaced with its specialized entity. In this way, the specialized entity inherits the variables of its general entity.

In the algorithm, the replace operation (Θ) in step2-4 is used to inherit all information of each general entity into its respective specialized entity. As an example, the variables inheritance operation in step4, $EVARpruned \leftarrow EVARsel\ \Theta_{ent=ent}^{ent\Leftarrow specent} SPECsel$, on the example SES is presented in Figure 4(a). Its operational effect is same as, but its complexity is lower than, the inheritance operation on the tree of Figure 4(b).

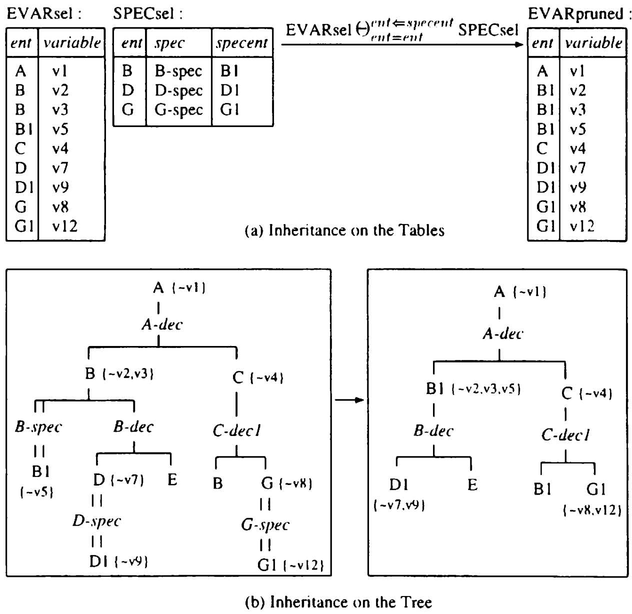


Figure 4: Inheritance of Variables

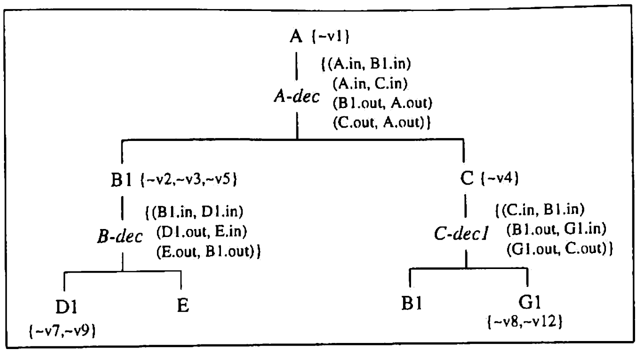
The algorithm in Figure 5 is to select one item from alternatives. Step1 selects one aspect hanging from an entity. Step2 is to select one specialized entity hanging from this entity. If there is only one item to be selected then it is selected automatically. In step3, the selection process proceeds into next entity in a breadth-first traverse. The selection rules and global selection constraints can also be incorporated into this selection process with some additional work.

Figure 6(a) depicts one possible PES pruned from the example SES in Figure 1. Figure 6(b) shows relational tables representation of the PES.

Algorithm: prune_select(SES)
/* input : SES (ASP, SPEC)
output : Selection (ASPsel, SPECsel) */

NEXT $\leftarrow$ {root(SES)} ;
do until NEXT = $\emptyset$
N $\leftarrow$ next(NEXT) ;
step1: As $\leftarrow \pi_{asp}(\sigma_{ent=N}(\text{ASP}))$;
A $\leftarrow$ select(As) ;
ASPa $\leftarrow \sigma_{ent=N \wedge asp=A}(\text{ASP})$;
ASPsel $\leftarrow$ ASPsel $\cup$ ASPa ;
step2: Ss $\leftarrow \pi_{specent}(\sigma_{ent=N}(\text{SPEC}))$;
S $\leftarrow$ select(Ss) ;
SPECs $\leftarrow \sigma_{ent=N \wedge specent=S}(\text{SPEC})$;
SPECsel $\leftarrow$ SPECsel $\cup$ SPECs ;
step3: NEXT $\leftarrow$ NEXT $\cup\ \pi_{subent}(\text{ASPa})$
$\cup\ \pi_{specent}(\text{SPECs})$ − <N> ;

Figure 5: Selection Algorithm



(a) tree representation

ASPpruned :

ent	*asp*	*subent*
A	A-dec	B1
A	A-dec	C
B1	B-dec	D1
B1	B-dec	E
C	C-dec1	B1
C	C-dec1	G1

ACOUPpruned :

asp	*ent1*	*port1*	*ent2*	*port2*
A-dec	A	in	B1	in
A-dec	A	in	C	in
A-dec	B1	out	A	out
A-dec	C	out	A	out
B-dec	B1	in	D1	in
B-dec	D1	out	E	in
B-dec	E	out	B1	out
C-dec1	C	in	B1	in
C-dec1	B1	out	G1	in
C-dec1	G1	out	C	out

EVARpruned :

ent	*variable*
A	v1
B1	v2
B1	v3
B1	v5
C	v4
D1	v7
D1	v9
G1	v8
G1	v12

(b) relational tables representation

Figure 6: PES Pruned from Example SES

4 RASES FRAMEWORK FOR MODELS MANAGEMENT

We are implementing a RASES framework shown in Figure 7, on a commercially available general purpose RDBMS, the INFORMIX. The RDBMS provides rich facilities that can be used to implement applications including the RASES framework. The implementation relies on the ESQL/C, in which the query language SQL is embedded into the host programming language C. The ESQL/C provides all the functionalities of the SQL and C. It allows one to perform computations, by using C language, on the results of SQL queries. The framework may provide a supporting tool to systematically organize a family of model structures of a system in the form of SES, following the principles of RASES. It also provides a guiding tool to extract model structures from the SES by the pruning process. Accordingly, the modelers, who has partial knowledge of the system, can synthesize complete simulation models which meet their modeling objectives. The synthesized models may be simulation codes for a specific simulation environment. For example, the current RASES framework constructs DEVSim++ codes for the DEVSim++ simulation environment. The DEVSim++ (Kim and Park 1992) is a realization of the DEVS formalism (Zeigler 1984) for modeling and simulation in C++.

There are three databases. *SES Base* and *PES Base* contains system entity structures and pruned entity structures, respectively. *Model Base* contains behavioral definitions of components, each of which may be atomic or coupled DEVSim++ simulation model. Users can access the databases through the user-interface provided by the framework. They can also manipulate the databases and exploit the power of RDBMS for their own purpose by using SQL queries. *SES Manager* module provides several facil-

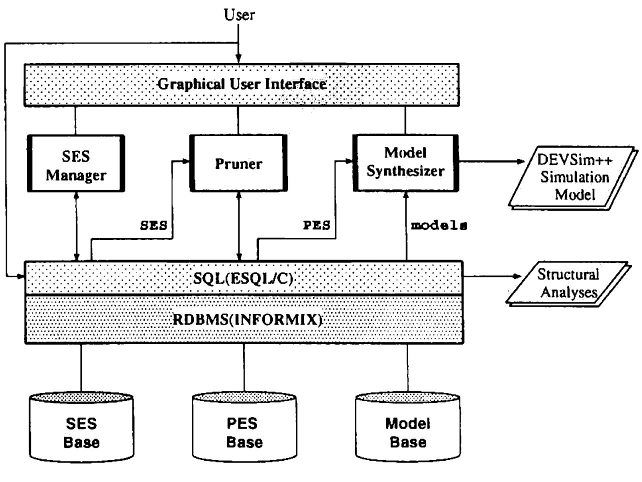


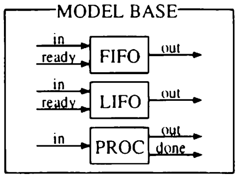
Figure 7: RASES Framework for Models Management

ities such as construction and modification of SESs. *Pruner* module is an implementation of the algebraic pruning algorithms. *Model Synthesizer* module is to synthesize a complete simulation model by combining a PES with behavioral definitions in the model base. Leaf entities of the PES are replaced with DEVSim++ atomic models, and higher level entities are mapped to coupled DEVSim++ models by coupling the lower-level models.

5 AN EXAMPLE : BUFFER-PROCESSOR MODELS

A simple example shows the applicability of the RASES framework to the models management. It assumes that there exist atomic models of specialized buffers FIFO, LIFO, and a processor PROC, in the model base as shown in Figure 8(a). From these models, configuration experts can construct a SES which configures possible model structures. For example, the SQL code of Figure 8(b) creates a SES, called PEL. The RASES framework, but, provides rich facilities which support users to create this SQL code. Figure 8(c) and Figure 8(d) shows the tables representation and tree representation of the SES, respectively. As shown in the SES, the root entity PEL has only one aspect, called *pel-dec*, which decomposes the PEL into a buffer(BUF) and a cascaded processor(PROC). The couplings of the *pel-dec* are composed of the parent entity(PEL) and the sub-entities(BUF, PROC) with their input, output ports. The general buffer(BUF) has two specialized types(FIFO, LIFO) under the specialization *buf-type*.

Once the SES of PEL is built, modelers can prune it into a PES according to their modeling objectives. There are two possible PESs, one of which is a PES with FIFO, and another is a PES with LIFO. Figure 9(a) (Figure 9(b)) shows an example of PES, where the FIFO has been selected as the specialized type of buffer. As shown in the PES, the FIFO replaces all occurrences of its general entity BUF. Next step is to synthesize the PES into a DEVSim++ simulation model. Figure 9(c) and (d) depicts the block diagram and the DEVSim++ code of the coupled model PEL, respectively. It is synthesized by combining the PES with the atomic models FIFO and PROC in the model base. The modelers can conduct appropriate experiments on the model PEL by using the DEVSim++ simulator. The coupled model PEL may itself be saved into the model base as shown in Figure 9(e), and in turn could be used as a component to construct other higher level models.



(a) Model Base

```
CREATE TABLE pel:asp   ( ent      CHAR(20),
                         asp      CHAR(20),
                         subent   CHAR(20) );
CREATE TABLE pel:spec  ( ent      CHAR(20),
                         spec     CHAR(20),
                         specent  CHAR(20) );
CREATE TABLE pel:acoup ( asp      CHAR(20),
                         ent1     CHAR(20),  port1  CHAR(20),
                         ent2     CHAR(20),  port2  CHAR(20) );
INSERT INTO pel:asp   VALUES ("PEL", "pel-dec", "BUF");
INSERT INTO pel:asp   VALUES ("PEL", "pel-dec", "PROC");
INSERT INTO pel:spec  VALUES ("BUF", "buf-type", "FIFO");
INSERT INTO pel:spec  VALUES ("BUF", "buf-type", "LIFO");
INSERT INTO pel:coup  VALUES ("pel-dec", "PEL", "in", "BUF", "in");
INSERT INTO pel:coup  VALUES ("pel-dec", "BUF", "out", "PROC", "in");
INSERT INTO pel:coup  VALUES ("pel-dec", "PROC", "done", "BUF", "ready");
INSERT INTO pel:coup  VALUES ("pel-dec", "PROC", "out", "PEL", "out");
```

(b) SQL Code

ASP :

ent	*asp*	*subent*
PEL	pel-dec	BUF
PEL	pel-dec	PROC

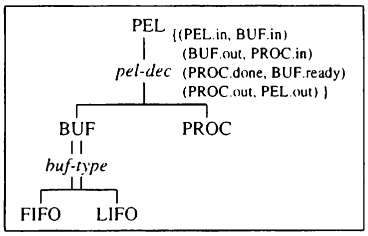
SPEC :

ent	*spec*	*specent*
BUF	buf-type	FIFO
BUF	buf-type	LIFO

ACOUP :

asp	*ent1*	*port1*	*ent2*	*port2*
pel-dec	PEL	in	BUF	in
pel-dec	BUF	out	PROC	in
pel-dec	PROC	done	BUF	ready
pel-dec	PROC	out	PEL	out

(c) SES : Tables Representation



(d) SES : Tree Representation

Figure 8: Construction of SES

6 CONCLUSIONS

This paper proposed a relational algebraic framework, called the RASES, to manage the complexity of the models management problem. By adapting system entity structure (SES) and relational algebra (RA) as a conceptual basis, we have laid a groundwork for an integrated approach to the problem. The framework is intended to support the work of managing large amounts of models and model structures, by providing flexible and rapid access to the database.

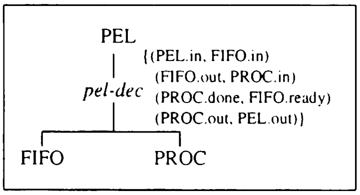
ASPpruned :

ent	*asp*	*subent*
PEL	pel-dec	FIFO
PEL	pel-dec	PROC

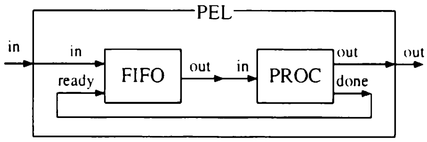
ACOUPpruned :

asp	*ent1*	*port1*	*ent2*	*port2*
pel-dec	PEL	in	FIFO	in
pel-dec	FIFO	out	PROC	in
pel-dec	PROC	done	FIFO	ready
pel-dec	PROC	out	PEL	out

(a) PES : Tables Representation



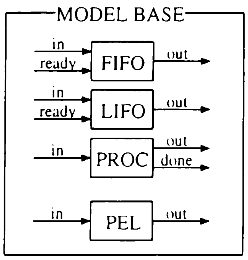
(b) PES : Tree Representation



(c) Block Diagram of Model PEL

```
void make_PEL()
{
  PEL.add_inports ("in");
  PEL.add_outports ("out");
  PEL.add_children (FIFO, PROC);
  PEL.add_coupling (PEL, "in", FIFO, "in");
  PEL.add_coupling (FIFO, "out", PROC, "in");
  PEL.add_coupling (PROC, "done", FIFO, "ready");
  PEL.add_coupling (PROC, "out", PEL, "out");
  PEL.add_priority (FIFO, PROC);
}
```

(d) DEVSIM++ Code for Coupled Model PEL



(e) New Model Base

Figure 9: Pruning and Model Synthesis

Although this paper focused more on the models management, the framework, with further research, can be applied to other domains if the domain knowledge can be represented in the form of SES.

REFERENCES

Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. *Comm. ACM* 13(6):377-387.

Kim, T. G., C. Lee, E. R. Christensen, and B. P. Zeigler. 1990. System Entity Structuring and Model Base Management. *IEEE Trans. Systems, man, and Cybernetics* 20(5):1013-1024.

Kim, T. G. 1990. Hierarchical Scheduling in an Intelligent Environmental Control System. *Journal of Intelligent and Robotic System* 3:183-193.

Kim, T. G., and S. B. Park. 1992. The DEVS Formalism: Hierarchical Modular Systems Specification in C++. In *Proc. of the 1992 European Simulation Multiconference*, 152-156.

Sevinc, S., and B. P. Zeigler. 1988. Entity Structure Based Design Methodology: A LAN protocol Example. *IEEE Trans. Software Engineering* 14(3):604-611.

Zeigler, B. P. 1984. *Multifacetted Modelling and Discrete Event Simulation.* London: Academic Press.

Zeigler, B. P., C. J. Luh, and T. G. Kim. 1991. Model Base Management for Multifacetted Systems. *ACM Transactions on Modeling and Computer Simulation* 1(3):195-218.

Zhang, G., and B. P. Zeigler. 1989. The System Entity Structure: Knowledge Representation for Simulation Modeling and Design. In *Artificial Intelligence, Simulation and Modeling*, eds. L. E. Widman, K. A. Loparo, and N. R. Nielsen, 47-73. New York: Wieley.

AUTHOR BIOGRAPHIES

HYU CHAN PARK is a Ph.D. candidate in the Department of Electrical Engineering at the Korea Advanced Institute of Science and Technology (KAIST), Korea. His research interests are focused on the applications of database and systems modeling.

WAN BOK LEE is a M.S. candidate in the Department of Electrical Engineering at the KAIST. His research interests are modeling and simulation environment.

TAG GON KIM is an Associate Professor in the Department of Electrical Engineering at the KAIST. He received Ph.D. degrees in electrical enrineering from the University of Arizona. His primary research interests include advanced simulation methodology, systems modeling, and object-oriented simulation environment.