To be presented at: Third Int. SNePS Workshop
July 28-29, 94 Buffalo, NY

SNePS as a Database Management System

Stuart C. Shapiro
Department of Computer Science
and Center for Cognitive Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
U.S.A.
shapiro@cs.buffalo.edu

July 29, 1994

SNePS can be used as a network version of a relational database system in which every element of the relational database is represented by a base node, each row of each relation is represented by a molecular node, and each column label (attribute) is represented by an arc label. Whenever a row r of a relation R has an element e_i in column c_i, the molecular node representing r has an arc labeled R to the special node **relation**, and an arc labelled c_i pointing to the base node representing e_i. Table 1 shows two relations from the Supplier-Part-Project database of Date[1], p. 114. Figure 1

Table 1: From Date's Supplier-Part-Project Database

SUPPLIER

S#	SNAME	STATUS	CITY
s1	Smith	20	London
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	London
s5	Adams	30	Athens

PROJECT

J#	JNAME	CITY
j1	sorter	Paris
j2	punch	Rome
j3	reader	Athens
j4	console	Athens
j5	collator	London
j6	terminal	Oslo
j7	tape	London

shows a fragment of the SNePS network version of this database.

1 SNePS as a Relational Database

The three basic operations on relational databases are **select, project,** and **join.** The next three subsections show how these operations may be expressed in SNePSUL.

[1] C. J. Date, *An Introduction to Database Systems 3rd Edition* (Reading, MA: Addison-Wesley) 1981.

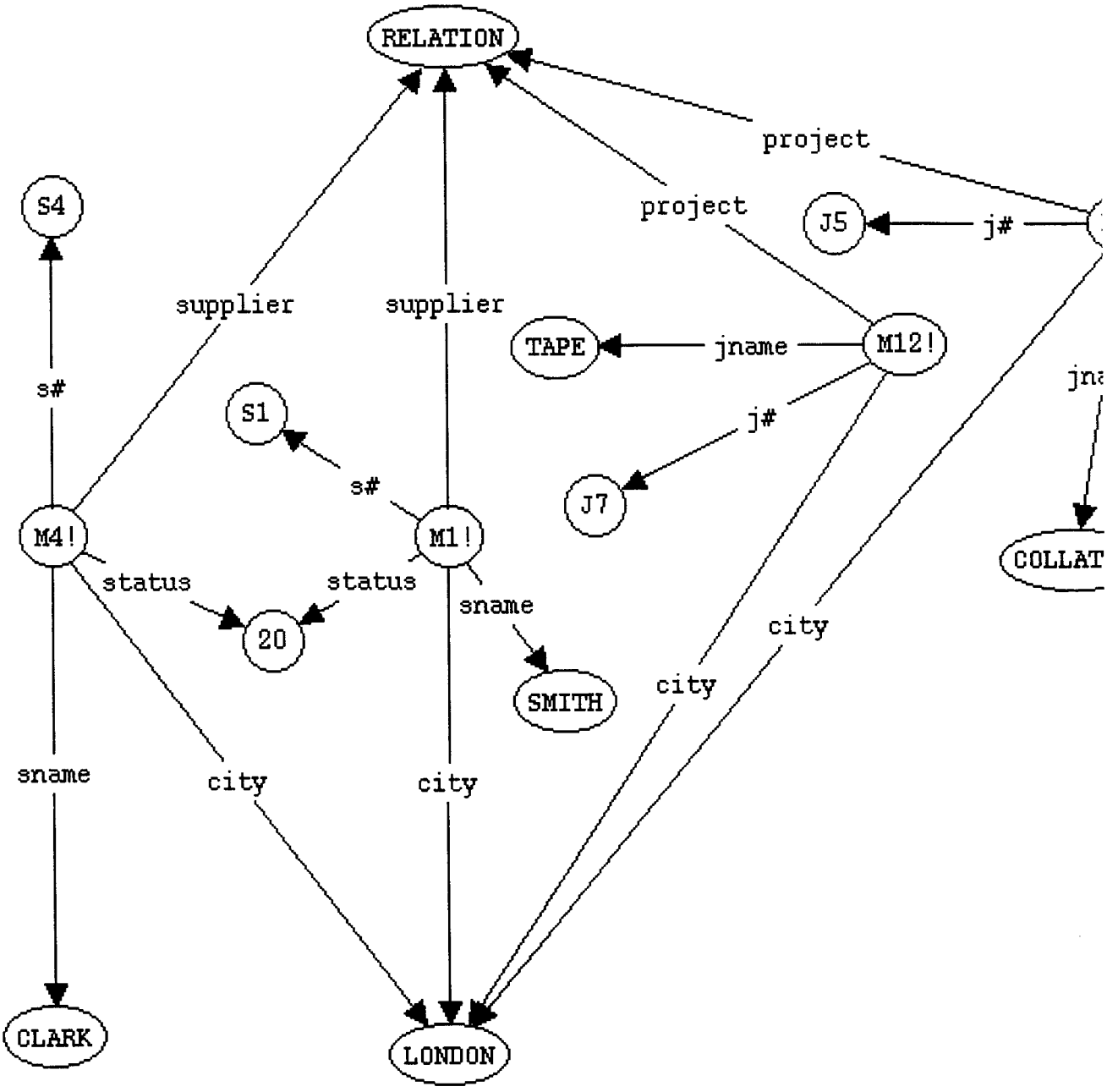

Figure 1: Fragment of SNePS network for the Supplier-Part-Project Database

1.1 Project

Project is a database operation that, given one relation, produces another that has all the rows of the first, but only specific columns. (Actually some of the rows might collapse if the only distinguishing elements were in columns that were eliminated.) The SNePSUL **dbproject** function has been designed for this purpose. For example, to show the STATUS and CITY of all suppliers, one can do

```
* (dbproject (find supplier relation) status city)
((STATUS (20) CITY (LONDON)) (STATUS (10) CITY (PARIS))
 (STATUS (30) CITY (PARIS)) (STATUS (30) CITY (ATHENS)))
 CPU time : 0.07
```

The **dbproject** function forms and returns a *virtual* relation, which is represented as a SNePS data type called a *set of flat cable sets.* Compare the following two ways of getting complete details of the SUPPLIER relation. The first uses the SNePSUL **describe** function to print the details of the nodes that make up the relation:

```
* (describe (find supplier relation))
(M1! (CITY LONDON) (S# S1) (SNAME SMITH) (STATUS 20) (SUPPLIER RELATION))
(M2! (CITY PARIS) (S# S2) (SNAME JONES) (STATUS 10) (SUPPLIER RELATION))
(M3! (CITY PARIS) (S# S3) (SNAME BLAKE) (STATUS 30) (SUPPLIER RELATION))
(M4! (CITY LONDON) (S# S4) (SNAME CLARK) (STATUS 20) (SUPPLIER RELATION))
(M5! (CITY ATHENS) (S# S5) (SNAME ADAMS) (STATUS 30) (SUPPLIER RELATION))
(M1! M2! M3! M4! M5!)
 CPU time : 0.15
```

The second uses **dbproject** to display a virtual relation with the same information:

```
* (dbproject (find supplier relation) supplier s\# sname status city)
((SUPPLIER (RELATION) S# (S1) SNAME (SMITH) STATUS (20) CITY (LONDON))
 (SUPPLIER (RELATION) S# (S2) SNAME (JONES) STATUS (10) CITY (PARIS))
 (SUPPLIER (RELATION) S# (S3) SNAME (BLAKE) STATUS (30) CITY (PARIS))
 (SUPPLIER (RELATION) S# (S4) SNAME (CLARK) STATUS (20) CITY (LONDON))
 (SUPPLIER (RELATION) S# (S5) SNAME (ADAMS) STATUS (30) CITY (ATHENS)))
 CPU time : 0.12
```

Virtual relations are created without building any new SNePS network structure. To make these relations permanent, use the SNePSUL **dbAssertVirtual** function. For example, to create a CITYSTATUS relation that is a projection of the SUPPLIER relation down the CITY and STATUS attributes, we would first define CITYSTATUS as a new SNePS relation:

```
* (define citystatus)
(CITYSTATUS)
 CPU time : 0.03
```

Then we would do

```
* (describe (dbAssertVirtual (dbproject (find supplier relation) city status)
                             (citystatus relation)))
(M13! (CITY LONDON) (CITYSTATUS RELATION) (STATUS 20))
(M14! (CITY PARIS) (CITYSTATUS RELATION) (STATUS 10))
(M15! (CITY PARIS) (CITYSTATUS RELATION) (STATUS 30))
(M16! (CITY ATHENS) (CITYSTATUS RELATION) (STATUS 30))
(M13! M14! M15! M16!)
 CPU time : 0.28
```

1.2 Select

Select is an operation that is given a relation and specific values for some of its attributes, and yields the rows of the relations in which those attributes take on those values. A selection from relation R_1 in which attribute a_{1i} takes on value v_{1i} is expressed in SNePSUL as

$$\texttt{(find } R_1 \texttt{ relation } a_{11}\ v_{11}\ \ldots a_{1n}\ v_{1n}\texttt{)}.$$

For example, to select rows of the SUPPLIER relation where the CITY is Paris or Athens and the STATUS is 30, we could do:

```
* (describe (find supplier relation city (paris athens) status 30))
(M3! (CITY PARIS) (S# S3) (SNAME BLAKE) (STATUS 30) (SUPPLIER RELATION))
(M5! (CITY ATHENS) (S# S5) (SNAME ADAMS) (STATUS 30) (SUPPLIER RELATION))
(M3! M5!)
 CPU time : 0.08
```

If we want a new permanent relation, say **supplier2**, to be this selection from the SUPPLIER relation, we could do:

```
* (define supplier2)
(SUPPLIER2)
 CPU time : 0.03

* (describe
   (dbAssertVirtual
    (dbproject (find supplier relation city (paris athens) status 30)
               s\# sname status city)
    (supplier2 relation)))
(M17! (CITY PARIS) (S# S3) (SNAME BLAKE) (STATUS 30) (SUPPLIER2 RELATION))
(M18! (CITY ATHENS) (S# S5) (SNAME ADAMS) (STATUS 30) (SUPPLIER2 RELATION))
(M17! M18!)
 CPU time : 0.23
```

1.3 Join

Join is a database operation that, given two relations, R_1 and R_2, with attributes $a_{11}, \ldots, a_{1n}$ and $a_{21}, \ldots, a_{2m}$, respectively, and an atttibute $a = a_{1i} = a_{2j}$ produces a relation with attributes $a_{11}, \ldots, a_{1n}, a_{21}, \ldots, a_{2j-1}, a_{2j+1}, \ldots, a_{2m}$, and every row, $e_{11}, \ldots, e_{1n}, e_{21}, \ldots, e_{2j-1}, e_{2j+1}, \ldots, e_{2m}$ where $e_{11}, \ldots, e_{1n}$ was a row of R_1, and $e_{21}, \ldots, e_{2j-1}, e_{1i}, e_{2j+1}, \ldots, e_{2m}$ was a row of R_2. For example, Table 2 shows the join of the relations in Table 1 on the attribute CITY.

This join may be created and displayed by the SNePSUL **dbjoin** command, which, like **dbproject** creates a virtual relation.

```
* (dbjoin city
          (find supplier relation) (s\# sname status city)
          (find project relation) (j\# jname))
(S# (S1) SNAME (SMITH) STATUS (20) CITY (LONDON) J# (J7) JNAME (TAPE))
(S# (S1) SNAME (SMITH) STATUS (20) CITY (LONDON) J# (J5) JNAME (COLLATOR))
(S# (S2) SNAME (JONES) STATUS (10) CITY (PARIS) J# (J1) JNAME (SORTER))
(S# (S3) SNAME (BLAKE) STATUS (30) CITY (PARIS) J# (J1) JNAME (SORTER))
(S# (S4) SNAME (CLARK) STATUS (20) CITY (LONDON) J# (J7) JNAME (TAPE))
(S# (S4) SNAME (CLARK) STATUS (20) CITY (LONDON) J# (J5) JNAME (COLLATOR))
(S# (S5) SNAME (ADAMS) STATUS (30) CITY (ATHENS) J# (J4) JNAME (CONSOLE))
```

Table 2: The join of SUPPLIER and PROJECT on CITY

S#	SNAME	STATUS	CITY	J#	JNAME
s1	Smith	20	London	j5	collator
s1	Smith	20	London	j7	tape
s2	Jones	10	Paris	j1	sorter
s3	Blake	30	Paris	j1	sorter
s4	Clark	20	London	j5	collator
s4	Clark	20	London	j7	tape
s5	Adams	30	Athens	j3	reader
s5	Adams	30	Athens	j4	console

```
(S# (S5) SNAME (ADAMS) STATUS (30) CITY (ATHENS) J# (J3) JNAME (READER))
 CPU time : 0.35
```

Again, to make the virtual relation permanent, **dbAssertVirtual** is used:

```
* (define supplierproject)
(SUPPLIERPROJECT)
 CPU time : 0.03

* (describe
   (dbAssertVirtual
    (dbjoin city
            (find supplier relation) (s\# sname status city)
            (find project relation) (j\# jname))
    (supplierproject relation)))
(M19! (CITY LONDON) (J# J7) (JNAME TAPE) (S# S1) (SNAME SMITH) (STATUS 20)
      (SUPPLIERPROJECT RELATION))
(M20! (CITY LONDON) (J# J5) (JNAME COLLATOR) (S# S1) (SNAME SMITH) (STATUS 20)
      (SUPPLIERPROJECT RELATION))
(M21! (CITY PARIS) (J# J1) (JNAME SORTER) (S# S2) (SNAME JONES) (STATUS 10)
      (SUPPLIERPROJECT RELATION))
(M22! (CITY PARIS) (J# J1) (JNAME SORTER) (S# S3) (SNAME BLAKE) (STATUS 30)
      (SUPPLIERPROJECT RELATION))
(M23! (CITY LONDON) (J# J7) (JNAME TAPE) (S# S4) (SNAME CLARK) (STATUS 20)
      (SUPPLIERPROJECT RELATION))
(M24! (CITY LONDON) (J# J5) (JNAME COLLATOR) (S# S4) (SNAME CLARK) (STATUS 20)
      (SUPPLIERPROJECT RELATION))
(M25! (CITY ATHENS) (J# J4) (JNAME CONSOLE) (S# S5) (SNAME ADAMS) (STATUS 30)
      (SUPPLIERPROJECT RELATION))
(M26! (CITY ATHENS) (J# J3) (JNAME READER) (S# S5) (SNAME ADAMS) (STATUS 30)
      (SUPPLIERPROJECT RELATION))
(M19! M20! M21! M22! M23! M24! M25! M26!)
 CPU time : 0.92
```

2 SNePS as a Network Database

Although SNePS can be treated as a relational database, as shown in the previous section, it is more naturally a network database. For example, to find the names of suppliers with the same status

as suppliers in the same city as the sorter project using relational database techniques, one would join the SUPPLIER and PROJECT relations on CITY, join the result with SUPPLIER again on STATUS, select rows where PROJECT is sorter, and project the result on the SNAME attribute.

However, in SNePSUL, one could just do

```
* (find (sname- status status- city city- jname) sorter)
(ADAMS BLAKE JONES)
 CPU time : 0.02
```

Additional examples of these techniques may be found in the SNePS DBMS demonstration.

3 Database Functions

Functions specifically supplied for treating SNePS as a Database Management System are documented in this section. Additional ones may be created using the functions documented in Chapter 4 of the SNePS 2.1 User's Manual. Note also **innet** and **outnet**, documented in Section 2.4 of the Manual, for saving the database across runs.

(**dbAssertVirtual** *virtualexp* [([*relation nodeset*]*)])
Evaluates *virtualexp*, which must return a virtual relation (set of flat cable sets), appends the list ([*relation nodeset*]*) to each flat cable set, asserts each resulting flat cable set as a SNePS molecular node, and returns the set of asserted nodes.

(**dbcount** *nodesetexp*)
Evaluates the SNePSUL nodeset expression, *nodesetexp*, and returns a node whose identifier looks like the number which is the number of nodes in the resulting set.

(**dbjoin** *relation nodesetexp1 relations1 nodesetexp2 relations2*)
A virtual relation (a set of flat cable sets) is created and returned. The virtual relation is formed by taking the nodes returned by the SNePSUL node set expression, *nodesetexp1* and the nodes returned by the SNePSUL node set expression, *nodesetexp2,* joining these two relations on the attribute *relation,* and then projecting the result down the *relations1* attributes from the first nodeset and the *relations2* attributes from the second nodeset. Note that **relations1** and **relations2** is each a list of relations.

(**dbmax** *nodesetexp*)
Evaluates the SNePSUL nodeset expression, *nodesetexp*, which must evaluate to a set of nodes all of whose identifiers look like numbers, and returns the node whose identifier looks like the biggest of the numbers.

(**dbmin** *nodesetexp*)
Evaluates the SNePSUL nodeset expression, *nodesetexp*, which must evaluate to a set of nodes all of whose identifiers look like numbers, and returns the node whose identifier looks like the smallest of the numbers.

(**dbproject** *nodesetexp relations*)
A virtual relation (a set of flat cable sets) is created and returned. The virtual relation is formed by taking the nodes returned by the SNePSUL node set expression, *nodesetexp,* and projecting down the SNePSUL relations included in the sequence, *relations.*

(`dbtot` *nodesetexp*)
Evaluates the SNePSUL nodeset expression, *nodesetexp*, which must evaluate to a set of nodes all of whose identifiers look like numbers, and returns a node whose identifier looks like the sum of the numbers.