APPLYING DEDUCTIVE DATABASE TECHNOLOGY TO NETWORK MANAGEMENT

Nalin Sharda, Refyul Fatri and Mohammad Abid
Victoria University of Technology
nalin@ matilda.vut.edu.au

ABSTRACT

Network Management is essential for successful operation of any communications network. Due to the complexity of modern networks, their management requires application of artificial intelligence based techniques. Two essential aspects of any Network Management system are, a large volume of data, and rules applied to this data. Deductive database systems cater for both. In this paper we examine the suitability of deductive database systems for Network Management application. Fundamentals of Network Management as well as deductive database systems are given. We describe a prototype Network Management system built to test the suitability of deductive database systems for Network Management.

1 INTRODUCTION

The function of any Network Management system is to monitor and control the behaviour of a network system to achieve better utilization of the system.

A Network Management system requires a database to store information about the network and its operation. This database is called the Management Information Base (MIB). The MIB is a database with information about the devices to be managed and management functions. The main objective of our project was to design and implement a prototype MIB using a deductive database system. In this project an MIB which contains information about Fault and Configuration Management has been developed using the Aditi deductive database system. We have based our prototype MIB on a subset of the computer network in the Department of Computer and Mathematical Sciences, Victoria University of Technology, called the CAMSNET.

1.1 ORGANIZATION OF THIS PAPER

The remainder of this paper is organized as follows. Section 2 looks at the history of logic programming and deductive databases. Examples are given to show how and why a deductive database is superior to a relational database. It also presents the structure of the Aditi deductive database system used in this project.

Section 3 gives a brief overview of Network Management. Section 4 describes the Management Information Base and the various management protocols currently used in the industry.

Section 5 covers methodology used for developing the MIB for the CAMSNET, and describes some basic terminology used in this database. This section also describes briefly the base relations and the derived relations defined for the CAMSNET-MIB. Finally, a few sample queries performed on this database are given.

Section 6 presents conclusions of the project and suggested directions for future work.

2 THE ADITI DEDUCTIVE DATABASE SYSTEM

2.1 INTRODUCTION

Deductive databases offer the prospect of a more intelligent way of handling data, by applying deductive rules to capture the complexity of information. Deductive databases are more powerful than relational systems, in that rules can be stored along with raw facts. This makes the resulting system more sophisticated than a relational system. Writing an application that has deductive power is simpler with a deductive database system than with a relational database system. This section presents the idea of logic programming, and some basic requirements for building a deductive database. Structure of the Aditi deductive database system and its various components are described in some detail.

2.2 DEDUCTIVE DATABASES

Logic programming began in the early 1970s based on theorem proving and artificial intelligence. The principal idea behind logic programming is the use of mathematical logic as a programming language. This was introduced by Kowalski (1947) and was made practical by Colmerauer through the implementation of the first logic programming language, **PROLOG** (PROgramming in LOGic). Most of the logic programming is based on the Horn-clause form [5, 6].

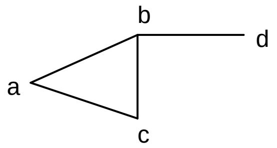
The field of deductive databases grew out of research, in the late seventies and early eighties, on the use of logic to manipulate large amounts of data. The deductive database concept is an extension of the work by Green (1969) for his question-answer system. Deductive databases use logic programming technology to generalize relational databases by providing support for recursive views, and non-atomic data, which makes database programming easier for many applications.

The expressive power and functionality of relational database query languages is limited as compared with those of logic programming languages. Another limitation of the relational model is that data is stored as an explicit table. This may be adequate for many applications, but there could be cases in which this can lead to complex queries for seemingly simple problems. One way to overcome these kinds of shortcomings is to use a deductive database system. The method of storing data differs between a relational database and a deductive database. A relational database stores relations explicitly by listing all facts in the relation. A deductive database, on the other hand, allows for a relation to be defined in terms of another relation. A relational database consists of facts only, whereas a deductive database may contain rules as well. This extra feature means that deductive databases are more expressive than relational ones [2, 3, 4].

Following are the advantages of using predicate logic as a database language:

- Virtually all database concepts of interest to users - queries, views and integrity constraints - can be viewed in logic terms, which allows the database to present a single unified interface to its users.
- Deductive databases provide more expressive power than relational databases. They can represent non-first-normal-form relations also.
- In a logic programming language such as Prolog, it is a simple matter to embed a programming interface into a deductive database, which simplifies the process of writing application programs. Prolog suits many problem domains that need the extra expressive power of deductive databases.

The following example gives some idea how a deductive database uses logic as a database language[2]. In this example the connectivity of the following graph is being stored in a deductive database.



This graph can be represented by the following four facts.

edge(a, b).
edge(a, c).
edge(b, c).
edge(b, d).

These four facts say that there are edges from **a** to **b**, from **a** to **c**, from **b** to **c** and from **b** to **d**.

An equivalent set of **SQL** statements would be:

```
create table edge (source char(20) not null, sink
  char(20) not null;
insert into edge values ('a', 'b');
insert into edge values ('a', 'c');
insert into edge values ('b', 'c');
insert into edge values ('b', 'd');
```

Deductive databases, unlike relational database systems, allow tuples (facts) to contain structured data.

Consider the following example:

course(202, john, [9200710, 9200711]).

The above example states that unit 202 is taught by john, and is attended by two students with student numbers 9200710 and 9200711. In **SQL**, the same data requires at least two relations, one for linking units to lecturers and one for linking units to students.

A query in a deductive database on the relation **edge** may look like the following:

?- edge(a, Y).
Y = {b, c}.

The above query asks for the names of all nodes **Y** that are at the ends of edges from **a**; the answer is the set containing **b** and **c**. In SQL the equivalent query is:

select sink from edge where source = 'a';
sink
'b'
'c'

In a deductive database, a new relation can be defined as a view on the other relations:

twoedges(X, Y) :- edge(X, Z), edge(Z, Y).

The **twoedges** relation is called a derivation rule, because in this we have derived a new fact from old facts. This derivation rule states that a node **X** is separated from a node **Y** by two edges if there is an edge from **X** to a third node **Z**, and an edge from **Z** to **Y**. The symbol ':-', means "is implied by".

An equivalent view in SQL would be:

```
create view twoedges(e1.source, e2.sink) as
select e1.source, e2.sink
from edge e1, edge e2
where e1.sink = e2.source;
```

Using logic, one can define a relation **path** if there is a sequence of one or more edges in the relation **edge**.

path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).

The above derivation rules state that "if there is an edge between **X** and **Y** then there is a path between **X** and **Y**; and that if there is a node **Z** such that there is an edge between **X** and **Z**, and a path between **Z** and **Y**, then there is a path between **X** and **Y**". Any change to the edge relation will be instantly reflected in the path relation as well.

Rules in which a relation name appears on both left and right sides of the ":-" symbol are said to be recursive rules.

Users can make queries on the relation **path** as if it were a normal relational database relation. For example, to find out which nodes are reachable from node **a,** and from which nodes one can reach node **c**, the queries and their responses would be:

Query:- **?- path(a, Y).** and **?- path(X, c).**
Response:- **Y = {b, c, d}.** and **X = {a, b}.**

These queries cannot be expressed directly in SQL because they involve recursion. If a user wants to ask this type of query in a relational database system, then the database designer must write a program in a procedural programming language such as C using embedded SQL.

In a deductive database a query such as **?- path(a, Y)** is answered by starting with the facts and continually deriving new facts from the existing ones, until no more facts can be derived. This means applying the first rule of **path** once and the second rule a variable (possibly large) number of times. Because this method goes from the facts towards the query, it is called bottom-up computation, in contrast to the top-down computation strategies used by the implementation of most logic programming languages.

2.3 PROPERTIES REQUIRED IN A DEDUCTIVE DATABASE SYSTEM

In a deductive database system the user interface should allow the user to specify the facts and rules about the system based on predicate logic. Therefore, a deductive database should meet the following requirements:

- The language used in a deductive database system should be based on logic. This means that the application as well as the database queries should be written in a language based on logic. It should also be possible to embed queries in an application program without changing the application's syntax. Writing application code and queries to the database should be seamlessly integrated. Prolog with extensions for type and mode declarations, quantification, and aggregate operations meets these requirements.
- It sould allow information to be shared by several users. Information is to persist beyond the lifetime of any given application program execution. This can be done through the support of transactions, recovery, integrity constraints and the ability to store rules as well as facts in the database.

The first requirement distinguishes deductive databases from relational database systems, whereas the second requirement separate deductive databases from logic programming systems[2].

2.4 INTRODUCTION TO ADITI

Aditi is a deductive database system developed at the University of Melbourne, and the MIB we have implemented is based on a Beta version of the same.

Aditi is a multi-user system with its own security mechanisms operating in a Unix environment[1].

The power of deductive databases lies in derived relations - ie. relations that can infer information at run-time. In Aditi, users define derived relations by writing programs in Aditi-Prolog, a pure (declarative) variant of Prolog augmented with mode declarations and flags to control the Aditi-Prolog compiler.

Rules in Aditi are required to be range restricted; this means the base relations (facts in Prolog terminology) must not contain variables, and that the rules defining the derived relations must be such that the variables which occur in the head of a rule must also appear in the body of the rule. Base relations are created from files of tuples (facts) and then linked into the database system. In this respect, Aditi's facilities are very similar to those of a relational database system.

Aditi rules are essentially Prolog rules with some extra declarations. These declarations are for the information of the compiler: what arguments will be bound when the predicate is called, which transformation to use on the rule, and which optimisation methods to use. For example, the transitive closure **path(X, Y)** of a base relation **edge(X, Y)**, defined in section 2.2, is given as follows:

```
?- mode(edge(f,f)).
?- mode(path(b,f)).
?- mode(path(f,b)).
?- flag(path, 2 magic).
path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).
```

The first four lines are declarations to the compiler. The first line declares that the predicate **edge** has two arguments and that it expects to be called with both arguments as free variables (variables that have no constraints imposed on their values).

The second and third lines declare that the predicate **path** has two arguments, and that it has two modes, ie. it can be called in either of two ways: it can be called with the first argument bound to a ground term and the second argument free (eg. **?- path(a, Y)**) or with the second argument bound and first argument free (eg. **?-path(X, b)**). The fourth line requests magic evaluation for the **path** predicate with arity 2. The last two lines are the rules defining this predicate.

Note that this rule is range restricted, ie. all variables which appear in the head also appear in the body of the rule.

2.5 THE STRUCTURE OF ADITI

The structure of Aditi is based on the client / server model found in many commercial relational database systems. Figure 1 shows the structure of Aditi.

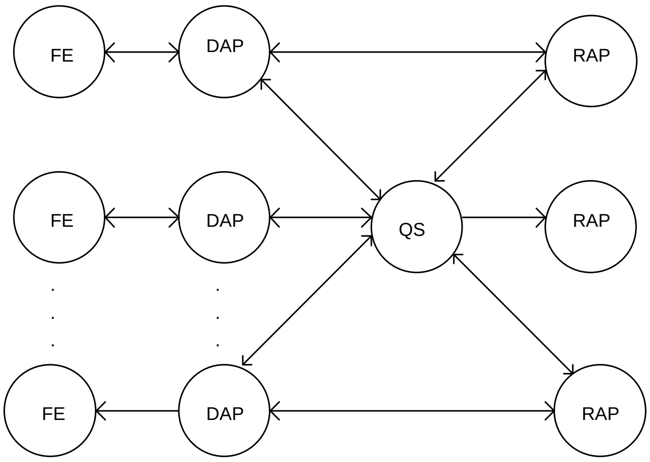


Figure 1 The Structure of Aditi.

Users can only interact with a front-end process (FE), which is regarded as a client of the system. This client communicates with a back-end process (server). The back-end-process performs the set of database operations, such as joining, merging and subtracting relations, on behalf of the clients. There are two kinds of systems: some systems have one server per client, while others have one server supporting multiple clients. Aditi is a hybrid of these two schemes. Some of the server processes are dedicated to individual clients while others are shared by all clients.

The dedicated server process is called the Database Access Process (DAP). DAPs have the responsibility for security enforcement for its clients. The DAP is also responsible for updating permanent relations and shutting down the system. It also performs all tasks connected with query evaluation except the execution of relational algebra operations. The tasks which are not performed by DAPs, are performed by a pool of server processes called Relational Algebra Process (RAPs). The RAPs are regarded as the workhorses of Aditi. The Query Server (QS) is the central management process of Aditi, it performs load management for Aditi. The query shell has both line-oriented and graphical user interfaces. The Query Server is also called the master process.

The relational algebra operations are sent to the QS for execution while a DAP evaluates a query. The QS passes the task on to a free RAP, otherwise the task is queued until a RAP becomes available. As a RAP completes a task it sends the result to the requesting DAP. The free RAP then informs the QS that it is available for another task.

3 NETWORK MANAGEMENT

3.1 INTRODUCTION

Network Management is becoming increasingly important in the communications industry. As the pace of data transaction within and between enterprises accelerates, the role of this network activity expands, especially in the business environment. The evolving need for Network Management has been increasingly appreciated by business planners, investors and top management of the world's most influential corporations and governments. Without it, the assets of an advanced (communications) network resource remain unmanageable and uncontrollable. Thus an efficient Network Management system is essential to avoid costly disruption to communications.

An important question is: why do organizations invest significant amounts of time and money building complex data networks that need to be maintained? Because data networks help to perform jobs efficiently, and give easier access to information.

Once a data network is in place, it must be managed efficiently to maximize its potential. This is done through a process called Network Management. The primary objective of every Network Management System (NMS) is quite specific: to keep the network working. In terms of actual performance, the Network Management system should monitor the network, assist in identifying and diagnosing problems, and then help to correct them.

3.2 FUNCTIONS AND BENEFITS OF NETWORK MANAGEMENT

Network management is an important component of any network system. The main functions of Network Management are as follows:

- The network manager provides facilities such as system configuration and reconfiguration. There are two types of network managers : passive network managers and active network managers. A passive network manager only "observes" the network. An active network manager actively participates in controlling the network.
- The network manager looks after the security of the network and also of the Network Management system.
- It is the responsibility of the network manager to ensure that the required network is available at all times, and it meets all the necessary requirements of the network user. This could include real-time service requirements. Some of the benefits of doing Network Management properly are:
- Improved response time.
- Improved availability.
- Less equipment.
- Lower service charges.
- Lower risks.
- Accurate accounting.

3.3 THE ARCHITECTURE OF A NETWORK MANAGEMENT SYSTEM

The architecture of a Network Management system is a complex entity. There are no specific rules that can be applied to the architecture of a Network Management system. However, the following guidelines should be considered to arrive at a complete solution [7].

- The system must provide a graphical interface showing the logical connection between the various levels of hierarchy.
- The system should provide a database system which contains all the information related to the desired application. The database information can be used for management functions such as Fault Management, Configuration Management, Security Management, Performance Management and Accounting Management.
- The system must have the provision for collecting information about all relevant network devices.
- The system must be expandable and easily customized. The system must also facilitate operations performed by the human Network Administrator.
- The system must provide a procedure for tracking outstanding problems. In particular, this will be useful for a complex and large network.

Network management architectures can be any of the following three types.

- Centralized architecture.
- Distributed architecture.
- Combination of centralized and distributed architecture.

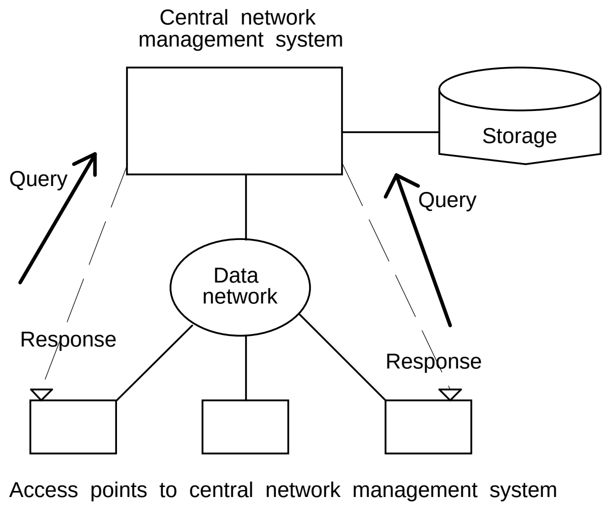


Figure 2 Centralized Network Management Architecture.

In a centralized architecture, shown in figure 2, a large system runs the majority of the desired applications. All the information related to applications are stored on the central machine. This architecture is ideal for an organization having centralized administration at one location.

In a distributed architecture, shown in figure 3, several peer network systems run simultaneously on the data network. In this set-up, each network management subsystem manages a specific part of the system. This architecture is ideal for an organization that is distributed over different geographical locations or has its administration distributed with many equal levels of management.

In a third possible architecture, the centralized and distributed methods are combined to form a hierarchical system. The central system forms the root of the hierarchy, and the distributed system comprises the leaves in the hierarchy. The central system has access to all parts of the network and it allocates tasks to each distributed system.

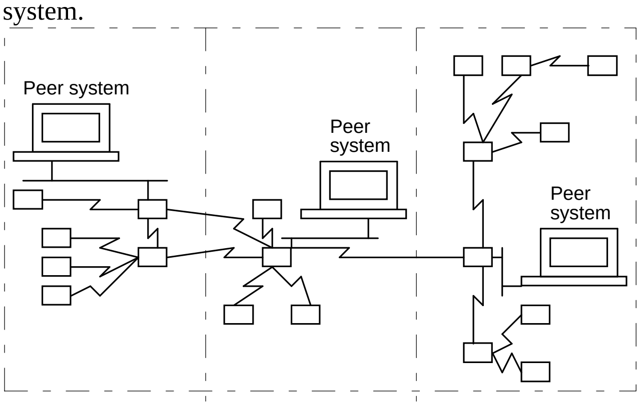


Figure 3 A Distributed Network Management Architecture.

3.4 NETWORK MANAGEMENT PROTOCOLS

All architectures require the use of certain protocols to ensure reliable data transmission. Protocols are a set of rules for transmitting and receiving data between devices. It is unlikely that any data network would be built entirely from products provided by a single company. Therefore the need arises for standard protocols. The International Standards Organization (ISO), develops, suggests and names standards for network protocols.

Two main standard Network Management protocols are:

- SNMP (Simple Network Management Protocol)
- CMIS/CMIP (Common Management Information Services/Common Management Information Protocol)

Both SNMP and CMIS/CMIP provide a means of obtaining information from or giving instructions to network devices. Both these Network Management protocols conform to the ISO Reference Model for Open Systems Interconnections (OSI). SNMP is used mostly for TCP/IP based data networks. These standard protocols for Network Management provide access to all network devices even if these are made by different manufacturers.

Queries for network devices may include the following:

1. The name of the device.
2. The version of software in the device.
3. The number of interfaces in the device.
4. The number of packets per second on an interface of the device.

Setable parameters for network devices may include the following:

1. The name of the device.
2. The address of a network interface.
3. The operational status of a network interface.
4. The operational status of the device.

Standardized Network Management protocols carry additional benefits in that the data sent and returned by a device are of a uniform appearance.

4 THE MANAGEMENT INFORMATION BASE

4.1 INTRODUCTION

This section covers the fundamental aspects of a Management Information Base (MIB). Further, the relationship between the OSI management protocol and an MIB is also briefly discussed.

4.2 INTRODUCTION TO MANAGEMENT INFORMATION BASE

A network is a collection of devices, and provides a facility for transferring data from one device to another. A typical network may include devices such as Central Processing Units, monitors, printers, modems etc. A management system is required to make effective use of the resources on the network. Such a management system requires a database to store information relating to network devices.

The database used to store information about network components is called a Management Information Base (MIB). The MIB developed in this project deals with the Fault and Configuration Management functions only. In addition, the MIB for this project has been developed using a deductive database.

4.3 NETWORK MANAGEMENT PROTOCOLS

Various standard Network Management protocols have been proposed by different standards organizations. As stated in section 3, the two main standard Network Management protocols are:

- CMIS/CMIP for OSI standard based networks
- SNMP for TCP/IP based networks

The MIB developed here is based on the OSI model. Figure 4 illustrates the OSI management model. In this model, the MIB which contains all the management information in the system has a component attached to each layer of the OSI reference model. N-layer management functions allow changing of layer parameters and environmental conditions at layer N.

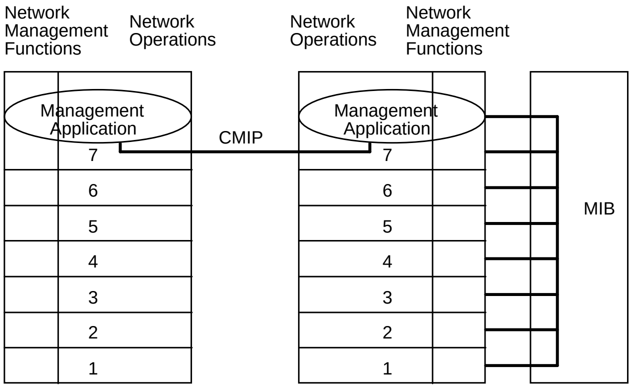


Figure 4 OSI Network Management Model.

Although, the Network Management functions are distributed over the seven layers, communication for Network Management between the two ends occurs only at the application layer (layer 7). This communication uses the CMIP protocol. The CMIP communication protocol is used to provide the CMIS services.

5 IMPLEMENTATION

5.1 INTRODUCTION

The administrator of a network in an organization needs to maintain and store a variety of information which will enable him / her to run the management operations smoothly. This storage can be readily achieved by using an information system. As explained in the first section, the purpose of this project is to design and implement a database for Network Management using the Aditi deductive databse system. This section presents the problem statement and the methodology used for designing the MIB database system. The terminology used in the Aditi deductive database system and some results obtained in this project are also given.

5.2 PROBLEM STATEMENT

Developing a Network Management system is essentially a problem solving process. The computer is particularly suited to solve problems in which there is a large amount of data, held in a database, that the user wishes to interrogate. As described in section 2, Prolog with database capability added and with the power to specify rules, is particularly suitable for such problems.

Unlike traditional programming languages, Prolog has the advantage of representing relations (data) either as facts or rules. Prolog has yet another advantage in the flexible way it enables queries to be formulated.

A database is to be created for the Network Administrator, so that if any problems occur on the network, the network manager can use this database to obtain the following information:

- relevant details of the equipment on the network,
- equipment in certain rooms (students or staff),
- performance of equipment, and
- faults in a particular device.

The development of a Network Management System begins with a description of the network entities. Once the system is defined, a functional specification is developed that lists the type of queries a user wishes to be answered by the system.

5.3 DATABASE SYSTEM DEVELOPMENT METHODOLOGY

The initial methodology considered for designing the MIB (Management Information Base) was NIAM (Nijssen's Information Analysis Methodology). NIAM methodology is usually used to design relational database systems that require normalization. Because of the limitation of NIAM (eg. normal form), we chose the ERC+ methodology. ERC+ stands for Entity-Relationship for complex objects [12]. The ERC+ method uses an object-oriented approach. A complex object is a collection of information components, such that each of these components, in turn, may be represented as a collection of information. ERC+ specifically allows for iterative description of an object, up to an arbitrary number of levels. The resulting structure is an attribute tree, with the object as the root. Moreover, any node in the tree may carry a unique attribute value, or a multiset (bag) of attribute values.

Following are the ERC+ features described in [12]:

1. The structure of an entity type consists of a set of one or more attributes.
2. Relationship types may have attributes as well.
3. Relationship types may connect any number of participating entity types; they are said to be cyclic if the same entity type participates more than once in the relationship type.
4. A role name is associated with each participation of an entity type into a relationship type. It is characterized by its minimum and maximum cardinalities, specifying whether it is a 0-1, 0-N, 1-1 or 1-N link from the entity type to the relationship type.
5. Attributes may be:
   - *Mandatory or optional.* An instance of an optional attribute may be empty (no value); for a mandatory attribute a value must be defined in each instance of the attribute.
   - *Monovalued or multivalued.* An instance of a multivalued attribute may include several (possibly duplicate) values, while an instance of a monovalued attribute is made up of a single value.
   - *Atomic or complex.* If atomic, the attribute is nondecomposable; its values are atomic. If complex, the attribute is made up of a set of other attributes, which are said to be the components of that attribute. Component attributes may be atomic or complex. This nesting can proceed to any number of levels.
6. Entity types and relationship types may have zero, one, or more sets of attributes serving as identifiers. If no identifier is known, the respective population may include duplicates (different occurrences with the same value). In particular, two or more relationships may connect the same entities and have the same values for their attributes.
7. Two generalizations are supported - the "is-a", and the "may-be-a" generalizations. The former corresponds to the well-known generalization concept; the latter has similar semantics but does not require an inclusion dependency between the subtype and the type. No automatic inheritance is implicitly built into the querying mechanism, but an explicit operator provides for the desired inheritance effects.

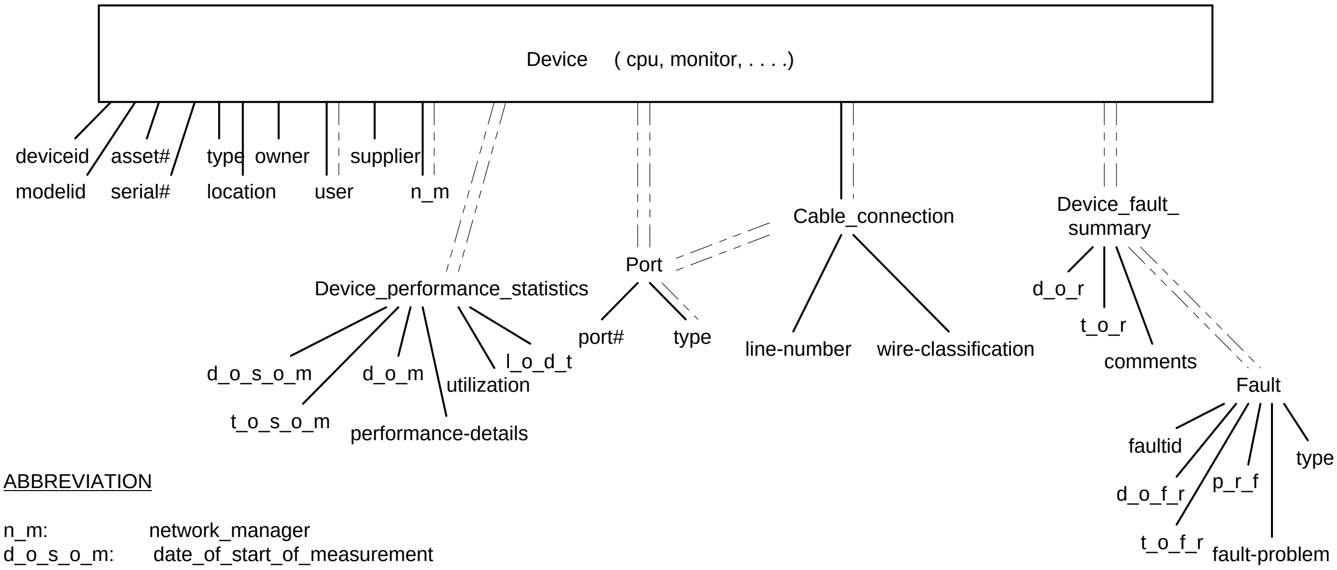
An ERG diagram for the MIB is shown in figure 5. Table 1 shows the four different types of links used for representing the different types of relations in figure 5.

The entity type **device** is mandatorily described by a deviceid, a modelid, an asset#, a serial#, a type, a location, an owner, a user, a supplier, and a network-manager (n_m). There may be one or more users and network-managers. The relations **device-performance-statistics, port, cable-connection, device-fault-summary** and **fault** are composite relations; ie. each one of these consists of a set of component attributes.

Table 1 Links described in the ERC+ model.

mandatory monovalued:	1:1
mandatory multivalued:	1:N
optional monovalued:	0:1
optional multivalued:	0:N

Relations and their attributes used in this project are described in section 5.5.



ABBREVIATION

n_m: network_manager
d_o_s_o_m: date_of_start_of_measurement
t_o_s_o_m: time_of_start_of_measurement
d_o_m: duration_of_measurement
l_o_d_t: length_of_down_time
d_o_r: date_of_restoration
t_o_r: time_of_restoration
d_o_f_r: date_of_fault_reported
t_o_f_r: time_of_fault_reported
p_r_f: person_reporting_fault

Figure 5 An ERC+ Diagram for Describing Devices on a Computer Network.

Table 2 Relation Names and the Corresponding Attributes.

Relation	Attributes
device	deviceid, modelid, asset#, serial#, type, location, owner, user, supplier, n_m
device_performance_ statistics	deviceid, d_o_s_o_m, t_o_s_o_m, d_o_m, performance_details, utilization, l_o_d_t
port	deviceid, port#, type
cable_connection	deviceid_a, port#_a, deviceid_b, port#_b, line_number, wire_classification
fault	faultid, d_o_f_r, t_o_f_r, p_r_f, fault_problem, type
device_fault_summary	deviceid, faultid, d_o_r, t_o_r, comments

Table 3 The Device Relation

deviceid	modelid	asset#	serial#	type	location	owner	user	supplier	n_m
cpu1,	dx486,	a_9201,	tp_9211,	cpu,	lab_1,	cams,	students,	total_peripherals,	raj
cpu2,	dx486,	a_9202,	tp_9212,	cpu,	lab_1,	cams,	students,	total_peripherals,	raj
...									
cpu12,	sx486,	a_9212,	tp_9222,	cpu,	lab_2,	cams,	students,	total_peripherals,	raj

Table 4 The Cable_Connection Relation

deviceid_a	port#_a	deviceid_b	port#_b	line_number	wire_classification
cpu1,	slota,	cpu2,	slota,	a1,	coax
cpu2,	slota,	cpu3,	slota,	a2,	coax
...					
cpu8,	slota,	cpu9,	slota,	a8,	coax

Tables 5 a to g Rules Used in the CAMSNET-MIB

Table 5a Rule for Defining a Fast Machine

fast_machine(Device, Model, Type) :- device(Device, Model, Asset, Serial, Type, Loc, Own, User, Sup, Name), Model = dx486, Type = cpu.	*This relation finds fast machines with two arguments

Table 5b Rules for Defining a Device List

device_list(Device) :- device(Device, _, _, _, _, _, _, _, _, _).	*Relation gives device names with one argument
device_list(Device, Loc) :- device(Device, _, _, _, _, Loc, _, _, _, _).	*Gives device names with two arguments
device_list(Device, Model, Type, Loc) :- device(Device, Model, _, _, Type, Loc, _,_,_,_).	* " " four arguments

Table 5c Rule for Defining IBM Equipment

ibm_equipment(Device) :- device(Device, _, _, _, _, _, _, _, Sup, _), Sup = ibm.	*Devices supplied by ibm

Table 5d Rules for Defining a Path

path(X, Y) :- cable_connection(X, _, Y, _, _, _).	
path(X, Y) :- cable_connection(X, _, Z, _, _, _), path(Z, Y).	*Recursive rule for finding path

Table 5e Rule for Defining a Terminal

terminal(Device, Loc) :- device(Device, _, _, _, Type, Loc, _, _, _, _), Type = cpu.	*Relation gives terminal name *connected to both segments with their location

Table 5f Rules for Defining Next Terminals

next(X, Y) :- cable_connection(X, _, Y, _, _, _).	*Relation gives terminal in either direction
next_terminals(X, Y, Z) :- next(X, Y), next(Y, Z).	*Relation gives terminal in both directions

Table 5g Rules for Defining Bus 1 and Bus 2

bus1(Device) :- device(Device, Model, _, _, _, _, _, _, _, _), Model = dx486, port(Device, _, Type), Type = ethernet.	*Relation gives all terminals connected to segment one
bus1(Device, Loc) :- device(Device, Model, _, _, _, Loc, _, _, _, _), Model = dx486, port(Device, _, Type), Type = ethernet.	*Relation gives terminals connected to segment 1 with respect to their location
bus2(Device) :- device(Device, Model, _, _, _, _, _, _, _, _), Model = sx486, port(Device, _, Type), Type = ethernet.	*Relation gives all terminals connected to segment two
bus2(Device, Loc) :- device(Device, Model, _, _, _, Loc, _, _, _, _), Model = sx486, port(Device, _, Type), Type = ethernet.	*Relation gives terminals connected to segment two with respect to their location

5.4 ADITI TERMINOLOGY

A program written in a logic programming languages consists of: variables, predicate symbols and the logical connectives for conjunction, negation and implication. To distinguish between the various components of a logic program the following conversions are used:

- Variables are represented by identifiers that start with an upper case letter.
- Predicate symbols are represented by identifiers that start with a lower case letters.
- Predicate symbols have an associated arity that specifies the number of their arguments.
- Numbers (integer and floating point) are called constant symbols.
- Comments are prefixed by the "%" or "*" symbol on each comment line.
- Conjunction (AND) is represented by a comma “,”, negation is represented by the word “not”, and implication ("x :- y" ie. if y is true then x will be true) is represented by the symbol “:-”. The underscore symbol “_” means don’t care.

To build a deductive database, the Aditi system contains commands for creating a database, creating a new relation, adding new tuples, and deleting tuples.

5.5 BASE RELATIONS IN THE MIB

In this section we describe a sample database which stores network component information. In addition, this database contains information about Fault and Configuration Management functions.

The six base relations developed for this database are:

- **device**
- **device_performance_statistics**
- **port**
- **cable_connection**
- **fault**
- **device_fault_summary**

The attributes used for these relations are listed in Table 2.

The tuples in the **device** relation are given in Table 3.

The **cable_connection** relation shown in Table 4 contains both device and port information. The **device_fault_summary** relation contains device and fault information.

Figure 6 shows the architecture of the CAMSNET network used for this project. This architecture shows that there are three labs; lab_1 and lab_2 are for students and lab_5 is for staff use only. There are six terminals in lab_1, four terminals in lab_2, and two terminals in lab_5. Terminals are connected using point-to-point connections. There are two segments as shown in Figure 6. Segment one connects terminal 1 to 2, 2 to 3 and so on to terminal 7. Similarly, segment 2 connects terminal 1 to 8, 8 to 9 and so on to terminal 12.

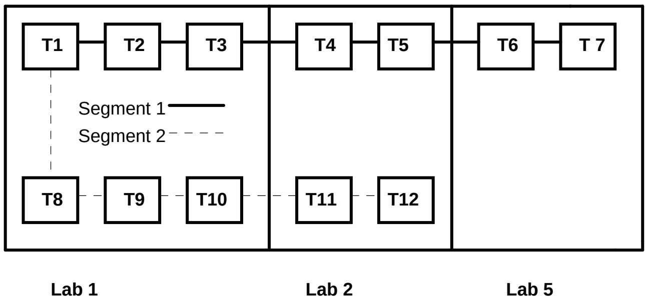


Figure 6 Terminal locations.

5.6 DERIVED RELATIONS

The power of deductive databases lies in derived relations - ie. relations that can infer information at run-time. In Aditi, users define derived relations by writing programs in Aditi-Prolog.

The rules developed for this database are given in Table 5.

5.7 QUERYING THE DATABASE

The Aditi query shell has two slightly different user interfaces. One is part of xaditi, the graphical front-end to Aditi. The other is for use from dumb terminals and from workstation windows running terminal emulation programs.

- "<-" is the query prompt in Aditi.
- All queries must end with a dot ".".

The simplest kind of query is an open query - the query you ask in base relations. In Aditi, derived relations may be queried in the same way as base relations. A few examples of querys for the CAMSNET-MIB and their responses are given in this section.

The following querys asks for the name of all devices.

```
1 <- device_list(Device).
Answer Set for Device:
(1)   cpu1
(2)   cpu2
(3)   cpu3
(4)   cpu4
 .     .
```

Note that the answer relation is of arity 1- a name beginning with a capital letter (eg. Device) represented as a variable. Periods "." and "." below the fourth answer mean that there are more items in the list, which have not been listed here.

If the query contains no variables, then the result is a zero arity relation. If the answer to such a query is "true", then the answer relation will have cardinality one: it will contain the special tuple "<true>". Otherwise the answer relation will have a cardinality of zero.

In our project database, there is a device called cpu1; therefore the following query gives `<true>` as the answer set.

```
2 <- device_list(cpu1).
Answer Set:
      <true>
```

Since there is no device called `cpu30` in our database the following query gives `No Answers` as the answer set.

```
3 <- device_list(cpu30).
Answer Set:
      No Answers
```

The next query asks for the names of all devices which are located in lab_1.

```
4 <- device_list(Device, lab_1).
Answer Set for Device:
(1)   cpu1
(2)   cpu2
(3)   cpu3
 .     .
```

The next query asks for the names of all Devices, Model#, and Locations, whose type of equipment is cpu.

```
5 <- device_list(Device, Model, cpu, Loc).
Answer Set for Device, Model, Loc:
(1)   cpu1,  dx486,  lab_1
(2)   cpu4,  dx486,  lab_2
(3)   cpu11, sx486,  lab_2
 .     .      .       .
```

The next query asks for the device and model numbers of fast machines. In 1994 the dx486 based machines were defined as the fast machines, as given in table 5a.

```
6 <- fast_machine(Device, Model).
Answer Set for Device, Model:
(1)   cpu1,  dx486
(2)   cpu3,  dx486
(3)   cpu4,  dx486
(4)   cpu5,  dx486
(5)   cpu2,  dx486
(6)   cpu6,  dx486
(7)   cpu7,  dx486
```

The following query asks for the name of equipment supplied by IBM.

```
7 <- ibm_equipment(Device).
Answer Set for Device:
(1)   cpu4
(2)   cpu5
(3)   cpu6
 .     .
```

Queries may also involve aggregate operations. The next query collects the value of type (from the **port** relation) in a list for ports which are connected to each terminal.

```
8 <- aggregate(Set = solutions (Type), [Device],
port(Device, Port, Type)).
Answer Set for Device, Set:
(1)  cpu1,   [ethernet, serial, parallel]
(2)  cpu2,   [ethernet, serial, parallel]
(3)  cpu10,[ethernet, serial, parallel]
(4)  cpu11,[ethernet, serial, parallel]
(5)  cpu12,[ethernet, serial, parallel]
 .     .       .       .
```

Similarly the "count" aggregation function computes the length of the list. The next query counts the number of devices which are located in lab_1.

```
9 <- aggregate(Count = count, [Type],
device(Device, Model, Asset#, Serial#, Type,lab_1,
Own, User, Sup, Name)).

Answer  Type,    Count:
(1)     printer,  1
(2)     keyboard, 3
(3)     modem,    3
(4)     monitor,  3
(5)     mouse,    3
(6)     cpu,      6
```

Besides solutions and count, the min, max, and sum aggregate operations are also available in Aditi. The following query finds the nodes which are reachable from cpu1.

```
10 <- path(cpu1, X).
Answer Set for X:
(1)   cpu10
(2)   cpu11
(3)   cpu12
(4)   cpu2
(5)   cpu3
(6)   cpu4
(7)   cpu5
(8)   cpu6
(9)   cpu7
(10)  cpu8
(11)  cpu9
```

We can also can ask which from which nodes one can reach the node called cpu4.

```
11 <- path(X, cpu4).
Answer Set for X:
(1)   cpu1
(2)   cpu2
(3)   cpu3
.     .
```

The next two queries ask the name of the terminal on either the left or right side. If the variable appears on the left side, it finds the terminal on the left side. Similarly if variable appears on the right side, it finds terminal on the right side.

```
12 <- next(cpu2, X).
Answer Set for X:
(1)   cpu3

13 <- next(X, cpu3).
Answer Set for X:
(1)   cpu2
```

Given one terminal we can also find the left and right terminals simultaneously.

```
14 <- next_terminals(X, cpu2, Y).
Answer Set for X, Y:
(1)   cpu1,  cpu3
```

As bus topology is widely used for LAN (Local Area Network) from the cable_connection relation, we define derived relations called bus1 and bus2, which give all the terminals connected to bus1 (segment one) or bus2 (segment two) as shown in figure 6. The following two queries find terminals connected to segment one, and two with their locations, respectively .

```
15 <- bus1(Device).
Answer Set for Device:
(1)   cpu1
(2)   cpu2
(3)   cpu3
(4)   cpu4
(5)   cpu5
(6)   cpu6
(7)   cpu7

16 <- bus1(Device, Loc).
Answer Set for Device, Loc:
(1)   cpu1,  lab_1
(2)   cpu2,  lab-1
(3)   cpu3,  lab_1
(4)   cpu4,  lab_2
(5)   cpu5,  lab_2
(6)   cpu6,  lab_5
(7)   cpu7,  lab_5
```

In the same way we can find terminals which are connected to segment two.

The next query asks for the terminals which are located in lab_1.

```
17 <- terminal(Device, lab_1).
Answer Set for Device:
(1)   cpu1
(2)   cpu2
(3)   cpu3
(4)   cpu8
(5)   cpu9
(6)   cpu10
```

6 CONCLUSION AND FUTURE DIRECTIONS

The aim of this project was to design and implement a deductive database for Network Management. In this paper, the fundamental aspects of Network Management and deductive database system have been described. We have used a Beta version of the Aditi deductive database system developed at The University of the Melbourne. We developed a sample database to see how the Aditi deductive database can be applied to Network Management. This database stores information about network components such as the CPUs, monitors, printers, etc. The results obtained from this database highlight the expressive power of the Aditi deductive database system.

The Aditi deductive database system is suitable for developing a Network Management system because of two reasons, firstly, it provides more expressive power than relational databases, and secondly it can describe recursive relationships between entities. The use of logic in deductive databases allows an elegant way for expressing facts, rules and queries. In addition, it is also possible to add or remove rules without changing the overall structure of the program.

Despite the expressive power of the Aditi deductive database system, there are a few drawbacks as well. One of the major limitations of the Aditi deductive database system is that it does not allow tuples to be modified directly. To modify a tuple, the tuple must be deleted and a new one inserted. This is a limitation of the Aditi system and does not apply to all deductive database system.

Different methodologies such as NIAM, ERC+, were considered for the development of this database. The ERC+ methodology was chosen to design this database, as ERC+ uses an object-oriented approach.

The overall conclusion of this project is that despite some drawbacks, a deductive database can be useful in building Network Management systems.

6.1 DIRECTIONS FOR FUTURE WORK

The database developed in this project is a static database system and data can only be entered manually. This database is not connected to a real-time data capture facility, which is essential for some Nework Management functions. A very useful extension of this work would be to develop a real-time interface for the Aditi system.

7 ACKNOWLEDGMENT

The authors would like to thank Prof. K. Ramamohanarao and his collaborators at The University of Melbourne for access to the Beta version of the Aditi deductive database system.

REFERENCES

[1] Harland, J. , Kemp, D. B. , Leask, T. S., Ramamohanrao, K. , Shepherd, J. A.,.Somogyi, Z. , Stuckey, P. J., and Vaghani, J. , “Aditi users’ guide”, Deductive Database Group, The University of Melbourne, 1992.

[2] Vaghani J., , Ramamohanarao, K. , Kemp, D. B., Somogyi, Z., Stuckey, P. J., Leask, T. S., and Harland, J., “An introduction to the Aditi deductive database system”, Deductive Database Group, The University of Melbourne, 1992.

[3] Harland, J. , Kemp, D. B. , Leask, T. S., Ramamohanrao, K. , Shepherd, J. A.,.Somogyi, Z. , Stuckey, P. J., and Vaghani, J., “Aditi-Prolog language manual”, Deductive Database Group, The University of Melbourne, 1992.

[4] Harland, J. and Ramamohanarao, K., “Experiences with a flights database”, Deductive Database Group, The University of Melbourne, 1992.

[5] Nussbaum, M., "Building a Deductive Database", Department of Computer Science, P. Universidad Cato′lica, Santiago, Chile, 1992, pp 11-47.

[6] Liu, M. and Cleary, J., "Deductive Databases - Where to Now?", Research Report No. 90/377/1, Department of Computer Science, University of Calgary, January 1990.

[7] Leinwand, A. and Fang, K., “Network Management A Practical Perspective”, Addison-Wesley Publishing Company, 1993.

[8] Beyda, W. J., "Basic Data Communications", Prentice-Hall International, Inc. 1989.

[9] Furman, E. L.,"Network Management for ATCS Communications Systems", Proceedings of the IEEE/ASME Joint Railroad Conference. St. Louis, Mo, USA, 1991, pp 71-76.

[10] Miyauchi, N., Nakakawaji T. and Tadanori, K.,“An Implementation of Management Information Base”, Proceedings of the First International Workshop on Interoperability in Multidatabase Systems, IEEE Computer Society Press, USA, 1991, pp 318-321.

[11] Lin Y. D., and Gerla, M., “Induction and Deduction for Autonomous Networks”. IEEE Journal on Selected Areas in Communications, Vol 11, December 1993, pp 1415-1425.

[12] Spaccapietra, S., and Parent, C., “A step forward in solving structural conflicts”. IEEE Transactions on Knowledge and Data Engineering, Vol 6, April 1994, pp 258-274.