INTERNATIONAL JOURNAL OF NATURAL AND APPLIED SCIENCES (IJNAS), VOL. 2, NO. 1(2007); P. 107 – 115.

Deductive enterprise model (DEM): a better approach to enterprise modeling

E. E. Williams[*1], P. C. Bassey[2] and B. O. Akinkunmi[3]

ABSTRACT

This work is centered on the deductive enterprise model (DEM) as a better approach to enterprise modeling and integration of any information system that will be able to deduce answers to questions that one would normally assume can be answered if one has a commonsense understanding of the enterprise. Review of other models is given and the generic enterprise model (GEM) is viewed as a subset of the DEM. Setbacks around these models are also discussed. Concepts about the educational (academic) system are defined and expressed using the First Order Logic (FOL) as representational language that can represent the deductive capabilities in the enterprise of concern. Deductive rules about these concepts were also expressed as axioms using the first order logic and they add up to the knowledge base as the inference mechanism when further implemented using PROLOG programming language. Instances from sample ontology of the academic system were used to further demonstrate the dynamic nature of the deductive enterprise model. Deductions made from these rules become updates to the knowledge base as deductions drawn become new facts and are added to the database of facts.

INTRODUCTION

The existence of the correspondence problem resulting from the legacy created independently by various systems that support enterprise function has long been in existence (Gruninger and Fox, 1996). Enterprise models have been used in the design and operation of enterprises, and it has so been in such a way that very few organizations of significant sizes can operate without enterprise models. It is also noticed that most information systems in use within an enterprise incorporate a model of some aspect of its structure, operations and/or knowledge. Enterprise model have different names, irrespective of the fact that they represent the same concept. This hinders communication among functions except they are further translated. There is inconsistency during interpretation and use of the knowledge. Above all, the cost of designing, building and maintaining independent models for an organization will be high (Uschold et al, 1998).

Some of the systems that support most enterprise functions were independently created and sometimes these functions did not share the same representations. Lack of commonsense deductive capability forced users to spend significant resources on programming every new report or function that is required (Stadder, 1997). Currently, most enterprise processes need to communicate directly with each other thereby allowing the defined concepts to be shared amongst the various units of the enterprise (Gruber, 1991). A common language is needed to interpret these functions across the various units that may need them. Enterprise modeling sets in to handle this problem through enterprise integration (Gruninger and Fox, 1996), (Yu and Mylpoulos, 1997).

The independence of most of these models does not allow knowledge to be shared. These drawbacks/ problems faced by business-process engineering brought about the introduction of a Generic Enterprise Model (GEM) which is a library of generally defined classes of objects that can be employed in defining a specific enterprise. GEM is composed of the following:

(a) A set of object classes structured as a taxonomy

(b) A set of well defined relations/meaning of an object class linking the object to other objects

(c) A set of attributes and semantics for each object class.

GEM has some benefits that make it outstanding. They include:

(i)The use of pre-defined object library which prevents the engineers from starting from the scratch to create an enterprise model, rather they quickly move on to model instantiation.

(ii) Provision of path for growth for enterprise modelers to follow, thereby preventing an omission that will be noticed at a later period. This is possible because other experienced modelers have traced the path and kept for others to follow.

(iii) Provision of shared conceptualization which enable other parts of the organization to understand what is presented in the enterprise model.

(iv) The reduction in time and cost of modeling an enterprise is also applicable.

[*] Corresponding author. Email: edemwilliam@yahoo.com

Manuscript received by the Editor July 26, 2006; revised manuscript accepted September 21, 2006.

[1]Department of Mathematics/Statistics & Computer Science, University of Calabar, PMB 1115, Calabar, Nigeria
[2]Department of Mathematics/Statistics & Computer Science, University of Uyo, PMB 1017, Uyo, Nigeria
[3]Department of Computer Science ,University of Ibadan, Ibadan, Nigeria.

Deductive Enterprise Model (DEM) as a Derivative of GEM

The instantiated GEM becomes useful based on the functions it can support, i.e. the categories of queries the GEM can give answers to. There is need for an additional processing which determines the answers to be provided. This is known as inference capability which assumes heritance as a deductive mechanism.

Although other approaches to developing knowledge-based systems exist, the rule-based reasoning technique is used in representing our domain knowledge. Rules are expressed in the form of an IF <antecedent> THEN <consequent>, where the consequent is a form of action or a conclusion that is deduced from the existing fact and again added to knowledge base. A rule can have more than one antecedent which are usually combined by a AND or OR operator. Also, several conclusions can be drawn from one rule, in whose case the antecedent simply compares an object with a possible value. The object in question is simply a variable representing some physical objects or states in the real world. A rule-based system consists of three (3) components namely, knowledge base of rules, database of facts and inference engine which uses the knowledge base of rules and database of facts in deducting other facts as part of the system.

We adopt the forward chain strategy to employ a deductive approach since our field is considering starting from small number of facts and very few rules, using them to deduce or come up with a suitable course of action or conclusion. This technique is data-driven, meaning that, reasoning will start from a set of data and ends up at the goal (conclusion).

The conclusion is reached and added to the fact database or recommendations made when all the antecedent of one of the rules is matched by the fact in the database, then the rule is triggered, and then fired.According to Fox (1992), expert systems provide deep-level processing. By deep level, we mean that a significant amount of knowledge or search, that is, deductions, has to be performed to provide a response to a query. To answer a query regarding the cause of a machine malfunction, the expert system might have to reason about the structure and behavior of the machine. It must have a detailed model of the domain, and it can be unique to the specific enterprise. Such systems tend to be costly to build and maintain and are narrow in scope. Commonsense queries require that the information system be able to deduce answers to questions that one would normally assume can be answered if one has a commonsense understanding of the enterprise. Such an understanding often represents knowledge about the enterprise acquired over a relatively short period of time, for example, three to nine months, and does not denote knowledge of an expert nature. That is, the knowledge should be broad and not deep and must support a tractable subclass of queries. For example, knowledge of an organization's structure, roles, goals, and resources would enable the deduction of what resources a person might allocate based on his/her role in the organization. It could be argued that the majority of queries posed to a database are in this third category: common sense. If GEMs were designed to support commonsense queries, a significant portion of the management information system (MIS) backlog could be done away with. Commonsense query processing assumes a third level of processing that we refer to as *shallow-level processing.* By shallow level, we mean retrieval that requires a small number of deductions to answer the query. For an enterprise model to support commonsense query processing, it must provide a set of rules of deduction, that is, axioms. For the works-for example, we would require an axiom stating that works-for is transitive: *x* works-for *y* AND *y* works-for *z* IMPLIES *x* works-for *z*.

Fox and Gruninger (1998) view query language as a means of interfacing with the enterprise model. They are of three types: factual, expert and common-sense. The common-sense queries require that the information system be able to deduce answers to questions that one would normally assume can be answered if one has a common-sense understanding of the enterprise. Again, for an enterprise model to support commonsense query processing, the following sets of deduction rules must be provided, that is, the axioms defining the meaning of the relations and attributes in the object library. With the axioms, the model would be able to make deductions. Such models that include axioms and a deduction engine is known as a deductive enterprise model (GEM). In other words, a GEM with deductive capability is called a Deductive Enterprise Model (DEM) of which this research is based on, meaning GEM is a subset of DEM.

Considering the domain for this work, the in-depth knowledge or understanding of the carrying capacity in each department of the institution will be obtained by both the modeler (engineer) and the user through the automatic support thereby enhancing their ability to make future recommendations.

More than a simple data model is needed in other to deduce what is implied by the model. Axioms provide the basis of ontology's deductive capability. This supports enterprise operations by deducing answers to commonly asked questions.

Note that without the common-sense deductive capability, users spend significant resources to program each new function that is required. The field of ontological engineering has emerged over the last five years to provide a more formal approach to enterprise modeling which implies the construction of a DEM. DEM must be flexible and consistent to elucidate the problem of different representation of the same enterprise knowledge.

ENTERPRISE KNOWLEDGE REPRESENTATION

Knowledge representation has to do with writing down in some languages a communicative medium description or pictures that

corresponds in some salient ways to the world or some state of the world. Knowledge representation is most fundamentally a substitute used to enable an entity to determine consequences by thinking rather than acting.

First order logic (FOL) is the language used for representing the domain knowledge. FOL is a formal language that deals with a theoretical model of constructing its realities.

The conceptualization of the domain knowledge was based on some competency questions which will ascertain the efficiency of our model. The competency questions are as stated in section two.

Enterprise objects/entities and relations

The following objects/entities were identified in our domain of discourse. ***classroom (room-id, capacity)***

Note that the classroom here might be lecture theatre, laboratory or ordinary classroom. In any of case, there is a room-id and capacity which gives the number space or seats the classroom can accommodate.

course (course-id, dept-list, credit-unit, title, specialization)

A course is defined by the following attributes: the course-id, list of departments teaching it, credit units for the course, course title and specialization.

instructor (instructor-id, rank, specialization-list)

An instructor is identified by his/her unique instructor-id, rank and the area of specialization of that instructor.

dept-level (dept-name, level, no)

The department-level is defined by the name of the department taking the course, level of students from the department offering the course, and the number of students offering the course from that department.

The following predicates are also defined as relations:

canteach (instructor-id, course)

An instructor who specializes in a particular field can teach certain courses. The attributes include the instructor-id teaching the course and the course he/she is teaching.

teaches (instructor-id, course)

An instructor teaches a course. The attributes include the instructor-id and the course he/she is teaching.

holds (course, room-id, time, duration)

A course holds at a certain room-id at a particular time. And the course will last for a certain number of hours (duration). Hence, course code, room-id, time and duration are attributes of the holds relation.

take-course (dept-name, level, course-id, status)

This predicate is defined by the name of department taking the course, the level of students offering the course and the identity of the course being taken and the status that tells whether the course is compulsory or required or elective.

Deductive Capabilities of Domain Rules

From the above concepts definition, other predicates can be deduced and in-turn used for further deductions. This is seen in the axioms stated below. Each of the axioms starts with its literary meaning or rule. The axioms were written in First Order Logic.

Axiom 1

If an instructor who specializes in a certain field teaches a certain course and this field has relationship with course under study, then the instructor can teach the course.

$$\forall\ i, c, r, sl, u.$$
$$\text{instructor}(I, r, sl) \wedge \text{course}(c, dl, u, t, s) \wedge \text{member}(s,sl) \Rightarrow \text{canteach}(i,c)$$

where,

i	=	instructor_id,
r	=	rank of the instructor,
sl	=	specialization list,
c	=	course_id,
dl	=	department_list,
u	=	credit unit,
t	=	course title and
s	=	field of specialization.

An instructor can teach a course, if he/she specializes in that particular field. The axiom answers such questions as: can an instructor teach a course? If an instructor specializes in the field, then *canteach* is deduced.

Axiom 2

If a department takes a course and the department-level is known with the number of students offering the course from that department and course status is compulsory or required, then the number of students in that department is added to the list of all the students offering the course from various departments.

Case 1 This has to do with where only two departments are taken into consideration.

$$\forall d1, l1, c, s.$$
$$\text{take\_course}(d1, l1, c, s) \wedge \text{dept\_level}(d1, l1, a) \wedge (s = \text{Compulsory} \vee s = \text{Required}) \wedge \text{total}(c,[\ ], 0)$$
$$\Rightarrow \text{total}(c,[d1], a).$$

where

d1	=	department's name,
l1	=	level,
c	=	course-id,

s = course status,

a = constant holding the number of students offering the course c, from the department.

Case 2 This involves more than two departments.

$\forall$ d1, l1, c, s.

take_course(d1, l1, c, s) $\wedge$ (s=Compulsory $\vee$ s = Required) $\wedge$ dept_level (d1, l1, a) $\wedge$ total(c, dls, t) $\Rightarrow$ total (c, dls1, t+a) $\wedge$ dls1 = append(dls, d1)

Where

d1 = name of the department taking the course,

l1 = level in which the course belongs,

c = course-id,

s = course status,

dls1 = updated list of departments offering the course,

t = total number from the department offering the course which when added to a gives the total number from all departments offering the course.

That the total number of students offering a course from a certain department is needed to allocate that group of students to the classroom that has the capacity equivalent to the number of students.

Axiom 3

If the total number of students is known from the list of all the students offering a course and a certain department takes the course and the department exists in the overall list, then the total number on the list stands as the final total.

$\forall$ c, dl, s.

total(c,dl,s) $\wedge$ ($\forall$d,c,st. take_course(d, l, c, st) $\wedge$ member(d, dl)) $\Rightarrow$ final_total(c,dl,s)

where

c = course-id,

dl = department level,

s = status of the course.

From this axiom, the total number of departments offering the course can be deduced.

Axiom 4

If a course holds at a certain room and the actual total of students offering the course is known and the capacity of the classroom is also known and the number of students is greater that the room capacity, then the room is insufficient for the course to be handled in it.

$\forall$c, r, t, d.

holds(c, r, t, d) $\wedge$ final_total(c, d, n) $\wedge$ classroom(r,cap)

$\wedge$ (n > cap) $\Rightarrow$ Insufficient(r,c).

Where

c = course_id,

r = room-id,

t = time,

cap = capacity of the room,

n = number of students registering for the course.

That a room is insufficient for a course is deduced from the total number of students registering for the course and the capacity the classroom can carry.

Axiom 5

If time for the second course is between the start time for the first course and the start time plus duration OR the time for the first course is between the start time of the second course and start time plus duration, then time overlap occurs.

$\forall$ t, t1, d, d1.

$((t \leq t1 \leq t+d) \vee (t1 \leq t \leq t1+d1))$

$\Rightarrow$ overlap(t, t1).

where

t = time for the first course

t1 = time for the second course

d = duration for the first course

d1 = duration for the second course.

This axiom reports that there is overlap when two courses are slated for the same time t and t1 or the duration for one of the courses having to coincide with the time of the other course.

Axiom 6

If a course is schedule to hold at a certain room, at a certain time and will last for some time and another course is also scheduled to holds at the same room, at a certain time and duration, and the time for the second course overlap with that of the first then there is venue clash.

$\forall$ c, t, r, d, c1, t1, d1.

holds(c, r, t, d) $\wedge$ holds(c1, r, t1, d1) $\wedge$

overlap(t, t1) $\Rightarrow$.clash(c,c1).

where

c = course-id,

t = time,

d = duration of course,

r = room-id,

c1 = course offered by another department d1 at time t1 and at the same room-id, r.

This axiom states that there is a venue clash among the courses taken by these departments with the same room-id, and if the time chosen by concerned departments overlaps.

Deductive enterprise model

Axiom 7

If a department takes a course and the same department takes another course and both courses holding at certain classrooms and at certain times, and the time scheduled for the two courses overlaps, then there is time clash.

$\forall$ d, l, c, s, c1, s1, r, t, dr, t1, d1. take_course(d, l, c, s) $\wedge$ take_course(d1, l, c1, s1) $\wedge$ d=d1 $\wedge$ holds(c, r, t, dr) $\wedge$ holds(c1, r1, t1, dr1) $\wedge$ overlap(t, t1) $\Rightarrow$ clash(c,c1).

where

d, d1	=	department's name,
I	=	level,
c	=	course-id,
s	=	status of course,
r	=	room-id,
dr	=	duration of course.

This axiom reports the occurrence of time clash between two courses offered by a department, and both courses holding at the same time.

Axiom 8

If an instructor teaches a course and also teaches another course and both courses holding at different venue and at specific time and duration, and the time scheduled for the two courses overlaps then there is lecturer clash.

$\forall$ i, c, r, t, d, i1, c1, r1, t1, d1. teaches(i,c) $\wedge$ teaches(i1, c1) $\wedge$ holds(c, r, t, d) $\wedge$ holds(c1, r1, t1, d1) $\wedge$ (i = i1) $\wedge$ overlap(t, t1) $\Rightarrow$ clash(c,c1).

where

i	=	instructor-id,
c	=	course-id,
t	=	time,
d	=	duration of course,
r	=	room-id,
c1	=	course offered by another department d1 at time t1 and at the same room-id, r.1

This axiom tells of lecturer clash when the time scheduled for two courses taught by the same lecturer overlaps.

Axiom 9

If an instructor teaches a course and also teaches another course and the total credit units taught by this instructor is known and is greater than 8 credit units, then the instructor has excess load.

$\forall$ i, c, cu, d, t, s. teaches(i, c) $\wedge$ course(c, d, cu, t, s) $\wedge$ teaches(i, c) $\wedge$ course(c1, d1, cu1, t1, s1) $\wedge$ credit_total(I, dl, b) $\wedge$ (b > 8) $\Rightarrow$ excessload(i).

where

i	=	instructor-id,
c	=	course-id taught by the instructor,
d	=	department taking the course,
cu	=	credit unit for the course,
t	=	time scheduled for the course
s	=	status of the course.

This axiom checks the total credit unit (load) taken by each lecturer to know whether he/she has excess workload.

Derivation of PROLOG clauses from logical axioms

To add to these predicates which are facts in the knowledge based, certain rules are given for further deductions to be made and the result of these deductions, added to the knowledge based as facts. Such rules were highlighted in our axioms given in section 2.5. The implementation of these axioms is given in the corresponding PROLOG code given below. It's important to note that PROLOG starts its implementation from the goal. To confirm that the goal is true, all other facts or clauses must be satisfied. Once the clauses have been searched for and found to exist in the knowledge based, the goal is true and added as fact in the knowledge based also, and we can say that a deduction have just been made.

The axioms have been implemented in PROLOG and both the first order logic and the corresponding PROLOG versions are highlighted below.

1. $\forall$x,l. x $\in$ (x) $\vee$ x$\in$(y,l) [FOL]
   member(X,[X|_]).
   member(X,[Y|L]):- member(X,L). [PRO]

This PROLOG implementation uses the member predicate to check and ensure that x is in the list L, then it returns true for the predicate.

2. $\forall$t,t1,d,d1. (t $\leq$ t1 $\leq$ t+d) $\vee$ (t1 $\leq$ t $\leq$ t1+d1) [FOL]
   overlap(T,Dr,T1,Dr1):- T =<T1, T1<(T + Dr);
   T1=<T, T<(T1 + Dr1).[PRO]

This rule returns true for overlap between two slated time for probably two different courses considering their duration and if one coincides with the other, then overlap is true, else it returns false.

3. $\forall$ i, c, r, sl, u.
   instructor(I, r, sl) $\wedge$ course(c, dl, u, t, s) $\wedge$
   member(s,sl) $\Rightarrow$ canteach(i,c) [FOL]
   canteach(I,C):- instructor(I,R,S), course(C,Dl,U,T,Sl),
   member(S,Sl). [PRO]

This rule adds that an instructor can teach a course if probably the instructor is known to exist and the course in question is also taken by a certain department and the area of specialization of that instructor is a member of the specialization list of the course.

4. ∀d1, l1, c, s.
take_course(d1, l1, c, s) ∧ dept_level(d1, l1, a) ∧
(s = Compulsory ∨ s = Required) ∧ total(c,[ ], 0)
⇒ total (c,[d1], a). [FOL] total([],0).
total([Number|Numbers],Sum):-
total(Numbers,Sum1),
Sum is Sum1 + Number. [PRO]

This returns x if x is added to an empty list or adds x to y returning the total of the two.

5. ∀ d1, l1, c, s.
take_course(d1, l1, c, s) ∧ (s=Compulsory ∨
s = Required) ∧ dept_level (d1, l1, a) ∧ total(c,
dls, t) ⇒ total (c, dls1, t+a) ∧ dls1 = append(dls,d1)
[FOL]
cutotal(_,L,S):-
cutotal(_,L,0,S).
cutotal(_,[],S,S).
cutotal(I,[Cu|Credit1],Psum,Tsum1):-
teaches(I,C),
course(C,D,Cu,T,S),
Npsum is Psum +Cu,
cutotal(I,Credit1,Npsum,Tsum1). [PRO]

In a situation where more than one course are taught by a lecturer, this adds the credit unit of all the courses taught by the lecturer together.

6. ∀ c, dl, s.
total(c,dl,s) ∧ (∀d,c,st. take_course(d, l, c, st) ∧
member(d, dl)) ⇒ final_total(c,dl,s) [FOL]
sttotal(_,L,S):-
sttotal(_,L,0,S).
sttotal(_,[],S,S).
sttotal(_,[N|Num],Psum,Tnum):-
take_course(D1,V,C,St),
dept_level(D1,V,N),
(St = compulsory;
St = required),
Npsum is Psum + N,
sttotal(C,Num,Npsum,Tnum).
[PRO]

In a situation where more than one department offers a course, this adds the number of registered students in all the departments together.

7. ∀c, r, t, d.
holds(c, r, t, d) ∧ final_total(c, d, n) ∧
classroom(r,cap) ∧ (n > cap) ⇒ Insufficient(r,c).
[FOL]
insufficient_seat(R,C):-
holds(C,R,T,Du),
classroom(R,Capacity),
take_course(D,L,C,_),
dept_level(D,L,N),
N>Capacity. [PRO]

This rule checks for any slated course where the capacity of the classroom is not sufficient for the number of students that register for the course.

8. ∀ c, t, r, d, c1, t1, d1.
holds(c, r, t, d) ∧ holds(c1, r, t1, d1) ^((t ≤ t1 ≤
t+d) ∨ (t1 ≤ t ≤ t1+d1)) ⇒.clash(c,c1). [FOL]
clash(C,C1):-
holds(C,R,T,Dr) ,
holds(C1,R,T1,Dr1),
overlap(T,Dr,T1,Dr1).
[PRO]

This returns true for an occurrence of a time clash.

9. ∀ d, l, c, s, c1, s1, r, t, dr, t1, d1.
take_course(d, l, c, s) ∧ take_course(d1, l, c1, s1)
∧ d=d1 ^ holds(c, r, t, dr) ∧ holds(c1, r1, t1, dr1)
^((t ≤ t1 ≤ t+dr) ∨ (t1 ≤ t ≤ t1+dr1))
⇒ clash(c,c1).[FOL]
clash(C,C1):-
take_course(D,L,C,S),
take_course(D1,L,C1,S1),
D is D1,
holds(C,R,T,Dr),
holds(C1,R1,T1,Dr1),
overlap(T,Dr,T1,Dr1).
[PRO]

This adds that a venue clash has taken place between two groups of people handling different course.

10. ∀ i, c, r, t, d, i1, c1, r1, t1, d1.
teaches(i,c) ∧ teaches(i1, c1) ∧ holds(c, r, t, d) ∧
holds(c1, r1, t1, d1) ∧ (i = i1)
^overlap(t,dr,t1,dr1) ⇒clash(c,c1). [FOL]
clash(C,C1):-
teaches(I,C),
teaches(I1,C1),
holds(C,R,T,Dr),
holds(C1,R1,T1,Dr1),
I is I1,

overlap(T,Dr,T1,Dr1). [PRO]

This adds that there is a lecturer clash, where a lecturer is found wanting in two different classes at the same time.

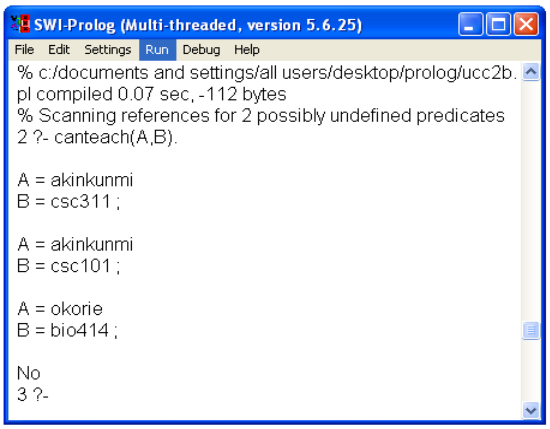
11. ∀ i, c, cu, d, t, s.
teaches(i, c) ∧ course(c, d, cu, t, s) ∧ teaches(i, c) ∧
course(c1, d1, cu1, t1, s1) ∧ credit_total(I, dl, b) ∧ (b > 8)
⇒ excessload(i).
[FOL]
excessload(I):-
teaches(I,C),
course(C,D,Cu,T,S),
teaches(I,C1),
course(C1,D1,Cu1,T1,S1),
C\=C1, cutotal(I,[Cu,Cu1],B), B>8. [PRO]

This adds to the knowledge based the fact that an instructor happens to handle more course than he/she should carry.

Analysis of the DEM

From the above PROLOG codes, one can pose queries that can answer the earlier stated competency questions. Below are example of these competency questions and the query posed and the result from the PROLOG system.

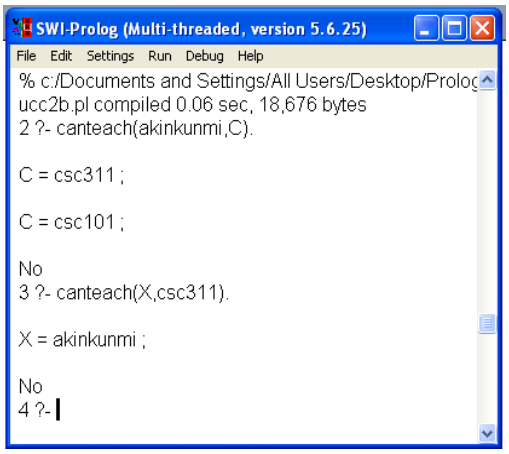
Can lecturer A teach course X?



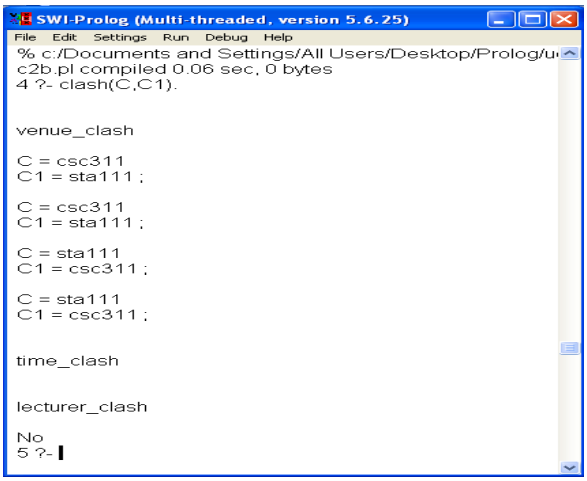
The query, *canteach(A,B).*, is posed with A and B being variables and PROLOG searches the knowledge_base for the possibilities which appear on our result screen. This tells us that *akinkunmi* is qualified to teach the course csc311 and csc101. In the same way *okorie* can also teach bio414.

One may also want to know the courses that a particular lecturer can teach or the qualified lecturer to teach a particular course. Posing the queries in the result screen tells us the efficiency of our model by answering the competency question:

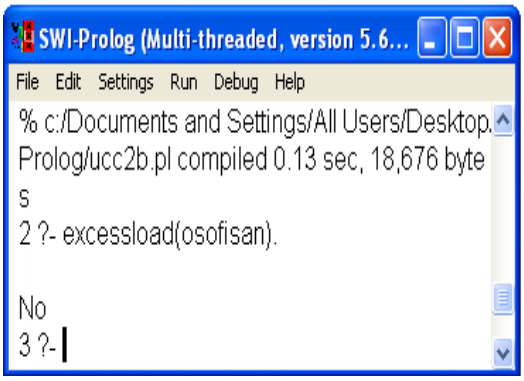
Is lecturer A qualified to teach course Y?



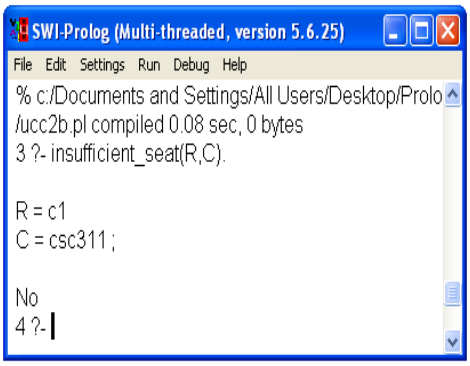
Questions like: Is there any occurrence of clash between two courses? What type of clash is it? and which of the courses are involved in the clash?, are also answered when posing some queries like the ones in the result screen below



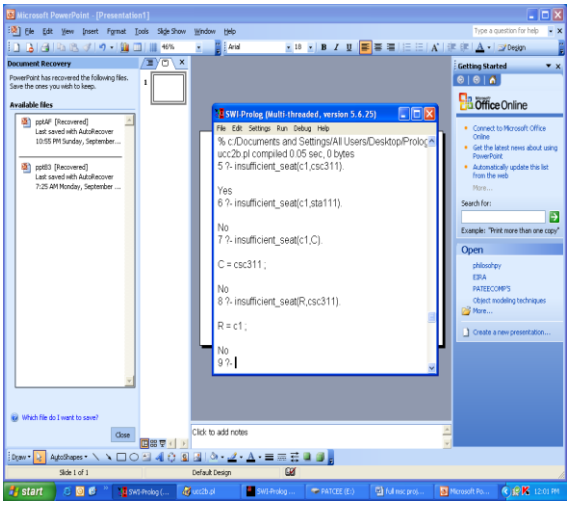
When one wants to know whether a particular lecturer or instructor has excess workload, queries like the one given in the result screen below answers the question: Does a particular lecturer has excess workload? The system gives 'Yes' or 'No' as the answer.



The next query posed is the one that determines whether the capacity of a classroom is sufficient for a particular group of students offering a course.



Another instance of queries posed to this rule is when the classroom and the course is known, then one can directly ask whether there is insufficient seat and the system will tell by saying 'Yes' or 'No'.



The above result screens handling the insufficient_seats predicate give answers to competency questions like:

Is a classroom capacity sufficient for the course allocated?

Are there courses allocated to classrooms where seats are not enough for the number of students registered for the course?

The above sample queries and results are obtained when testing the efficiency and competence of this research model.

CONCLUSION

The deductive enterprise model for the university academic system (University of Ibadan) was given. With the formalization of the university academic system (carrying capacity), most enterprise control and managerial problems of knowing whether to increase the number of students to be enrolled in a programme/department, or increase the number of lecturers employed or to build new structures can be tackled. This is not peculiar to academic systems only but could be extended to other administrative processes within the university or other enterprises as a whole. Issues like planning of examination venue can be done with the help of our model with little enhancement to the ontology. The introduction of an additional processing determines the answers to be provided, called inference capability assumes heritance as a deductive mechanism that lead to the deductive enterprise model which differ from the generic enterprise model with pre-defined object libraries.

The following recommendations should be followed by the concerned bodies. They include:

(i) Awareness should be created in the artificial intelligence (AI) field from the introductory level of computer science learning in our schools, colleges and universities. Probably, at the university level, AI courses should be made compulsory.

(ii)The federal government, like other countries of the world, should encourage AI computing by sponsoring AI research work within and outside the country.

(iii) Universities should introduce the use of this theoretical model in most planning and control decisions making by appropriate bodies/units of the university.

(iv) NUC should enforce the use of this model by our universities.

REFERENCES

Copeland, J. (2000). What is Artificial Intelligence? http://www.c.usfca.edu/www.AlanTuring.net/turing_archive/pages/Reference%20Articles/what_is_AI/What%20is%20A108.html.

Fadel, F. G.; Fox, M. S.; and Gruninger M. (1994). A Generic Enterprise Resource Ontology. In: Proceedings of the 3rd IEEE Workshop on Enabling Technologies: *Infrastructure for Collaborative Enterprises*, *Morgantown*, West Virginia.

Fox, M. S. (1992). *The TOVE Project Towards a Common-Sense Model of the Enterprise*. Toronto, Ontario.

Fox, M. S., Barbuceanu, M, Gruninger, M. and Lin, J. (1997). *An Organization Ontology for Enterprise Modeling.* Enterprise Integration Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto, Canada. http://www.eil.utoronto.ca/enterprise-modelling /papers.org prietula-23aug97.pdf

Fox, M. S., Chionglo, J. F. and Fadel, F. G. (1993). A common-sense model of the enterprise, Toronto, Ontario, Canada. In :Proceedings of the *2nd Industrial Engineering Research Conference*:425-429.

Fox, M., Gruninger, M. and Zhan, Y. (1993). Enterprise Engineering: An Information Systems Perspective. In: *Proceedings of the 3rd Industrial Engineering Research Conference (1994),* Atlanta GA.

Galton, A (1990). *Logic for Information Technology, Paper Edition.* John Wiley and Sons Publishers, New York.

Gomez-Perez, A, Garcia-Pinar, J. M., Fernandez, M. and Blazquez, M. (2005). *Building Ontologies at the Knowledge Level using the Ontology*, Design Environment. Madrid, Spain.

Gruber, T. R. (1991). The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference,* (J. A. Allen, R. Fikes, & E. Sandewall Eds.)M.A. Morgan Kaufmann ,Cambridge, : 601-602.

Gruninger, M. and Fox, M. S. (1996). *The Logic of Enterprise Modeling, modeling and Methodologies for Enterprise Integration,* (P. Bernus and L-Nemes Eds.), Chapman and Hall Cornwall, Great Britain.

Guarino, N. (1995). Formal Ontology, Conceptual Analysis and Knowledge Representation. In: *International Journal of Human Computer Studies.(* Guarino, N. and Poli, R Eds): 625-640.

Guide to Prolog Programming. http://www.ktiml.mff.cuni.cz/~bartak/prolog/first-steps.html.

Stadder, J. (1997). Towards a framework for enterprise modeling and integration. http://www.aiai.ed.ac.uk/project /enterprise/

Kim, H. M., Fox, M. S., Gruninger, M. (1995). *An Ontology of Quality for Enterprise Modeling.* Department of Industrial Engineering, University of Toronto, Toronto, Ontario, Canada.

Kriz, J. and Sugaya, H. (1989). *Logic Programming for Industrial Applications in Concepts and Characteristics of Knowledge-Based Systems.* Elsevier Science publishers, North Holland.

Locke, C. (1999). Common Knowledge or Superior Ignorance. http://www.psych.utoronto.ca/~reingold/courses/ai/cyc.html.

Noy, N. F. and McGuinness D. L.(2003). *Ontology Development 101*: A Guide to Creating Your First Ontology, Stanford University, Stanford, CA. http://protege.stanford.edu/publications/ontologydevelopment/ontoloy101-noy-mcguinness.pdf

Porto, A. (1989). *A framework for deducing useful answers to queries in Concepts and Characteristics of Knowledge-Based Systems*. Elsevier publishers, North Holland.

Prolog Programming Language. http://www.engin.umd.umich.edu/CIS/course.des/cis400/prolog/prol_g.html

Prolog Programming Tutorial I – Overview. http://www.cse.cuhk.edu.hk/~csc4510/prolog/tutorial-1/1.html

Tham, K.D., Fox, M. S., Gruninger, M. (1994). *A cost ontology for enterprise Modeling.* Department of Industrial Engineering, University of Toronto, Ontario, Canada.

University of Ibadan Academic Calendar, 1999-2002, University Ibadan press, Ibadan Nigeria.

University of Ibadan Faculty of Science Prospectus for 2005/2006 and 2006/2007 session, University of Ibadan Press, Ibadan.

Uschold, M., King, M, Moralee, S. and Zorgios, Y. (1998) *The Enterprise Ontology* The Knowledge Engineering Review , Vol. 13, Special Issue on Putting Ontologies to Use.

Yu, E. S. K. and Mylpoulos, J. (1997). Modeling Organizational Issues for Enterprise Integration. University of Toronto, Toronto, Ontario, Canada. http://www.fis.utoronto.ca/~yu