ENIAM: A More Complete Conceptual Schema Language

Peter Creasy

Key Centre for Software Technology
Department of Computer Science
University of Queensland
St. Lucia Qld 4067
Australia

Abstract

The problem of knowledge representation has been extensively addressed in the information systems field and there has been a wide range of suggestions for capturing semantics. A number of systems and methodologies have been proposed. Most seem to suffer from a variety of problems which make representation difficult, e.g. mixing of type and instances, too many and complex structures. Representation languages which permit a graphical form are becoming more widely accepted under the motivation of being able to see the data structures which exist. However the semantics are somewhat limited with, for example, few integrity constraints being represented. In this paper we show how a conceptual schema language with a graphical form can be extended to permit a wider class of semantics to be represented.

1. Introduction

Conceptual schema languages with a graphical form have become increasingly significant in information systems (e.g. Harel 1988) as the importance of being able to see what the data structures are becomes accepted.

An essential part of any knowledge representation system is the representation of facts. As syntactic rules permit facts which are syntactically valid but semantically invalid within the universe of discourse, we additionally need constraint specifications. Correspondingly we distinguish two classes of data types - the fact encoding construct (a data structure type which is associated with a population of instances) and validation rules. It is the latter structures which are of interest in this paper.

NIAM (Nijssen 1977, 1986; Verheijn & van Bekkum 1982) is a methodology with an associated fact-based language with a graphical representation. It uses only a few validation constructs (e.g. mandatory role, uniqueness). While we can represent, what we regard as some of the more important constraints, there is still a large class which cannot be represented. In addition subtype definitions and derivation rules have no formal representation.

Most of the presentations in the integrity constraint area (e.g. Reiter 1988) use first order predicate logic (FOPL), a language unfamiliar to most analysts. Existential graphs (Peirce 1960; Roberts 1973) have been developed in a number of forms which correspond to propositional, first order and higher order logics. The graphs provide a very readable logic representation in graphical form which can be readily integrated with graphical knowledge representation languages. We discuss the first order form which is represented graphically using negation, conjunction and the existential quantifier.

We use the existential graphs to extensively extend the power of NIAM. This provides us with ENIAM, a graphical first order logic language which permits the subtype definitions, derivation rules and constraints to be formally graphically represented.

Finally, we discuss an extension to the NIAM methodology which guides analysts/users in the expression of constraints in ENIAM.

2. NIAM

The NIAM language is fact-based. It has a graphical representation which is the form generally used for conceptual schema design. The facts (fact type instances) are semantically irreducible associations between entity instances, with each entity instance playing a role in the fact (sentence). The methodology abstracts from facts to give fact types, the fact encoding construct.

Figure 1 shows a NIAM diagram which represents people holding passports and visiting countries. The main constructs are: entity types (circles), e.g. PERSON, COUNTRY; roles (rectangles), e.g. visited, was-visited-by, which correspond to verbs of a sentence; fact types (compound rectangles), e.g. VISIT, HOLD; label types (dotted circles), e.g. person-name; and constraints.

A relationship type between entity types is a fact type. Each fact or reference type consists of a number of roles. These describe the part (or role) each entity or label type (to which the role is joined) plays in the fact or reference type. For example the role played by PERSON in CALLED is "is-named".

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

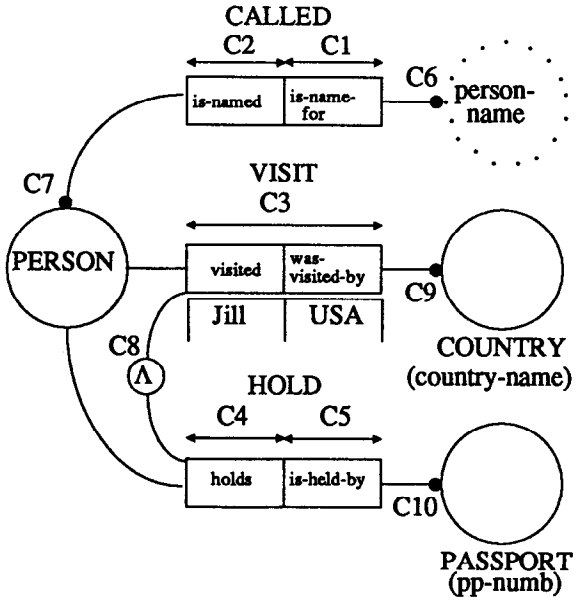


Figure 1:

Entities are non-lexical and are lexically identified by reference types (e.g. CALLED). These are special fact types which associate entity types with the identifying label type. The references can be abbreviated by writing the corresponding label type in parentheses following the entity. We have shown this for the entity type COUNTRY.

An example of a VISIT fact type instance is <Jill,USA>. We can show such populations as in figure 1, where the non-lexical enitites have been represented by the corresponding lexical representation. This fact can be read as *the person identified by person-name Jill visited the country identified by country-name USA.*

Some of the constraints we can represent in NIAM are intrafact (and intrareference), interfact uniqueness, mandatory role, equality, exclusion and subset constraints. The arrows above the roles represent intrafact or intrareference uniqueness constraints which are placed on the instances of the role or roles. They indicate that the role or roles uniquely identify a single fact instance, i.e. they specify functional dependencies. In this case constraints C4 and C5 indicate person $\longleftrightarrow$ passport while C3 indicates person, country $\rightarrow \phi$. Constraints C1 and C2 indicate unique naming for the entity type PERSON.

The constraint C7 is a mandatory role constraint which indicates that if an entity (or label) takes part in any fact (or reference) instance then it must take part in an instance of the fact (or reference) type corresponding to the constraint. The subset constraint C8 indicates that anyone who visits a country must have a passport.

The fact types can be mapped to relational structures using, what is essentially, a synthesis algorithm. For figure 1 the resulting relations are VISIT(person , country) and HOLD(person , passport).

We have not mentioned all the features of the language. One which should be mentioned is subtyping, an important part of the language. An example is shown later. It should be noted that it is not necessary to write down the populations, fact type names, role names or constraint names unless they are needed for clarification. This results in a much less cluttered diagram.

3. Existential Graphs

Existential graphs were proposed by Peirce to represent logic and resulted from the work he did over a number of years. The graphs are simple with few concepts. They are one of many graphical representations for logic which have been proposed over the years, e.g. Venn diagrams (see Gardner 1983). The graphs use few operators (user symbols, lines and closures e.g. circles), are easy to read and write and offer the power of FOPL. Peirce proposed a number of what he called *parts* to the existential graphs. He called these the Alpha, Beta and Gamma parts. The parts of interest to us here are the Alpha part (equivalent to propositional logic) and the Beta part (equivalent to FOPL). Roberts (1973) has shown completeness and consistency for existential graphs. In particular Roberts uses Quine's (1955) ML logic system. Sowa (1984) has demonstrated the usefulness of existential graphs as a mechanism for representing FOPL.

The Alpha part has only three basic symbols: the sheet of assertion, the cut and the graph. The graphs are laid out on the *sheet of assertion* (SA), an arbitrarily large area, which equates to a model of the universe of discourse. A graph instance is something which asserts a possible state in the universe, viz. proposition. The sheet of assertion is also considered to be a graph, as the blank SA expresses whatever initially holds in the UoD.

Graph instances written on the SA are asserted to be true. By a graph is meant "any sign (symbol) which expresses in a proposition some state in the universe" (Roberts, p 32).

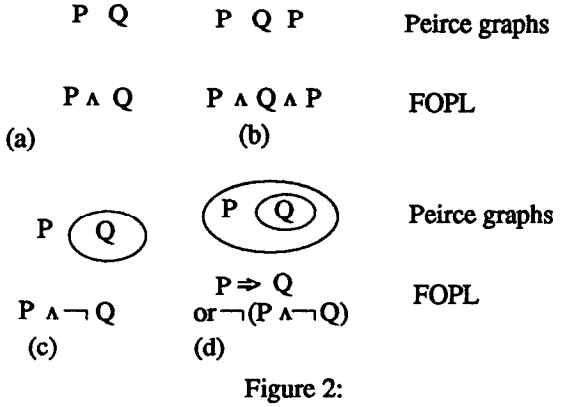


Figure 2:

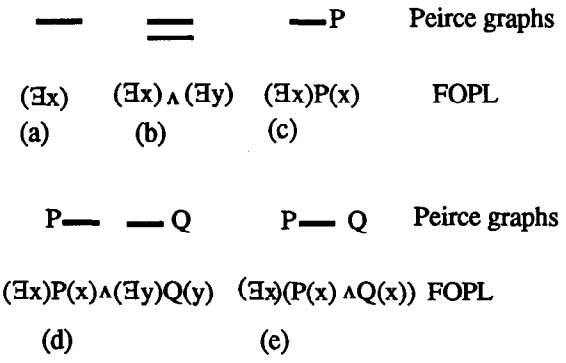


Figure 3:

Some examples are shown in figure 2. It should be noted that the shape of the graphs (e.g. whether lines are straight or curved or the shape of the closures) is not important. Figure 2(a) contains two graphs which together state the propositions represented by P and Q hold, i.e. the conjunction of these holds. Peirce distinguished graph and graph instance. In his terms there is strictly only one graph P of which there may be a number of occurrences (graph instances); e.g. we may have P repeated a number of times such as in figure 2(b). In the following we shall use the term graph rather than graph instance, except where we wish to make the distinction.

A single line, with no crossings, can be drawn around a graph (a cut) to indicate the negation of the graph. For example figure 2(c) is equivalent to $P \wedge \neg Q$. For material implication (e.g. $P \Rightarrow Q$, or "if P then Q") the term "scroll" is used. This example is shown in figure 2(d). The graphs are read from the outside inwards (*endoporeutic*). In this example (figure 2(d)) reading from the outside inwards we have the antecedent ("if P") and then the consequent ("then Q").

The Beta part additionally has a *line of identity*, used to represent an individual in the universe, and a *spot* which is equivalent to the FOPL predicate, e.g. P of figure 3(c). In the Alpha part the graph and the (propositional) symbol were one and the same. However in the Beta part a graph may consist of a number of symbols, hence the need to distinguish the symbol and the graph. The line of figure 3(a) indicates the existence of something, e,g, $(\exists x)$, while figure 3(b) indicates $(\exists x)(\exists y)$, i.e. two individuals which may be the same.

We still have the convention that two graphs drawn on the same SA represent the conjunction of the graphs. Thus figure 3(d) represents $(\exists x)P(x) \wedge (\exists y)Q(y)$. In figure 3(e) the line of identity between P and Q indicates that the two individuals represented by the ends are identical, i.e. $(\exists x)(\exists y)(P(x) \wedge Q(y) \wedge x = y)$ or $(\exists x)(P(x) \wedge Q(x))$. This can be extended to n individuals.

Just as a predicate can have more than one argument, more than one line may be attached to a predicate (symbol). The

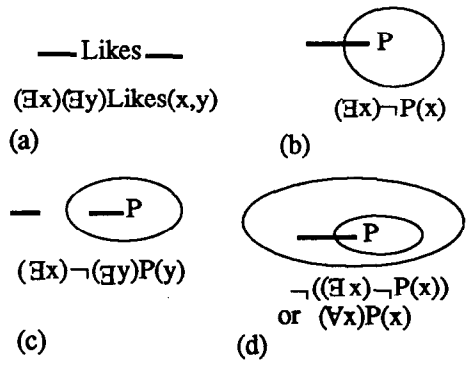


Figure 4:

places of attachment are called *hooks*, and the meaning of the hooks must be understood just as the positions of the predicate arguments in FOPL need to be understood. Figure 4(a) represents *someone likes someone*. In this case we arbitrarily decided to consider the hook in front of the predicate to be the person doing the liking and the other hook to be the person liked.

It is possible to have any nesting of cuts. Graphs are referred to as being evenly enclosed or oddly enclosed, depending on the depth of nesting. This is significant as those graphs evenly enclosed have been negated an even number of times and thus have a positive sign (no negation) in the expanded FOPL (or propositional logic) form, while those oddly enclosed have a negation sign. Anything not enclosed (i.e. on the SA) is considered to be evenly enclosed.

When a line (of identity) crosses a cut the nesting of the cuts determines the equivalent FOPL variable scoping, with the outermost extremity of the line determining the position of the existential quantifier (the endoporeutic view). For example figure 4(b) is equivalent to $(\exists x)\neg P(x)$, i.e. the existence is at the outer level; figure 4(c) is equivalent to $(\exists x)\neg(\exists y)P(y)$; figure 4(d), where the outer part of the line of identity is oddly enclosed and P is evenly enclosed, is equivalent to $(\neg\exists x)P(x)$, i.e. $(\forall x)P(x)$, in this case the existence is at level 1.

A line of identity passing through an empty cut indicates the non-identity of the individuals at the extremities. For example figure 5(a) is equivalent to $(\exists x)(\exists y)(P(x) \wedge Q(y) \wedge x \neq y)$. The negation of a ligature indicates the non-identity of the individuals denoted by the extremities of the ligature.

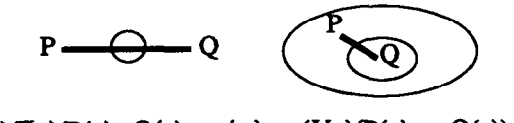


Figure 5:

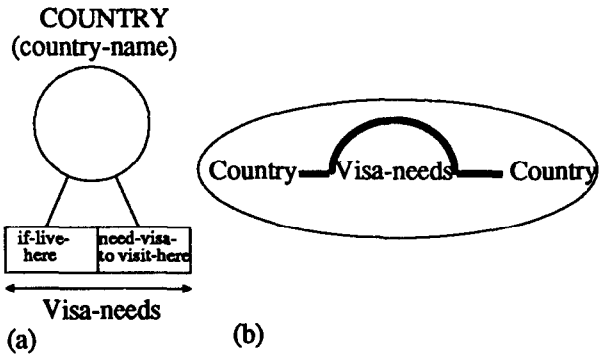


Figure 6:

We shall look at some examples to introduce a methodology for drawing the graphs and to compare the graphs with FOPL. As we have seen material implication is drawn with what Peirce called a scroll. Thus "if something is a P then it is a Q" is shown in figure 5(b). In this type of structure the antecedent is always oddly enclosed and the consequent evenly enclosed.

With only negation and conjunction we need these to represent disjunction. While this may appear awkward, we emphasize that the visual representation is important and one can quickly learn to recognize and draw this picture as a disjunction.

Constants are not treated differently from other symbols, which we have so far thought of as predicates. We treat a constant the same as a unary predicate. If necessary, we can distinguish constants by enclosing them in quotes. However for the usage with NIAM, as we shall see, this is not necessary as there is no ambiguity.

4. ENIAM

NIAM constraints are set oriented. In terms of sets, the existential graphs are "member oriented" in that we express in an exclusion constraint, say, that there is no element that is in both sets involved in the constraint. This potentially gives greater scope for expressing constraints. For example, given the NIAM schema of figure 6(a), we may wish to express that *one does not need a visa to visit ones own country*. We cannot express this with NIAM's set constraints as a country will generally appear in both roles, i.e. in both sets, and we thus can't use the exclusion constraint. However existential graphs do allow us to express "there cannot be a country which acts as a lives-in and a requires-visa-for in the same row". This is shown in figure 6(b). The combination of the NIAM constraints and the existential graph constraints is thus potentially powerful and simple.

To be useful as a constraints language the logic first needs to be extended by introducing "inequality" predicates ($<$, $\leq$, $\geq$, $>$, $\neq$) for numeric values. We can then use these predicates as in figure 7, which indicates that for *an active flight (one which we can make a booking on) the capacity must be less than the booked seats.* Bookings(flight,seats) and capacity(flight,seats) represent the bookings and capacity for an active flight. Either of the forms of figure 7 are acceptable. We interpret figure 7(a) as

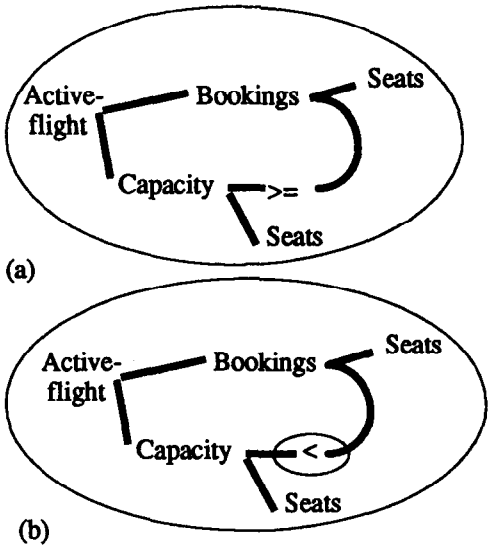


Figure 7:

$$\neg((\exists x)(\exists y)(\exists z)(\text{Active-flight}(x) \wedge \text{Seats}(y) \wedge \text{Seats}(z) \wedge \text{Bookings}(x,y) \wedge \text{Capacity}(x,z) \wedge y \leq z)).$$

One of the advantages of a graphical language is allowing a user to see immediately the structures (e.g. relationships between object types). Over-complication of a diagram defeats this advantage; thus it is important to keep the number of graphical symbols to a minimum. By using existential graphs it permits the full power of FOPL to be used to express constraints with few symbols. We could use set constraints of NIAM when appropriate, i.e. to express what we shall call our "specialty" constraints (uniqueness, mandatory role etc.), and existential graph constraints when membership constraints are required or NIAM does not have the appropriate symbol.

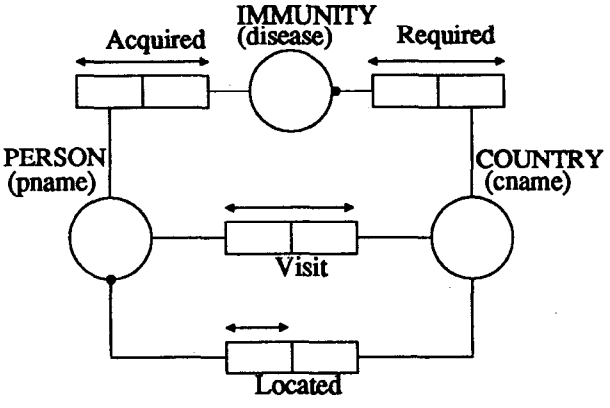


Figure 8:

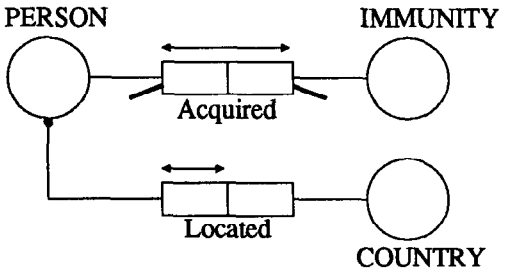


Figure 9:

4.1. The Extended Language

We shall use figure 8 as our example in demonstrating how we can expand NIAM's constraints. As most of the examples don't apply to all of figure 8 we shall only present the relevant sections in the examples. In figure 8 people are scheduled to visit countries. Each country has certain vaccination (immunisation) requirements and each person is recorded as having certain vaccinations. It is required to ensure that people have the necessary vaccinations to visit the countries they have been scheduled to visit.

If we wish to express the constraint that *there must be someone who has a vaccination* we use the fact type Acquired as an existential graph predicate as in figure 9.

As a further example suppose we have the subset constraint *anyone visiting a country must have a have a vaccination* we express this as the embedded existential graph in figure 10. However, if we use the fact types as predicates directly the diagram rapidly becomes unmanageable. For practical reasons we are excluded from using further instances of the predicates.

We thus also propose taking a "copy" or "image" of the relevant fact types and use those as the predicates in an existential graphs diagram. Thus the above constraint can be rewritten as in figure 11. The image A is an "image" of the fact population Acquired (what we have previously called a predicate instance). The dotted line indicates the image. The image is drawn in the same orientation as the fact type, thus the image role r is the PERSON hook. If it is required to draw the image in a different orientation to

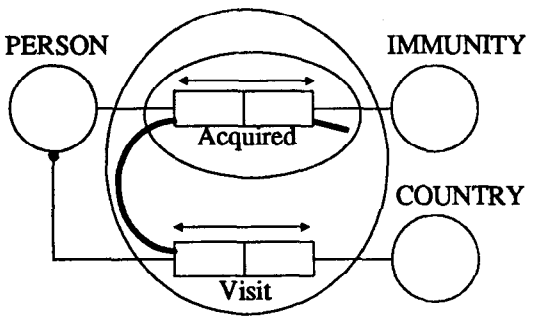


Figure 10:

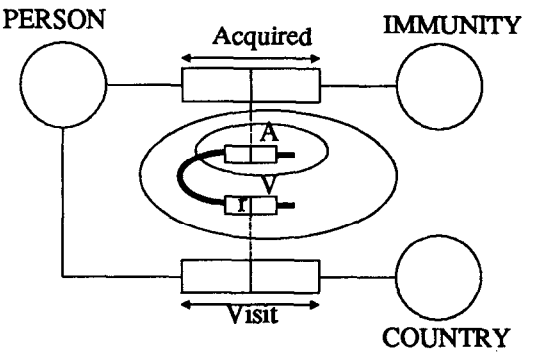


Figure 11:

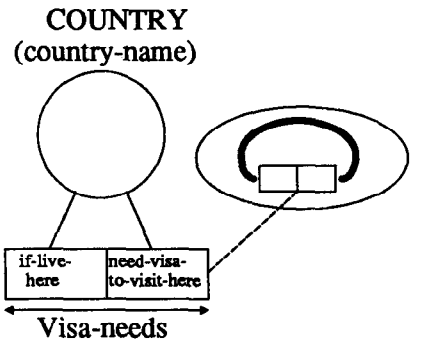


Figure 12:

the fact type we can always attach the image by a dotted line from a (non-central) role to the corresponding role in the fact type, although it would normally be obvious from the types over which the roles are defined. By a "non-central" role we mean a role other than the middle role of an n-ary for odd n. For example we express the constraint *no one needs a visa to visit their own country* as in figure 12, although in this case it is not really relevant which way the attachment is made.

In the embedded existential graphs we shall explicitly represent constants. For example *all visitors from Australia have a Polio vaccination* would be represented as in figure 13.

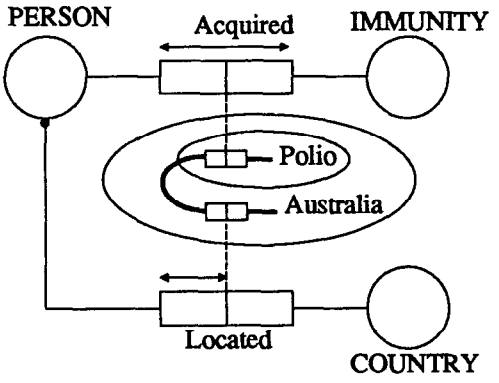


Figure 13:

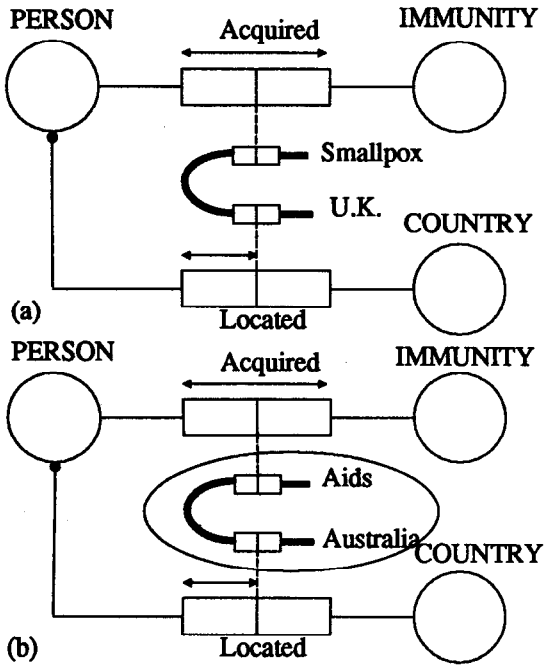


Figure 14:

The constraint *at least one U.K. visitor has a smallpox vaccination* is shown in figure 14(a), *no visitor from Australia has an Aids vaccination* is shown in figure 14(b).

Often we need constraints involving more than 2 fact types, in particular those involving a join. For example we may require that *people visiting a country must have the necessary vaccinations*, viz.

(Visit * Required)[person,immunity] $\subseteq$ Acquired.

We can express this in ENIAM in a more natural manner: *if a person is visiting a country which requires vaccinations then the person must have those vaccinations.* Using the scroll we express this in ENIAM as in figure 15.

We can also use the existential graphs to represent derivation rules. Where rules are of the form "consequent if antecedent",

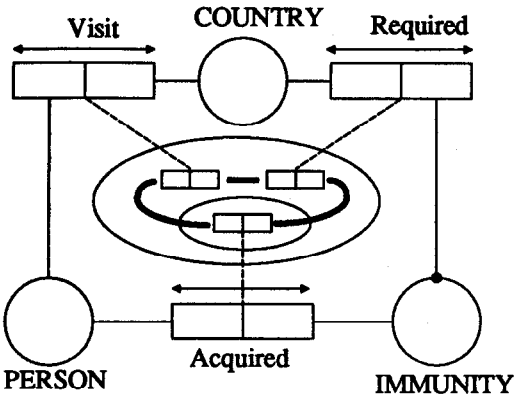


Figure 15:

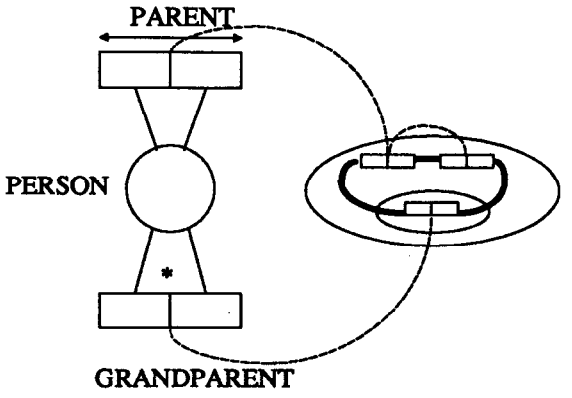


Figure 16:

e.g. *x is the grand-parent-of z if x is the parent-of y and y is the parent-of z* then we can use the scroll. This particular example is shown in figure 16.

It would be desirable to be able to represent an element of an entity type, i.e. to have the equivalent of unary type predicates. In this case we express the image of an entity type as a large blob. The constraint of figure 17 indicates that there exists at least one person in the information base (which does lead to problems in not permitting us to have an empty information base).

Given this construct we have a representation for the mandatory role. We represent *every person has a surname* as in figure 18.

We can use this construct to represent the subtype definition. The example of figure 19 is taken from Halpin (1988). We shall represent, say, the TVviewer definition as:

if Person viewing TV for PeriodRatio > 0 then TVviewer

Using this mechanism the subtype definition of figure 19 can be expressed as in figure 20.

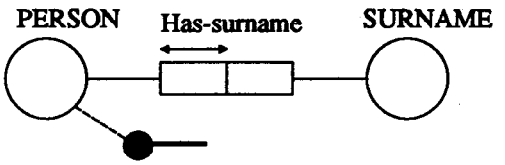


Figure 17:

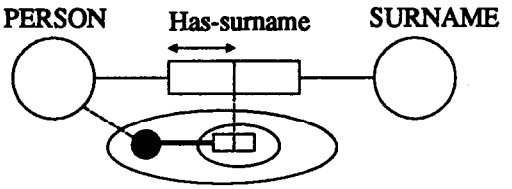
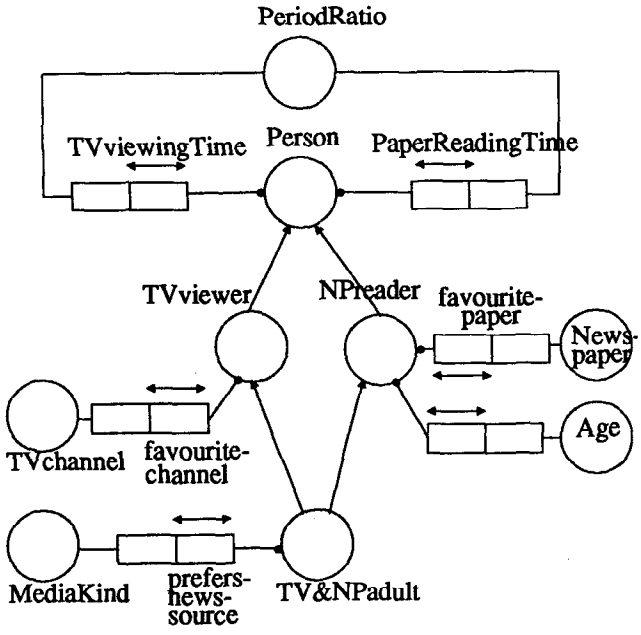


Figure 18:



TVviewer = Person viewing TV for PeriodRatio > 0
NPreader = Person reading newspapers for PeriodRatio > 0
TV&NPadult = TVviewer ∩ NPreader having Age > 18

Figure 19:

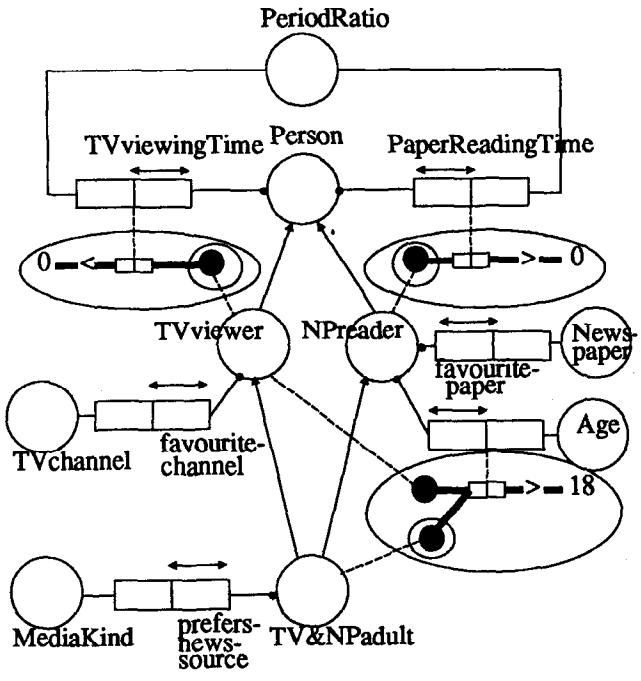


Figure 20:

4.2. The ENIAM Methodology

For most constraints we tend to use the "if then" construct (viz. the scroll). In our examples so far the only ones not using the scroll were existence or non-existence constraints. These constraints are fairly straight forward. The only problem anyone may have is in phrasing them as a negation. The existence constraints, e.g. *someone is booked on a flight*, are the simplest. However, if we start with an empty database then no facts are initially present and thus such constraints are initially violated. Nevertheless we shall consider such constraints in the following.

We claim that there are four basic structures which we can use to represent most constraints. These are:

(i) existence, e.g. *someone is to visit France*, figure 21(a).

(ii) non-existence

(a) of one element, e.g. *there is someone who is not to visit France*, figure 21(b).

(b) all elements, e.g. *no one is to visiting France*, e.g. figure 21(c).

(iii) scroll, e.g. *everyone is to visit France*, figure 21(d).

Whenever we have constraints like *someone . .*, or *Jill . .*, i.e. those involving the existence of a fact we use a type (i) structure. For constraints that involve the existence of an unspecified individual for whom we wish to express a negative we use (iia). Other negatives without a conditional

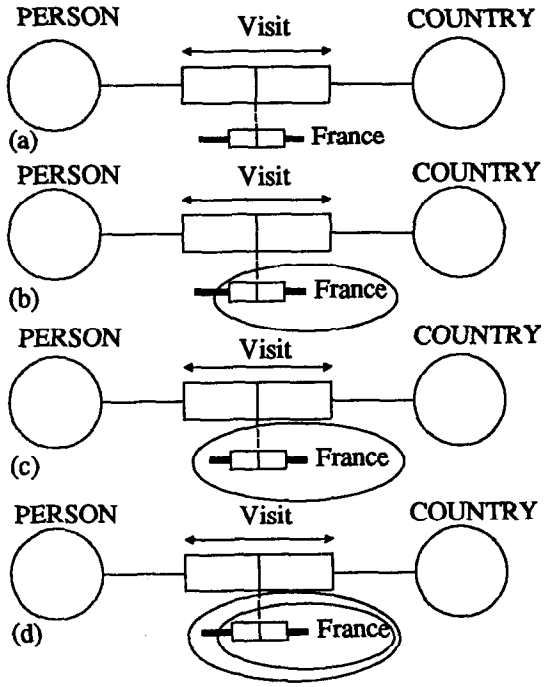


Figure 21:

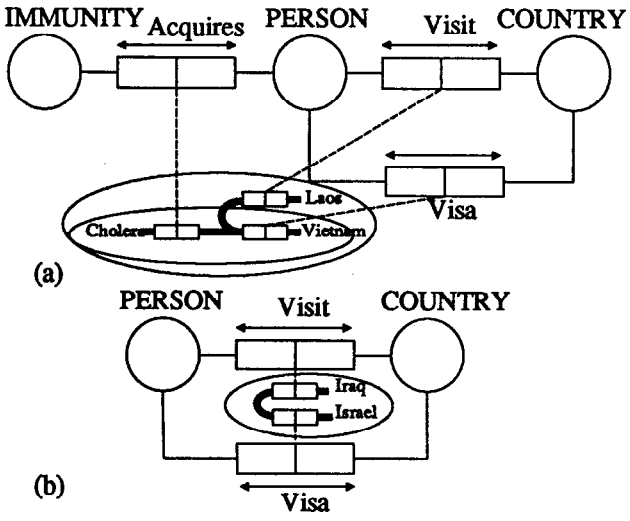


Figure 22:

such as *no <statement>*, e.g. *no person is to visit France* and those involving specific individuals, e.g. *Bill is not to visit France*, use a type (iib) structure.

Those involving (a) conditionals, such as *if <condition> then <statement>*, e.g. if someone visits a country that person needs a passport, (b) negative conditionals, e.g. *no one other than people visiting a country needs a passport*, and *no one may visit a country without a passport*, and (c) those involving all elements of a set, such as *any, every, when, whenever*, e.g. *everyone visiting a country needs a passport* use a type (iii) structure.

For multiple conditions and statements etc. we join the predicates as we have previously suggested. For example *everyone who visits Laos needs a cholera vaccination and a visa for Vietnam* would be expressed as in figure 22(a). While *no one may visit Iraq and have a visa to Israel* would be expressed as in figure 22(b).

5. Conclusions

We have shown that we can significantly extend the power of NIAM using a relatively simple graphical language which has the power of FOPL. We have demonstrated in a classroom environment that existential graphs can be easily learned and ENIAM used to express constraints, subtype definitions and deduction rules. We believe there is no necessity for analysts to have a background in logic to use ENIAM.

As we showed in section 4.2 most constraints involve the scroll and will not involve structures more complicated than those shown. The various forms can be learned by example and combined as necessary. ENIAM has the advantage that, if required, the full power of FOPL can be used.

We have shown elsewhere (Creasy 1988) that the graphs can be simply mapped to Prolog. We have also shown (loc. cit.) that the technique suggested by Nicolas (1982) can be applied to the graphs to permit efficient computation of most constraints. We have implemented the algorithms and demonstrated the feasibility of the technique.

6. References

Creasy, P.N. (1988) "Extending Graphical Conceptual Schema Languages", Internal Report, University of Queensland.

Gardner, M. (1983) *Logic machines and Diagrams*, (first published 1958), The Harvester Press, Brighton.

Halpin, T.H. (1988) *Introduction to Information Systems*, CS112 Lecture Notes, University of Queensland.

Harel, D. (1988) "On Visual Formalisms", *Comm ACM* **31**, 5 (May), pp 514-530.

Nicolas, J-M. (1982) "Logic for Improving Integrity Checking in Relational Data Bases," *Acta Informatica* **18**, pp 227-253.

Nijssen, G.M. (1977) "On the Gross Architecture of the Next Generation Data Base Management Systems," Proceedings 1977 IFIP Congress, Toronto. North Holland, Amsterdam.

Nijssen, G.M. (1986) "On Experience with Large-Scale Teaching and Use of Fact-based Conceptual Schemas in Industry and University," in *Proceedings IFIP Conference on Data Semantics (DS-1)*, R. Meersman and T.B. Steel Jr (Eds), Elsevier North-Holland, Amsterdam.

Peirce, C.S. (1960) *Collected Papers of Charles Sanders Peirce*, vol 4, A.W. Burks (Ed.), Harvard University Press, Cambridge, Mass.

Quine, W. V. (1955) *Mathematical Logic*, (revised edition) Harvard University Press, Cambridge, Mass.

Reiter, R. (1988) "On Integrity Constraints," *Proceedings 2nd Conference of Theoretical Aspects of Reasoning about Knowledge*, Pacific Grove, California, March 7-9, pp 97-111.

Roberts, D.D. (1973) *The Existential Graphs of Charles S. Peirce*, Mouton, The Hague.

Sowa, J.F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Weseley, Reading, Massachusetts.

Verheijn, G. and van Bekkum, J. (1982) "NIAM: An Information Analysis Method" in *Information System Methodologies - A framework for understanding*, Olle et al (Eds), CRIS3 Task Group of IFIP WG8.1, IFIP.