TOWARDS MORE POWERFUL CONCEPTUAL SCHEMA LANGUAGES

Peter Creasy
University of Queensland

TOWARDS MORE POWERFUL CONCEPTUAL SCHEMA LANGUAGES

Peter Creasy
Key Centre for Software Technology
Department of Computer Science
University of Queensland

ABSTRACT

One of the phases of information systems design methodologies is that of conceptual schema design. This phase involves identifying relevant objects, their properties and propositions involving these objects. These are described in a conceptual schema using a conceptual schema language.

Conceptual schema languages which permit a graphical form are becoming more widely accepted under the motivation of being able to see the data structures that exist. However the semantics are somewhat limited with, for example, few integrity constraints being represented. In this paper we show how conceptual schema languages with a graphical form can be extended to permit a wider class of semantics to be represented.

1. INTRODUCTION

The conceptual schema design phase of information systems design is one of the more important phases. A number of methodologies and languages have been proposed for this phase. In addition, languages which have a graphical form have become increasingly significant in information systems (e.g., Harel 1988). These graphical forms have enabled the user to see the data structures.

Information systems have three perspectives (Olle et al. 1988):

- the data perspective which is concerned with the data to be recorded within an information system.
- the process perspective which is concerned only with processes which operate on this data.
- the behavior (or event-oriented) perspective which is concerned with events which cause the processes to be performed.

It is the first perspective which is of concern in this paper.

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.

A number of other models/languages have been proposed, in most of which, e.g., IFO (Abiteboul and Hull 1987), the "primary focus in the development...has been on the structural component of the model" (p. 526). This paper proposes not a new language but a method of extending existing languages to include the "integrity-specification component." The method could be applied to any fact-based language. We demonstrate the method using a fact-based language which already can represent a reasonably wide class of constraints.

NIAM (Verheijn and van Bekkum 1982; Nijssen and Halpin 1989) is a methodology with an associated fact-based language with a graphical representation. It is informally described using a few examples. The methodology involves taking a significant set of sentences from the UoD and from these abstracting to obtain the fact types (the information bearing construct).

However there are a number of problems with the language. 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.

The entity-relationship (E-R) model (Chen 1976) is a similar model, although without the methodology of NIAM. It has even fewer validation constructs than NIAM, although many proposals have been made in this area.

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 these languages to include a graphical first order logic language which permits the subtype definitions, derivation rules and constraints to be formally graphically represented. We show this for NIAM and E-R.

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

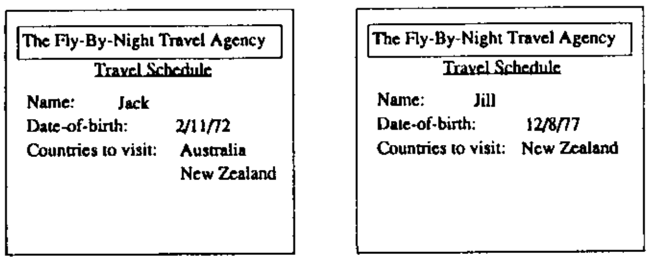
2. NIAM

We informally describe NIAM using a few examples. The methodology involves taking a significant set of sentences from the UoD and from these abstracting to obtain the fact types which, together with the database, is the fact encoding mechanism. The set of sentences should be significant in the sense that it will enable us to determine all constraints that can be represented in NIAM.

For example, let us take a travel agency business. Suppose we have forms such as in Figure 1. The NIAM methodology uses what Nijssen (1986) has called the "cookbook" approach. With this approach the users take familiar examples of facts from their application, for example, *Jack visits Australia*. We need now to take the user from the instance level to the type level. Nijssen suggests the heuristic of telephoning someone and informing him/her of the fact and "in this way the user is forced to make a visual to auditory transformation." With this verbalization process, the sentences are expressed in natural language rather than in a formal artificial language prescribed by "computing experts." There are some semi-formal rules for specifying the sentences, but this helps the users to formalize the problem in their minds.

The user may specify the statement

Jack is going to Australia.



The Fly-By-Night Travel Agency
Travel Schedule
Name: Jack
Date-of-birth: 2/11/72
Countries to visit: Australia
New Zealand

The Fly-By-Night Travel Agency
Travel Schedule
Name: Jill
Date-of-birth: 12/8/77
Countries to visit: New Zealand

Figure 1: Sample form

However, this is not structured enough. We are relying on the listener to interpret that Jack is a person and Australia is a country. As it is the user with the expert knowledge of the UoD, this domain information must come from the user. If we persuade our user to be a little more explicit, we should receive the message:

The person with person name Jack will visit the country with country name Australia.

Sentences such as this reveal the deep structure of the sentence. We are listening for verbs, entity categories or types, label categories or types and label instances. Examining sentences in terms of these structures we obtain:

The PERSON	**entity type**
with PERSON-NAME	**label type**
Jack	**label instance**
will-visit	**verb (role)**
the COUNTRY	**entity type**
with COUNTRY-NAME	**label type**
Australia	**label instance**
and	**connective**
was-born-on	**verb (role)**
the DATE	**entity type**
with EUROPEAN-DATE-CODE	**label type**
2/11/72	**label instance**

The user (the expert of the UoD) has been able to take us from a surface structure to a deep structure. From here we proceed to step 2 which involves a quality check on the elementary facts. We ensure that the entities are fully designated and that the facts cannot be split into smaller ones without loss of information.

We carry out the abstraction process at step 3. For this we ignore the label instances and present the remaining entity types, verbs and label types in a graphical form.

From this (and the remainder of a significant sample), we can derive the schema of Figure 2, although strictly it is a subschema. We have also included part of the population of the significant sample, however this is usually only done either for explanatory purposes or to check the schema. The label types (PERSON-NAME and COUNTRY-NAME) are represented as dotted circles. The current extension of the label type PERSON-NAME is Jack and Jill. The entity types (PERSON and COUNTRY) are represented as circles. The current extension of PERSON is those people called Jack and Jill. These entities are non-lexical. We have represented them by arbitrary lexical marker. Often in the diagrams, we use the corresponding lexical representation (e.g., Jack).

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.

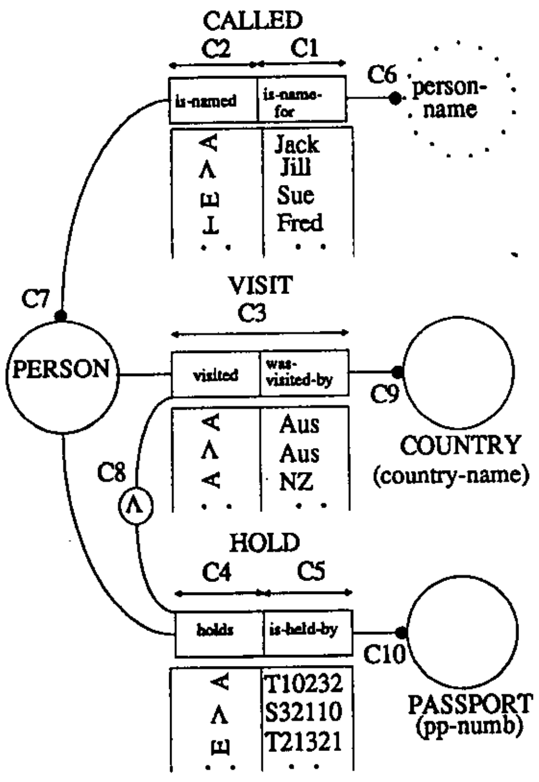


Figure 2: A NIAM Conceptual Schema

Figure 2 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."

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 2, where the non-lexical entities 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 the functional dependencies person $\leftrightarrow$ 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 2 the resulting relations are VISIT($\underline{\text{person , country}}$) and HOLD($\underline{\text{person}}$, $\underline{\text{passport}}$).

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 first proposed by Peirce (1960) at the beginning of the century. As a graphical representation of logic they were intended to be easy to learn, read and write. Many other forms have also been proposed (see Gardner 1983). The graphs have few operators, are easy to translate to the Peano-Russell notation and suit the purpose we intend for them.

The graphs have a number of parts:

- Alpha part, equivalent to propositional logic.
- Beta part, equivalent to first order predicate logic.
- Gamma part, equivalent to second, higher order and model logics.

Roberts (1973) has shown completeness and consistency for existential graphs. In particular, Roberts uses Quine's (1955) ML logic system.

We shall informally describe existential graphs and relate them, where appropriate, to the more widely understood propositional and first order logic and the Peano-Russell notation. 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 1973, p. 32). Some examples are shown in Figure 3. It should be noted that the shape of the graphs (whether lines are straight or curved or the shape of the closures) is not important. Figure 3(a) contains two graphs which together state that the propositions represented by Raining and Cold hold, i.e., the conjunction of these holds. Peirce distinguished graph and graph instance. In his terms, there is strictly only one graph, Raining, of which there may be a number of occurrences (graph instances); e.g., we may have Raining repeated a number of times such as in Figure 3(b). In the following we shall use the term graph rather than graph instance, except where we wish to make the distinction.

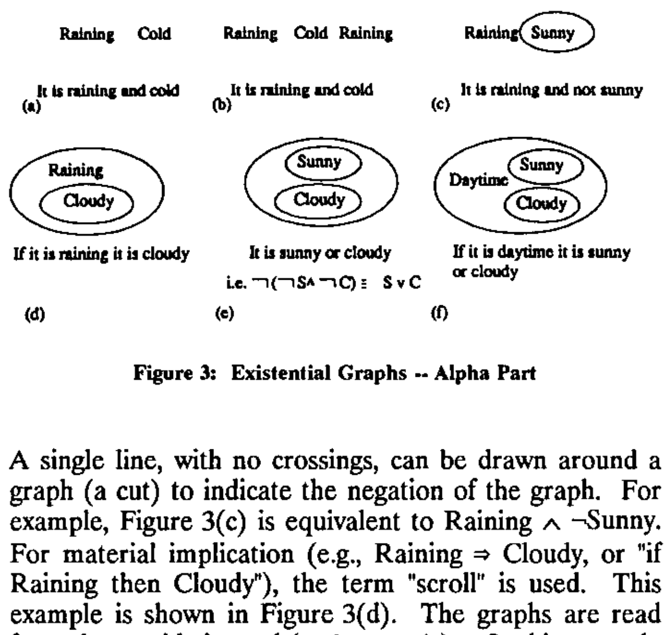


Figure 3: Existential Graphs -- Alpha Part

A single line, with no crossings, can be drawn around a graph (a cut) to indicate the negation of the graph. For example, Figure 3(c) is equivalent to Raining $\wedge$ ¬Sunny. For material implication (e.g., Raining ⇒ Cloudy, or "if Raining then Cloudy"), the term "scroll" is used. This example is shown in Figure 3(d). The graphs are read from the outside inward (*endoporeutic*). In this example (Figure 3(d)), reading from the outside inward we have the antecedent ("if Raining") and then the consequent ("then Cloudy").

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.

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., Cold of Figure 4(b). 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 4(a) indicates the existence of something, e.g., (∃x).

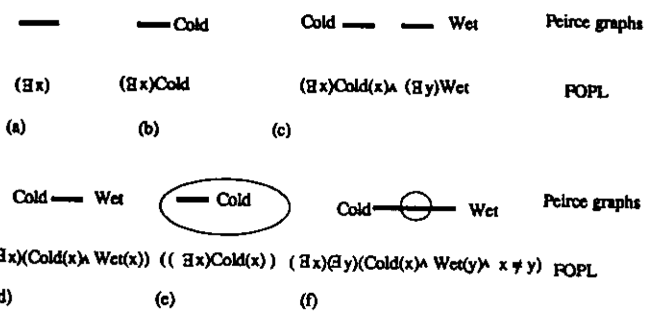


Figure 4: Existential Graphs -- Beta Part

We still have the convention that two graphs drawn on the same SA represent the conjunction of the graphs. Thus Figure 4(c) represents (∃x)Cold(x) $\wedge$ (∃y)Wet(y). In Figure 4(d), the line of identity between Cold and Wet indicates that the two individuals represented by the ends are identical, i.e., (∃x)(∃y)(Cold(x) $\wedge$ Wet(y) $\wedge$ x = y)) or (∃x)(Cold(x) $\wedge$ Wet(x)). This can be extended to n individuals.

A line of identity passing through an empty cut indicates the non-identity of the individuals at the extremities. For example, Figure 4(f) is equivalent to (∃x)(∃y)(Cold(x) $\wedge$ Wet(y) x≠y). The negation of a ligature indicates the non-identity of the individuals denoted by the extremities of the ligature.

Just as a predicate can have more than one argument, more than one line may be attached to a predicate (symbol). The 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 5(a) represents *someone is located somewhere*. In this case we arbitrarily decided to consider the hook in front of the predicate to be the person and the other hook to be the place.

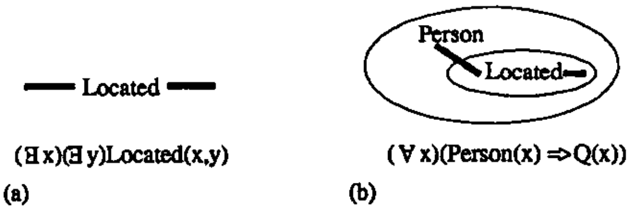


Figure 5: Binary Predicates

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 5(b) is equivalent to $\neg((\exists x)(\mathrm{Person}(x) \wedge \neg(\exists y)\mathrm{Located}(y)))$, i.e., $(\forall x)(\mathrm{Person}(x) \Rightarrow (\exists y)\mathrm{Located}(x,y))$. This figure also demonstrates the use of the scroll in the Beta part. After some acquaintance with the graphs it is very easy to read this as *everyone is located somewhere.* In this type of structure, the antecedent is always oddly enclosed and the consequent evenly enclosed.

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 our usage, as we shall see, this is not necessary as there is no ambiguity.

4. EXTENDING CONCEPTUAL SCHEMA LANGUAGES

While FOPL can be used as a conceptual schema language, it is very cumbersome. The fact type of NIAM and the relationship set of E-R are just predicates. However, both NIAM and E-R are strongly typed languages. The fact type VISIT of Figure 2 must involve a person and a country. Figure 5(b) *represents every person must be located somewhere,* although we have not specified the type for the second predicate.

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 one's 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.

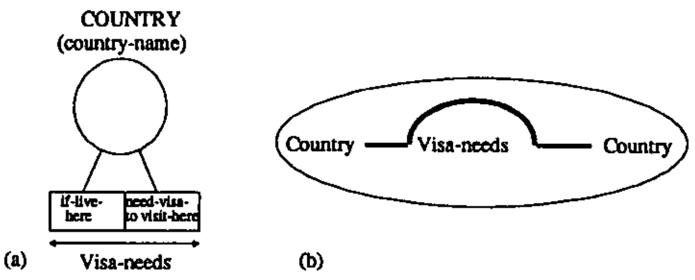


Figure 6: A Constraint with Existential Graphs

One of the advantages of a graphical language is allowing a user to see immediately the structures (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.

We shall incorporate existential graphs into NIAM and E-R diagrams. We shall use fact types and relationship sets as the existential graph predicates. NIAM cannot express existential quantification and thus we cannot express that *someone is located in a country.* We use the incorporation of NIAM and existential graphs of show this in Figure 7.

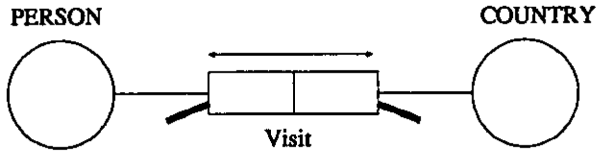


Figure 7: Incorporation of NIAM and Existential Graphs

To use a predicate for more than one logical expression we use predicate instances. We can do the same in the incorporated diagram by using an "image" of the fact type as in Figure 8, which represents that *there is a flight to the U.S.A.* Figure 9 specifies the constraint *Bookings must not exceed the capacity of the flight!*

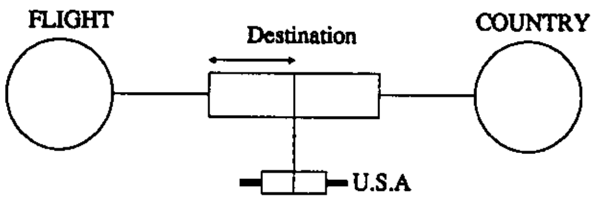


Figure 8: Using Images of Fact Types

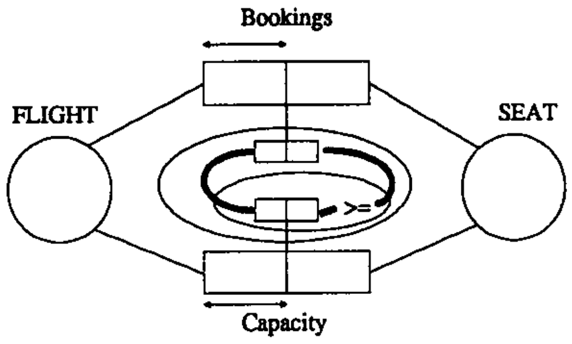


Figure 9: Constraint Specification with the Scroll

We can also use the existential graphs to represent derivation rules. Where rules are of the form "consequent if antecedent," 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. An example is shown in Figure 10, where the maximum bookings for a flight can be derived from the type of aircraft used on a flight and the capacity of the aircraft. This type of structure is also useful as the constraint *A person must be recorded as visiting a country if they are booked on a flight which has that country as a destination, viz.*

(Flies * Destination)[person,country] ⊆ Visit.

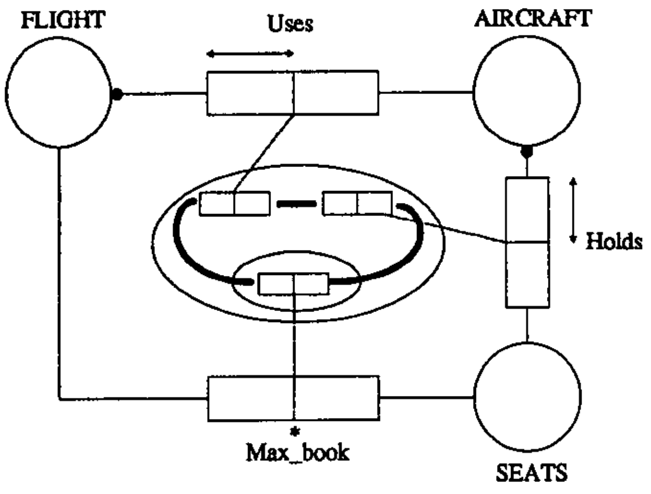


Figure 10: Derivation with the Scroll

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 11(a) 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 11(b).

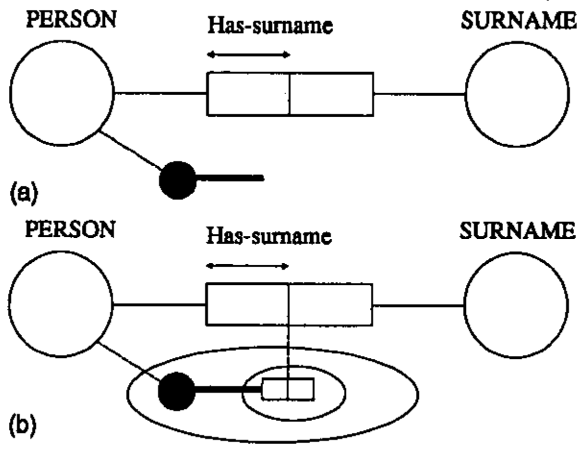


Figure 11: Using Entity Type Images

This construct can be used to represent the subtype definition. An example of a subtype definition is given in Figure 12. Traveller is a subtype of Person and is defined to be those people who visit countries.

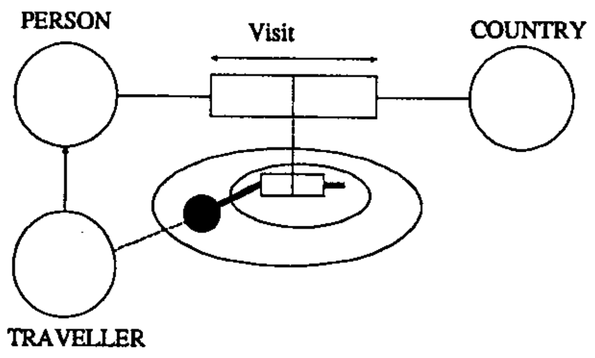


Figure 12: Subtype Specification

We can use existential graphs to extend the graphical form of any fact-based language. You can take your favorite graphical knowledge representation language and use the relationship type as the relationship predicate. If the language is strongly typed (as most such languages are), the same approach can be taken as we did with NIAM.

In particular the entity-relationship model (Chen 1976), which is widely used, is well suited for such an extension. E-R's relationship type and entity type are treated in the same way as NIAM's fact type and entity type. We simply use differently shaped image symbols to correspond to those of E-R. The example in Figure 13 is the E-R version of Figure 9.

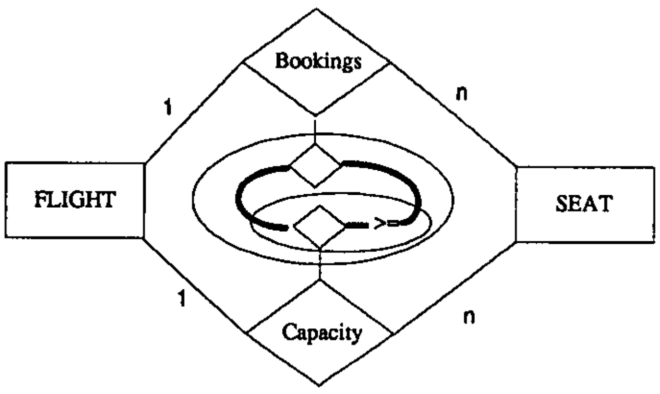


Figure 13: Incorporation of E-R and Existential Graphs

5. EXTENDING THE METHODOLOGIES

The extended methodology is independent of the language the graphs are being incorporated with. To make this point we give the examples using E-R.

For most constraints we tend to use the "if then" construct (the scroll). In our examples so far, the only ones not using the scroll were existence or non-existence constraints. These constraints are fairly straightforward. The only problem anyone may have is in phrasing them as a negation. The existence constraints (*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 14(a).

(ii) non-existence

(a) of one element, e.g., *there is someone who is not to visit France*, Figure 14(b).

(b) all elements, e.g., *no one is to visiting France*, e.g., Figure 14(c).

(iii) scroll, e.g., *everyone is to visit France*, Figure 14(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 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.

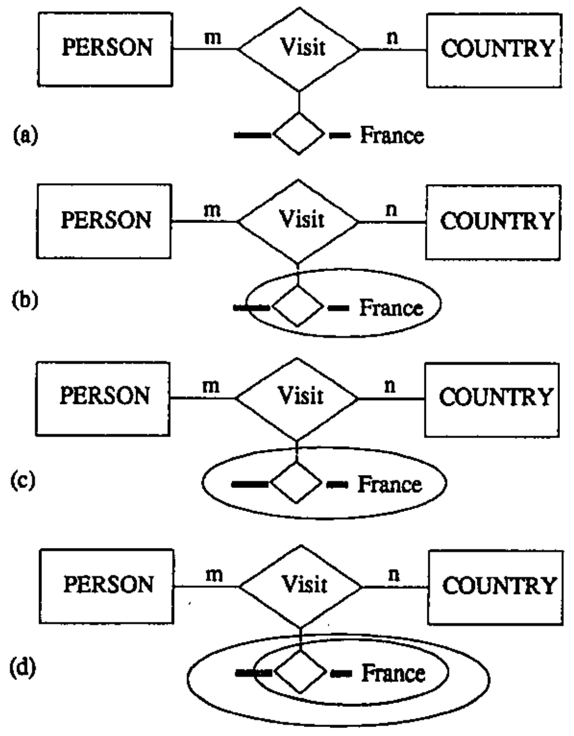


Figure 14: Possible Constraint Forms

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. Where there are multiple conditional statements, the predicates can be joined and expressions of any complexity composed.

6. CONCLUSIONS

We have shown that we can significantly extend the power of fact-based languages using a relatively simple graphical language which has the power of FOPL. The techniques could be used with any language with graphical predicates. The advantage of the fact-based languages is their strong typing.

We have demonstrated elsewhere (Creasy 1988) that existential graphs can be easily learned and 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 the graphs.

As shown in Section 5, most constraints involve the scroll and will not involve structures more complicated than those presented. The various forms can be learned by example and combined as necessary. The graphs have the advantage that, if required, the full power of FOPL can be used.

Having represented the constraints graphically, we need a method to compute them to determine whether or not they hold. We have shown elsewhere (Creasy 1988) that the graphs can be simply mapped to Prolog to achieve this. While Prolog may be useful in demonstrating the computability of the constraints, it is totally inefficient to recompute the constraints after each change of the database state. As an alternative method of computing, we note that the existential graph form also lends itself to a technique suggested by Nicolas (1982). This permits efficient computation of most constraints.

7. ACKNOWLEDGEMENTS

We wish to acknowledge the anonymous referees for constructive comments on the presentation of this paper.

8. REFERENCES

Abiteboul, S., and Hull, R. "IFO: A Formal Semantic Database Model." *ACM Transactions on Database Systems*, Volume 12, Number 4, December 1987, pp. 525-565.

Chen, P. P. "The Entity-Relationship Model -- Towards a Unified View of Data." *ACM Transactions on Database Systems*, Volume 1, Number 1, March 1976, pp. 9-36.

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

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

Harel, D. "On Visual Formalisms." *Communications ACM*, Volume 31, Number 5, May 1988, pp. 514-530.

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

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

Nijssen, G. M., and Halpin, T. A. *Conceptual Schema and Relational Database Design: A Fact-Based Approach*, Englewood Cliffs, New Jersey: Prentice Hall, 1989.

Olle, T. W.; Hagelstein, J.; Macdonald, I. G.; Rolland, C.; Sol, H. G.; Van Assche, F. J. M.; and Verrijn-Stuart, A. A. *Information Systems Methodologies -- A Framework for Understanding*. Wokingham, England: Addison-Wesley, 1988.

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

Quine, W. V. *Mathematical Logic*, (Revised Edition). Cambridge, Massachusetts: Harvard University Press, 1955.

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

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

Verheijn, G., and van Bekkum, J. "NIAM: An Information Analysis Method." In T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart (Eds.), *Information System Methodologies -- A Framework for Understanding*, CRIS3 Task Group of IFIP WG8.1. Amsterdam: North-Holland, 1982.