SEMANTIC REPRESENTATION OF PRODUCT MODELLING DATA

Ž TURK, PhD.,
University of Ljubljana, Civ.Eng.Dept.,
Jamova 2, Ljubljana, Slovenia;
e-mail: ziga.turk@uni-lj.si

ABSTRACT

Analysis and design needed for the development of engineering information technology (IT) solutions are inherently much more complex than comparative efforts in the scope of commercial applications. A semantic representation of conceptual model schema and other information is required. In the paper, logic programming is suggested for the internal representation of thesauri data and conceptual models. Logic programming specialises in setting the rules which define relations between objects; which is exactly the case in conceptual modelling. Prolog is a logic programming language and was successfully extended in this research to provide for the representation of data and logic of thesauri and conceptual models.

Key Words
conceptual modelling, Prolog, engineering database design, thesauri, NIAM, CAD.

INTRODUCTION

Developers and end-users of integrated CAD solutions have been using information technology to handle the growing volume and complexity of information that needs to be considered to achieve a competitive edge in the design. The motive for that is quite clear; tens and perhaps hundreds of tables with a total of more than one thousand fields are needed to structure information regarding a segment of the design process (like structural design). Several thousand may be needed to store the actual data about the design.

To be able to design systems of such complexity, more or less formal approaches have been used by the engineering software developers. They were adopted from the world of business database design (Leung and Nijssen, 1988, Appleton, 1988) or object oriented analysis and design (Turk, 1992) and in many ways augmented to accommodate for the differences between typical engineering and commercial information. The modelling approach, specifically "conceptual modelling approach" (Bjork and Pentilla, 1989) has established itself in the engineering database design community. Various techniques for information analysis have been proposed (comparison in Fereshetian and Eastman, 1991); graphical or textual grammars have been defined, formal approaches for the translation of the resulting graphs into databases and the verification of schemes have been suggested (Rasdorf and Abudayyeh, 1992).

The utilisation of information technology during the design of a conceptual model is limited to the ones that facilitate the manipulation of the representation of the conceptual model. These include:

- drawing and charting aids for the construction of a graphical representation of the results of information analysis.

- grammar parsers of the textual representation of the results of information analysis.
- specialised software for storing and retrieving analysis information, usually part of a 4GL relational database, expert system or object database.
- specialised software for translating analysis information into database schemes,
- various other general software like editors, word processors and presentation tools for the storage of the conceptual model data.

Common to all these aids is that they store only the passive representation of the conceptual models. The information in IDEF1X or NIAM charts and EXPRESS listings is passive in the sense that an external source of knowledge is needed for the interpretation of the information. NIST PDES Toolkit (Clark, 1990) is an example of such a tool.

The paper is based on the following:

Assumption: Information analysis in engineering disciplines results in a number of data types and relations which is an order of magnitude larger than in commercial applications.
Motive: Because of that, more advanced tools are needed for the analysis, representation and communication (exchange) of the results of the conceptual models.
Proposed solution: A computer understandable semantic representation of conceptual models and other related data should simplify the conception of computer based tools that would ease the development, improve the quality and ensure the correctness of conceptual models of engineering and architectural products.

Organisation of the paper

The paper will not focus on the alternative (relational, semantic, object data bases) representation of data that is being used during the analysis and design of engineering and architectural *artifacts* but on the data created during the design and analysis of *conceptual models of these artifacts*. Semantic representations and their potential usage will be investigated. The modelling approach will be recursively applied to the information modelling process (section "Representing models"). The representation in logic will be selected for the internal representation of the information analysis results. Examples of the utilisation of logic programming from the scope of thesauri representation and NIAM schema representation will be given in the section "Representation in logic". Section "Conclusions" will provide the evaluation of the results and some future research directions.

REPRESENTING MODELS

Creating models is considered one of the basic activities of the human brain. As observed by Spinoza, "it is easier to relate images with other images of those things that which we perceive clearly and transparently. The more images an image is related to, the more often it is alive (Spinoza, 1988)". Cognitive science that has been building on the European philological tradition, puts the ability to build mental models (Spinoza would call that a collection of related images) in direct correlation with the human intelligence. Consequently, knowledge itself was defined as the ability to build models (Sowa 1984).

In the computer age models are being implemented in computer software. Observing the process from this angle, computer programming is the transformation of mental models into

realisation within software. Such a computer-based model can be viewed as a product which is the result of some process - the modelling process (Fig. 1). The life cycle of such a model is similar to life cycles of engineering products. The analysis is followed by design, development, instantiation, usage and adaptation. In the figure's left branch: the development of the product model in software. Right branch: the usage of the product model for the design of products. The term "design" is thus used in two contexts and has two distinct meanings - one during the software development (software design, schema design), the other during product development (architectural design, engineering design).

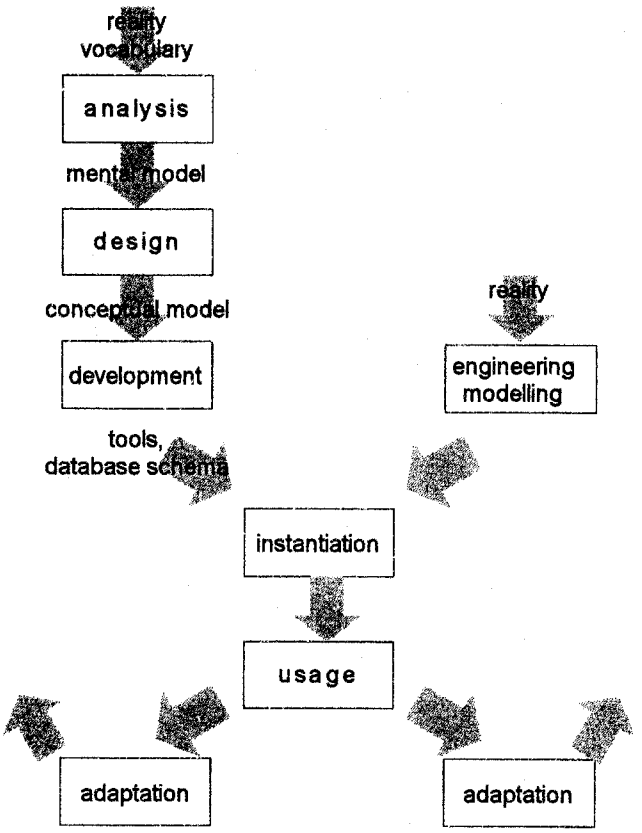

Fig. 1. The modelling processes.

Models are designed based on a UoD (Universe of Discourse) using a modelling technique. The technique includes (a.) modelling methods, (b.) strategies outlining how to design models and (c.) a notation for communicating the models with humans and computers. The notation for communicating the models with other computers requires some kind of a formally defined representation. Two types of representations may be distinguished.

Semantic vs. machine readable representation

The result of the design stage of the modelling process may be represented in a format readable by humans (often as diagrams) or by computers. Primary goal of the information modelling techniques is that humans (both application developers and end users) would understand the diagrams and the information they contain. It is desirable that the information models are stored in a computer readable format for further processing.

The developers expect that the conceptual model contains the necessary information needed to develop programs, which will manipulate the information discussed in the conceptual model. For example, such information is the fact that a window is a special kind of an opening and that an opening is associated with a wall type boundary element and that it has certain geometrical and topological attributes. The programs, which will deal with windows and openings are expected to simulate the understanding of these basic facts - they should allow the setting of the mentioned attributes and disallow placing a window in free space outside a wall plane.

What is not, however, expected is that the tools for the analysis and design would understand the meaning of the conceptual model. For instance, such a program should understand that if there is a two way relation between two fields, these should be indexed. It should understand that an aggregation relation and its inverse are in contradiction if applied to the same two entities. It should be able to warn against ambiguities, missing or recursive definitions... Applied to the example with windows, the conceptual modelling tool, knowing that a window is a kind of an opening, should not allow it to have a disjoint set of attributes (relations) to some other openings.

Any form of the description of conceptual models may be translated into a graph structure - a semantic network - which contains nodes and links. Nodes and links are basic building blocks of the Prolog language which is why the language was selected for storing the conceptual model information.

CAD data (about the designed artifacts) could be translated into such a schema as well, but the number of nodes and links is too big for such a representation to be effective. Because of that (1.) listed references of the application of Prolog to engineering all deal with very limited domains and (2.) in CAD systems, semantic networks are simplified into trees, hierarchies or related tables.

Programming with logic in engineering

Prolog language is an implementation of logic programming restricted to Horn clause theory. A Prolog program is a set of declarations, which define data objects and relations between them. Running a program means setting a goal and asking the program, if the goal could be satisfied, given the defined relations. Introduction to the language can be found (among others) in (Clocksin and Mellish, 1981) and its application for the artificial intelligence in (Bratko, 1986).

In engineering large majority of declarative knowledge manipulation was performed using friendlier tools like expert systems (Adeli, 1988). Some were using products based on Prolog language with the language itself hidden from the end user (Alim and Munro, 1988). The direct application of Prolog has been noted in areas with particularly strong need for the declaration of knowledge. Structural engineering design codes were translated into Prolog lists and were processed by the Prolog program STAPRO (Topping and Kumar, 1989). Knowledge about designing steel roof thrusses and the relevant code information was encoded as Prolog predicates (Koumousus and Georgiou, 1991). Thomson *et al* (Thomson *et al*, 1988) extended the language with new predicates for the encoding of design codes. It was used as a basis for an intelligent object oriented database (Watson and Chan, 1989) and in the design of relational databases (Obretenov *et al*, 1988). Eastman *et al* are using Prolog for the representation of the

EDM model (Eastman, 1992). An overview of application of logic programming to the design of architectural (and engineering) artifacts with examples on architectural spatial design can be found in Coyne (Coyne, 1988).

REPRESENTATION IN LOGIC

Representation of thesauri data

The research in the thesaurus representation was motivated by the work of Vanier on the hypertext representation of thesauri data (Vanier, 1992). Essential data from the hypertext version of the thesaurus was made available in ASCII (Fig. 2 left). Each term in the thesaurus can have ten sets of relations to other terms. It can have a French term (FT), broader term (BT), narrower term (NT), whole term (WT), part term (PT), use-for term (UF), use term (US), general term (GT), related term (RT) and associated term (AT).

It was being used as an additional source of knowledge during object oriented analysis and design (Turk and Vanier, 1993). Interactive browsing of the thesaurus and its usage in medium sized applications resulted in the following problems:

a. In order to get answers to slightly more complex queries like - is there a broader (narrower) term up (down) the hierarchy that has the same part term as this one? The answer is needed to define member and inheritance relations of objects.
b. After working on the designs manually for some time, an algorithm on how to do design, most notably the classification part, was beginning to emerge. It would be desirable to have a representation that would allow the programming of this algorithm.
c. Hypertext representation provides very limited tools for validation of the thesauri data in general. For instance are the inverse relations (like NT and BT) actually inverse for all instances. This special case has been validated before, but when developing new thesauri or adding to existing ones, such a facility is essential.

The ASCII version of the thesaurus (Fig. 2 left) was translated into Prolog. In clause form (Fig. 2 middle) each of the relations is defined as a functor. Data objects rBT, rUF, rPT and rRT are functors. Each relation is a structure of its own.

Most of the trivial questions like *what are the related terms of term column* are asked directly without any additional programming:

```
?- rRT(columns,X).
```

And the Prolog engine lists the following answers:

```
X = peristyles ;
X = piers_bridges ;
X = pilasters
```

A bit more imagination is needed to list hierarchies of, for instance, broader terms. If design_features are a broader term of the term column, then broader terms of design features are broader terms of columns as well. First, we need to define a functor for such hierarchically broader terms. We shall call it hBT:

```
hBT(X,Y):-
      rBT(X,Y).
hBT(X,Y):-
      rBT(X,Z),
      hBT(Z,Y).
```

It has been defined that any Y is a hierarchically broader term of X, if Y is a direct BT of X (first clause) or if there exists such a Z, which is a direct BT of X and Y is a hierarchically broader term of Z. After adding the two clauses to the knowledge base, the following session is possible:

```
?- hBT(columns,X).
```

Prolog list the answers:

```
X = design_features ;
X = building_functional_elements
```

ASCII source	*clause form*	*plain English*
**columns BT design features UF main columns posts(columns) secondary columns spiral columns stanchions FT colonne(architecture) PT bases(columns) capitals(column) plinths shafts(columns) RT peristyles piers(bridges) pilasters	rBT(columns,design_features). rUF(columns,main_columns). rUF(columns,posts_columns). rUF(columns,secondary_columns). rUF(columns,spiral_columns). rUF(columns,stanchions). rPT(columns,bases_columns). rPT(columns,capitals_column). rPT(columns,plinths). rPT(columns,shafts_columns). rRT(columns,peristyles). rRT(columns,piers_bridges). rRT(columns,pilasters).	Broader term of columns are design_features. Use for term of columns are main_columns. Use for term of columns are posts_columns. Use for term of columns are secondary_columns. Use for term of columns are spiral_columns. Use for term of columns are stanchions. Part term of columns are bases_columns. Part term of columns are capitals_column. Part term of columns are plinths. Part term of columns are shafts_columns. Related term of columns are peristyles. Related term of columns are piers_bridges. Related term of columns are pilasters.

Fig. 2: Thesaurus representation.

So far it has only been demonstrated that most of the functionality of procedurally programmed electronic thesauri can be implemented by trivial one liner programs. Exploiting the possibilities a bit further enables the rationalisation of the knowledge base. Pairs BT/NT and PT/WT are by definition inverse. First the knowledge base must be checked for the built in inverse errors:

```
inverseError(X,Y):-
      hBT(X,Y),
      hNT(Y,X).        /* version of rNT similar to rBT*/
inverseError(X,Y):-
      hPT(X,Y),
```

```
hWT(Y,X).        /* version of rNT similar to rBT*/
```

Then all the rBT and hWT can be deleted and the knowledge on the inverse relation added to the database:

```
rBT(X,Y):-
      rNT(Y,X).
rWT(X,Y):-
      rPT(Y,X).
```

The raw size of the original clause form was nearly two megabytes. Using the knowledge on the relation, it could be reduced by 30%.

Some other interesting queries into the database were: what is the relation between two terms, what are the paths that connect the two terms. Which are the root terms - the terms on top of the hierarchies and which the leaf terms. If a set of connected terms is defined as a connected-term-graph, is the whole thesaurus one huge connected-term-graph or is it clustered into several smaller ones. What are the 'topics' of the smaller graphs if they exist. If there is only one graph, is there a set of bridges between more its sections. Can the sub-graphs or the sections of the graph provide data on how to cluster data (object, classes), on how to choose class categories or product model aspects.

Logic representation of NIAM Schema

NIAM diagrams (Leung and Nijjsen, 1988) are another example of a graph data structure. Similarly to the thesaurus example, we suggest the usage of Prolog atoms for the representation of data items and Prolog structures for the representation of the relations between data items.

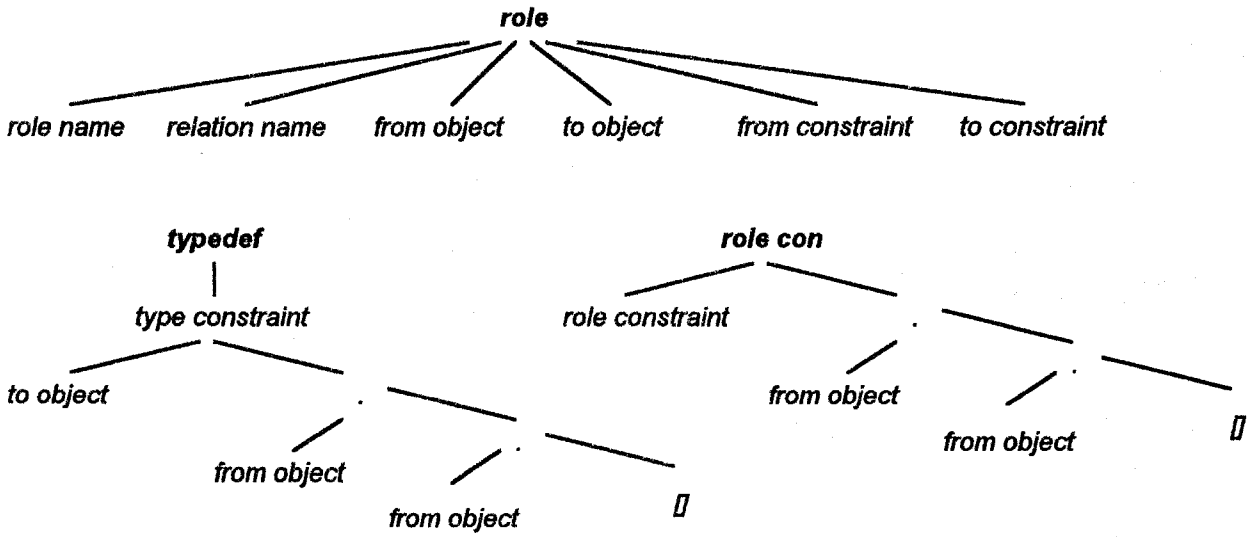

Fig. 3. Basic data structures for representing NIAM diagrams.

NIAM diagrams can be represented using three data structures (Fig. 3). Role structure ("role" in Fig.3) carries information about one relation between two objects (LOTs or NOLOTs). It may be named or unnamed. Basic role constraints may be encoded as well. The type definition

("typedef") defines sub typing and sub typing constraints between a NOLOT and a list of other NOLOTs. Finally, if we want to impose special constraints on roles ("role con"), the roles must be named. A constraint applies to a list of named roles. The full syntax for textual representation of a subset of NIAM diagrams is listed in the Appendix.

To simplify the translation of diagrams into textual form, Prolog's syntax has been extended to allow for a clearly readable English representation. The test case was the Building structural systems model (Turner and Wassim, 1990). In Fig. 4 there are two diagrams taken from that document and the corresponding Prolog representation is in Fig. 5.

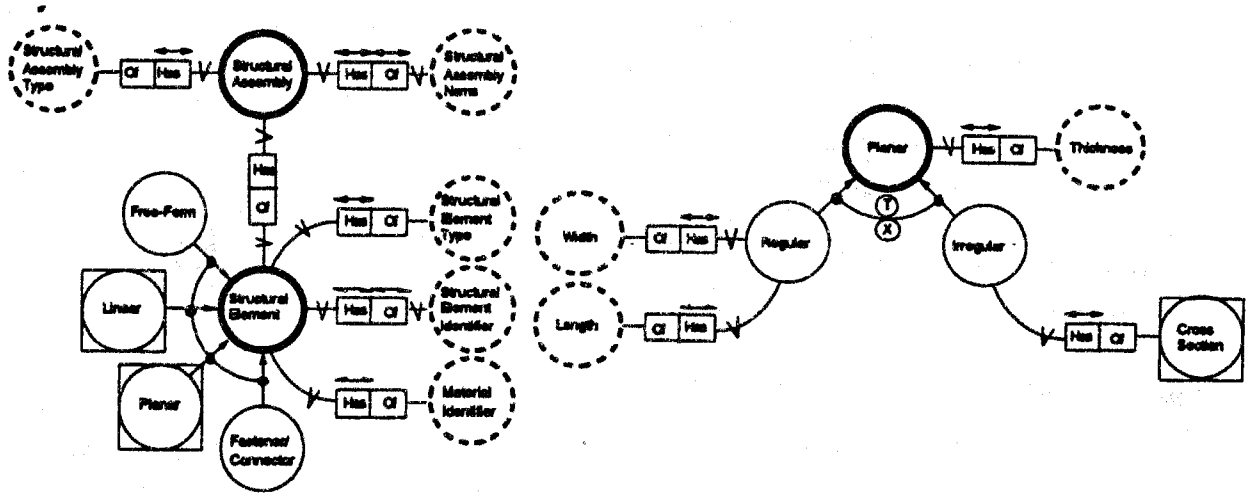

Fig. 4. NIAM diagrams for structural assembly and planar.

```
% structural assembly (6.)

role(each structuralAssembly has oneOrMore structuralElement).
role(each structuralAssembly has one structuralAssemblyName).
role(each structuralAssembly has one structuralAssemblyType).
role(each structuralElement has one structuralElementType).
role(each structuralElement has 1 exactly structuralElementIdetifier).
role(each structuralElement has one materialIdentifier).
typedef(structuralElement isOneOf [freeForm,linear,planar,fastenerConnector]).

% planar

role(each planar has one thickness).
typedef(planar isOneOf [regular,irregular]).
role(each regular has one width).
role(each regular has one length).
role(each iregular has one crossSection).
```

Fig. 5. Structural assembly and planar in Prolog.

Like in the thesaurus example, the semantic representation is very well suited for the validation, inquirement and processing of the schema information. It is very easy to discover cycles in typedef relations or inconsistencies of the role relations. A set of diagrams that was

translated into Prolog, can be viewed as a graph. The operations in the graph, like acyclic path finding, shortest path finding, hamiltonian path finding, finding if the graph is connected ... are all programs of the size of less than one page. To find how structural elements and length are related, it is enough to define the following predicates:

```
path(A,Z,P,PATH):-
     path1(A,[Z],P,PATH),
     writePath(PATH).

path1(A,[A|Path1],[A,Path1],[]).

path1(A,[Y|Path1],P,[ZZ|SHOW]):-
     connected(X,Y,ZZ),
     not member(X,Path1),
     path1(A,[X,Y|Path1],P,SHOW).

writePath([]).
writePath([X|Tail]):-
     writePath(Tail),
     write(X),nl.
```

and ask:

```
?- path(structuralElement,length,_,_).
```

to get:

```
[,structuralElement,isOneOf,,planar]
[,planar,isOneOf,,regular]
[each,regular,has,one,length]
yes
```

Combining models and thesauri

A compatible representation of the model information and the thesauri information opens new perspectives for the usage of both. If the same terms were used or synonyms defined, it is quite easy for one database to profit from the other. For instance, if there is a path up to three arcs long in the model database, we can consider two terms related:

```
related(X,Y):-
     path(X,Y,_,[_,_]);
     path(X,Y,_,[_]).
```

A part term can be defined to get information from the has roles:

```
rPT(X,Y):-
     role(_,has,Y,X,_,_).
```

In the near future, when there are plenty of computerised conceptual models available, we will be able to augment the semantic information in the thesauri with the semantic information encoded in the models. On the other hand, conceptual models can benefit from the semantic

information in the thesauri. Model designers should take care not to allow the following to be true:

```
thTest(X,Y):-
      hBT(X,Y),
      typedef(_(X,LIST)),
      member(Y,LIST).
```

If something is defined as a broad term in the thesaurus, it should not be defined as a subtype in the model.

CONCLUSIONS

Prolog language is the natural choice for the semantic representation of data items and relations between them. It is particularly well suited in areas where the ratio between the number of relations and number of data items is high. Conceptual models and thesauri are examples of such areas. Project management, CAD and business data are not.

Using semantic representation for any data used during the conceptual modelling makes possible or greatly simplifies automated validation, complex inquiries, automated schema generation and format translation of conceptual models. This is particularly important in areas with large complex models where some kind of formal correctness is essential. Compatible representation of various information enables synergetic effects on the knowledge present in each one of the representations and a possibility to reuse and combine the knowledge.

A successful attempt was made to implement the facts known about the concepts in a thesaurus and in a conceptual model as the first order predicate logic clauses. The Prolog language was used and was extended with some classification and NIAM schema specific predicates. Canadian construction thesaurus and the proposed building structural systems model was translated into that representation. It has been shown that the semantic representation enables more complicated querying of the concepts database and has richer semantic content which also enables automated schema verification and a proof of correctness. The representation is also open for the application in standards representation (classifier trees) and associative databases.

A major drawback in the logic representation is its user unfriendliness. Computing speed and RAM requirements may alsc be a problem for less capable Prolog systems and PC hardware. Since user interface design is not Prolog's strong point a client-server architecture between a Prolog engine and a hypertext user friendly shell will be investigated.

ACKNOWLEDGEMENT

The author wishes to thank Dana J. Vanier for the ASCII version of the Canadian Thesaurus of Construction Science and Technology and for creative comments to the presented work.

REFERENCES

Adeli H (1988). *Expert systems in construction and civil engineering*, Chapman and Hall, London.

Alim S, Munro J (1987). PROLOG-based expert systems in civil engineering, *Proceedings - Institution of Civil Engineers, Part 1, Design and construction*, Vol 83, 1-14.

Bjork B-C, Penttila H (1989). A scenario for the develoment and implementation of a building product model standard, *Advances in Engineering Software, Vol 11, No 4*, 176-187.

Bratko I (1986). *Prolog programming for artificial intelligence*, Addison Wesley, New York, USA.

Clark S N (1990). An introduction to the NIST PDES Toolkit, NISTIR 4336, NIST, Gaithersburg MD, USA.

Clocksin W F, Mellish C S (1981). *Programming in Prolog*, Springer-Verlag, Berlin, Germany.

Coyne R (1988). *Logic models of design*, Pitman Publishing, London, Great Britain.

D. Appleton Company (1988). *Information modeling manual IDEF-1 Extended*, D. Appleton Company Inc., report, USA.

Eastman C M. (1992). Personal discussion.

Fereshetian N, Eastman C M (1991). A comparison of information models for product design, *Proceedings of the CIB seminar The computer integrated future*, Eindhoven 16-17.9.1991, Eindhoven, Calibre, University of Technology, The Netherlands.

Koumousus V K, Georgiou P G (1991). An expert system for the design of industrial building steel roof trusses, in *Artifical intelligence in structural engineering*, Civ-Comp Press, Edinbourgh, 181-187.

Leung C M R, Nijssen G M (1988). Relational database design using the niam conceptual schema, *Inform. Systems Vol 13, No 2*, 219-227.

Obretenov D, Angelov Zh, Mihaylov J, Dishleava P, Kirova N (1988). A knowledge based approach to relational database design, *Data & Knowledge Engineering Vol 3, No 3.*

Rasdorf W J, Abudayyeh O Y (1992). A formal approach to schema design for engineering databases, *Adv Eng Soft 1992, 14*, 23-31.

Sowa J F (1984). *Conceptual Structures: Information Processing in Mind and Machine*, Addison Wesley, New York, USA.

Spinoza B de (1988). *Etika*, Filozofska knjiznica, Slovenska matica, Ljubljana, Slovenia (in Slovene).

Thomson J, Bird S, Thomson D, Marksjo B, Sharpe R (1988). Extending Prolog to provide better support for design code expert systems, *Microcomputers in Civil Engineering, Vol 3, No 2*, Jun, 93-109.

Topping B H V, Kumar B (1989). Knowledge representation and structural engineering design codes, *Engineering Applications of AI, Vol 2*, September.

Turk Z (1992). Object Oriented Modelling Techniques and Integrated CAD, to be published in *Proceedings of the CIB Joint Workshop on Computers in Construction*, Montreal, Canada, May 1992.

Turk Z, Vanier D (1993). Classification systems in object oriented modelling of buildings, *Proceedings of the DMMI'93*, Bled, Slovenia.

Turner J, Wassim J, (1990). *Building structual systems model*, PDES/STEP AEC Comittee working paper, Univ. of Michigan, Ann Arbor, Michigan, USA.

Vanier D J (1992). Canadian thesaurus of construction science and technology: A hypercard stack, to be published in *Proceedings of the CIB Joint Workshops on Computers in Construction*, Montreal, May 1992.

Watson A S, Chan S H (1989). A Prolog based object oriented engineering DBMS, in *Artifical intelligence techniques and applications for civil and structural engineering*, Civ-Comp Press, Edinbourgh, 37-48.

APPENDIX

niamFact:

role
roleCon
typedef

role:

role (*[roleName,] englishRole*) .
role (*[roleName,] relationName, objectName, objectName*) .
role (*[roleName,] relationName, objectName, objectName, ideaConstraint, ideaConstraint*) .

englishRole:

ideaConstraint objectName relationName ideaConstraint objectName

roleCon:

roleCon (*roleConstraint*, [*nameList*]) .

typedef:

typedef (*objectName typeConstraint* [*nameList*]) .

ideaConstraint:

each
oneOrMore
zeroOrMore
one

some
exactly (*parameterList*)

typeConstraint:
hasSubtypes
canBeOf
canBeOneOf
isOneOf
isManyOf

roleConstraint:
exclusion
totalExclusion
total
equality
uniqueness
subset

nameList:
name
name , *nameList*

parameterList:
constant
constant , *parameterList*

relationName:
name

objectName:
name

roleName:
name

name:
valid-Prolog-name-for-an-atom