# COMPUTATIONAL INTELLIGENCE FOR DECISION SUPPORT. CONCEPTUAL DATA AND KNOWLEDGE MODELING

**Delia UNGUREANU**
**Francisc SISAK**
Automatics Department, "Transilvania" University of Brasov, M.Viteazu street, no.5, 500174, Brasov, Romania, phone: +40 0268 418836
e-mail: delia@deltanet.ro, sisak@unitbv.ro

*The starting point of building agent-based systems is the conceptual modeling. There are different aspects for different kinds of conceptual modeling. At first we discuss in the paper conceptual data modeling with an emphasis on Entity-Relationship (ER) modeling and its relationship with object-oriented approaches. The other part of the paper is devoted to conceptual knowledge modeling, as well as a discussion on knowledge representation and reasoning. Two specific structured knowledge representation schemes are discussed, namely, frames and conceptual graphs. Since users are always an important factor in the integrated problem solving process involving all parties (including intelligent agents and human beings), we also included in this paper a brief discussion on the issue of user modeling.*

**Keywords:** computational intelligence, agent, knowledge, conceptual graphs.

## 1. INTRODUCTION

The entity-relationship (ER) approach provides an effective way for conceptual modeling of data. Data can be described in terms of *things* and their *connections*. There are two basic kinds of basic components in an ER model: *entity sets* consist of entities, as well as *relationship sets* connecting the entity sets.

Usually we represent ER using diagram (ERD). The ERD is the graphical expression of the logical structure of a database. The major constructs involved in ER modeling are:

- An *entity set* (E) - is a set of entities of the same type that share the same properties (or *attributes*). An entity is a thing or object in the real world that is distinguishable from all other things. It is represented by a set of attributes.
- A *relationship set* (R) - is a set of relationships of the same type. A relationship is an association among several entities.

A relationship set may have its own attributes (just like an entity set). An important aspect of a relationship set is the *mapping constraint* (namely, *cardinalities*): one to one (1:1); one to many (1:N); many to one (N:1); many to many (M:N). Both entity sets and relationship sets are described by *attributes*.

## 2. SOME IMPORTANT CONCEPTS

In relational databases, we have already learned concepts related to keys. These concepts can be extended to entity-relationship models.

- **Keys for entity sets**: There are several different types of keys for entity sets:
  - *Superkey*: It uniquely identifies an entity in the entity set.
  - *Candidate key*: It is the minimal super key (attributes used as candidate key are

usually underlined).

- *Primary key*: It is the designated candidate key.
- *Foreign key*: It is a set of attributes which form the primary key of another relation. **Keys for relationship sets**: Primary keys for relationship sets are formed from primary keys of associated entity sets.

- **Existence dependency**: If the existence of entity x depends on the existence of entity y, then: x is existence dependent on y; entity y is a dominant entity; entity x is a subordinate entity.
- **Total versus partial participation** of entity set E in relationship set R:
  - *Total participation*: every entity participates in at least one relationship.
  - *Partial participation*: only some entities participate in relationships.
- **Weak entity sets**: An entity set which does not have sufficient attributes to form a primary key.

The following is the general process of developing an ERD: (a) obtain data requirements; (b) entity sets designation; (c) relationship sets designation.

ERDs can be converted to a form closely related to predicate logic, which is a relation. The general steps needed for converting an ERD to tabular format are stated below. After the ERD is converted to the table format, relational database techniques can then be used.

- For strong entity set E: we represent E by a table with distinct columns; each column corresponds to one of the attributes of E. Each row corresponds to an entity of the entity set.
- For weak entity set A owned by strong entity set B: we represent it by a table with distinct columns; each column corresponds to one of the attributes of A or attributes of the primary key of B.
- For relationship set R (R does not link a weak entity set to its owner strong entity set): we represent it by a table with distinct columns; each column corresponds to one of the attributes in primary keys of associated entity sets or R's own descriptive attributes.
- For many-to-one relationship: for a N:1 relationship set R from entity set A to entity set B, if there is an existence dependency of A on B, combine the tables A & R.
- For multivalued attribute M: we create a table T with a column C that corresponds to M and columns corresponding to the primary key of the entity set or relationship set of which M is an attribute.

Important to relational database design is the concept of *key*.

*Primary keys*:

- Entity relation: The primary key of the entity set in ERD becomes the primary key of the entity relation.
- Relationship relation: The union of the primary keys of the related entity sets becomes a superkey of the relation.

*Foreign key*: An attribute in a relation is a primary key of another relation.

To illustrate ER modeling and its conversion to table format, we use representation for a banking enterprise (Fig.1a). Applying the conversion we obtain the tables shown in Fig.1b.

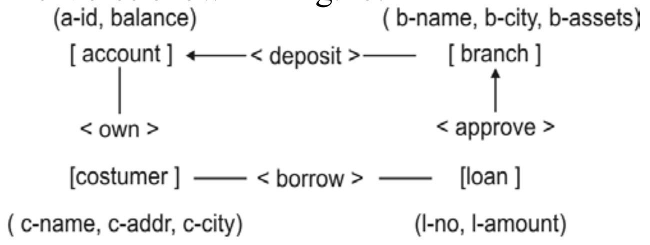
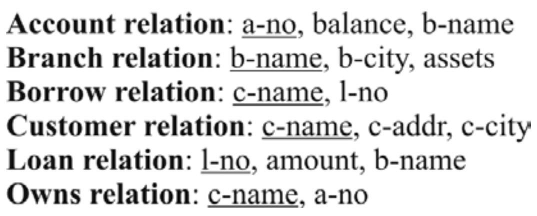


Figure 1a - The banking ER diagram



**Account relation**: <u>a-no</u>, balance, b-name
**Branch relation**: <u>b-name</u>, b-city, assets
**Borrow relation**: <u>c-name</u>, l-no
**Customer relation**: <u>c-name</u>, c-addr, c-city
**Loan relation**: <u>l-no</u>, amount, b-name
**Owns relation**: <u>c-name</u>, a-no

Figure 1b - The tables for the Banking

There are some well-known problems of ER modeling which have been used to lead to more advanced modeling techniques, including extended ER models and object-oriented models:

- *Specialization*-some entity sets should have a closer relationship than relationship with other entity sets.
- *Generalization*-common attributes of entities can be stored only in one entity set.
- *Aggregation*-representation of relationships among relationships. Conceptually, an object communicates with the rest of the system by sending messages to invoke various *methods*.

## 3. REMARK ON LEGACY DATA MODELS

Entity-Relationship model was introduced to unify three existing logical models: relational, network, and hierarchical. The network data model is the ER model with all relationships restricted to be binary, many-one relationships. Entity sets are represented directly by logical record types with attributes as their fields.

Binary, many-one relationships are kept; arbitrary relationships should be converted by creating new logical record types. We can use a simple directed graph model for data. Retrieving data requires users to write queries to express how the links are navigated. The hierarchical data model can be considered as a special case of the network model.

Queries in object-oriented data model may follow similar considerations as the network model, such as specifying navigation path.

Since object-oriented data model may be considered as the "current" data model, relational database systems sometimes are also considered as "legacy" systems. The building data warehouses largely depends on relational data modeling techniques.

There are some reasons of studying legacy data models, including the following:

- *Data re-engineering*: In some cases, we have to re-implement legacy systems.
- *Building data warehouses*: It has been increasingly popular to build data warehouses for decision support queries. The source data used for building a data warehouse may be acquired from legacy databases.
- *Object-oriented implementation*: The legacy systems may shed lights on learning object-oriented data modeling techniques.

## 4. KNOWLEDGE MODELING FOR KNOWLEDGE REPRESENTATION

Just like data modeling is concerned with how to conceptually view the data, knowledge modeling is concerned with how to conceptually view the knowledge to be represented. While relational model captures the logical representation of data, an ER diagram, which represents the schema of the corresponding relational model, can also be viewed as a primitive version of knowledge modeling. Knowledge modeling serves as an intermediate step for knowledge representation (KR) and reasoning, because it is preferable to have the contents of the knowledge captured before the exact format of representation is determined. Knowledge modeling approach for building knowledge-based systems may be carried out through two steps: (a) capture the contents of knowledge and (b) select the appropriate knowledge representation scheme and convert the captured knowledge into that scheme.

## 5. STRUCTURED KNOWLEDGE REPRESENTATION

Many factors contribute to complexity of knowledge representation. One is *granularity*. Often we represent knowledge in small pieces because is easy to revise them. The problem is that it lacks the "global" perspective.

In many applications, knowledge should be represented in a more structured manner. *Structured knowledge representation schemes* (such as associative networks, frames and conceptual graphs) have been developed for this purpose. The term "structured knowledge" is also called aggregated knowledge. The predicate expressions and productions rules themselves may also be called as "structured knowledge", mainly from researchers in neural network community, because they are more structured than subsymbolic representation, and can be obtained from machine learning algorithms applied on neural networks. Another issue is *nonmonotonic reasoning*. A specific problem in nonmonotonic reasoning is *the frame problem*, which is concerned with how to represent change. Truth maintenance systems have been developed to deal with nonmonotonic reasoning in knowledge-based systems. Their task is to check the validity of knowledge stored in the knowledge base.

There are several general issues of KR and structured schemes. Knowledge representation schemes serve the role of "languages" at the symbol level. Desired features of KR languages include the ability to do the following:

a. Handle qualitative knowledge;
b. Allow new knowledge to be inferred from a set of facts and rules;
c. Allow representation of general principles as well as specific situations;
d. Capture complex semantic meaning;
e. Allow for meta-level reasoning.

To represent structured knowledge, we can represent knowledge as a *graph*, with *nodes* corresponding to facts or concepts, and *arcs* corresponding to relations or associates between concepts. Using network representations, we view knowledge as organized using explicit links or associations between objects in KB. There have been many proposals, focusing on either structures or actions, but for a long time, there has been a lack of "standard" network representation. Recently, *conceptual graphs* became one candidate for this standardization.

## 6. FRAME SYSTEMS

A frame system may be viewed as an alternative to network representations. We discuss frame systems through two levels: first *individual frames*, and then *frame systems*. Frame contents include: an identifier; relationship with other frames; descriptors of requirements for frame match; procedural information; default information; new instance information (unspecified at beginning) and others. Frame systems provide a much clearer picture than semantic networks themselves. They add to the power of semantic nets by allowing complex objects to be represented as a single frame, rather than as a large network structure.

Parallel to the development of frame languages has been the development of object-oriented programming languages (OOPL). Frame systems and OOPLs are quite similar, differing primarily in emphasis.

In order to fully support the needs of object-oriented paradigm, a programming language should support capabilities of encapsulation, polymorphism and inheritance. We combine both data items and the methods into a single structure, called a class. A class is a set of object instances with shared features. Methods or procedures characterize the behavior of a class. A method is *polymorphic* if it has many different behaviors, depending on the types of its arguments. Associated with the concept of the class is *inheritance*, which is a mechanism for supporting class abstraction in a programming language, as well as in a knowledge representation scheme. A class can inherit properties from its *superclass(es)*; these properties, along with the properties owned by this class itself, can be inherited by its *subclasses*.

Inheritance allows a frame to inherit properties from its parent. Frames can be connected through class-subclass relationships to form a frame system. The inheritance is not mandatory; in fact, a subclass (or an instance) can override some of the properties of the class it belongs to. This is advantageous in many real-world situations. We may set up default values for inhetited properties, because these values may be overridden later. The overriding feature involved in inheritance gives frame systems the power of nonmonotonic reasoning. The Prolog, LISP, or OOPL like C++ are used for implementing a frame system.

## 7. CONCEPTUAL GRAPHS

We discuss *conceptual graphs* (CGs) in the context of *conceptual modeling*. A CG is a finite, connected, bipartite graph. The nodes of the graph are either concepts or conceptual relations; each node is connected to the other kind of node. The following are some basic definitions:

• *Concept node*: It is represented as a box. A concept can be either a concrete or abstract entity. A concept node connects to conceptual relations. A concept node may represent an individual or a type.

• *Conceptual relation*: It is represented as ellipses connecting concepts and plays the role of a labeled arc. Usually it connects two concept node (and is thus binary), but it could be an n-ary relation.

CGs do not use labeled arcs; instead the conceptual relation nodes represent relations between concepts. There is a connection between conceptual graphs (CGs) and ERDs. Concepts in CG resemble concepts in ERDs while conceptual relations resemble relationship sets. The CGs do not have the stored data associated with concepts and conceptual relation nodes.

We consider the following example: "John donated a computer to Transilvania University". The conceptual graph is shown in Fig.2a and his linear form in Fig.2b.

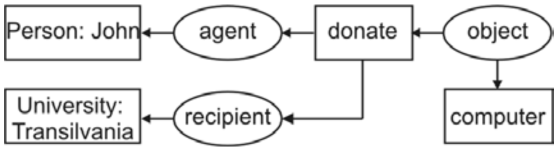


Figure 2a - A conceptual graph

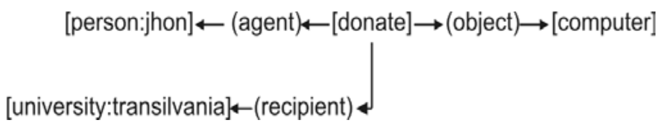

[person:jhon]← (agent)←[donate]→(object)→[computer]
[university:transilvania]←(recipient)

Figure 2b – A conceptual graph in linear form

Conceptual graphs can be constructed by organizing concepts and conceptual relations around some "key" activities (which are usually verbs). The following operations defined on CGs can be viewed as rules:

- *Simplify*: It allows deletion of duplicated relations.
- *Copy*: This is the operation for duplicating a CG.
- *Restriction*: It allows concept nodes in a graph to be replaced by a node representing their specialization.
- *Join*: It combines two CGs sharing some concept node into one. If we consider the CG: [ibm PC]→(price)→[1000Eu]; the result after join is shown in Fig.3.

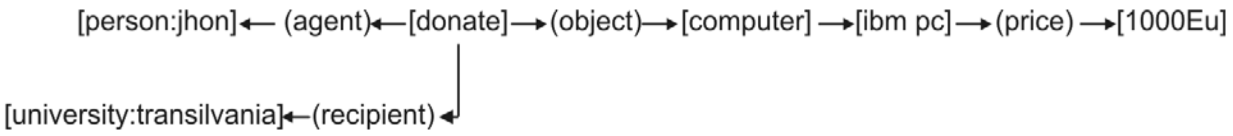

[person:jhon]← (agent)←[donate]→(object)→[computer] →[ibm pc]→(price) →[1000Eu]
[university:transilvania]←(recipient)

Figure 3 - A conceptual graph after join

Conceptual graphs employ *propositional nodes* to handle issues in logic. A propositional node is a kind of high level node - that is, a node itself may be a graph. For example, consider the statement "Andrew knows that John donated a computer to Transilvania University". Here "knows" is a relation that takes a proposition as an argument. The new conceptual graph can be represented like in Fig.4.

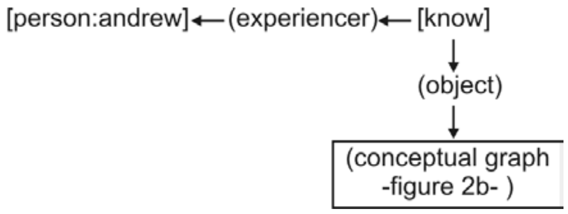

[person:andrew]←(experiencer)← [know]
(object)
(conceptual graph -figure 2b- )

Figure 4-A conceptual graph with propositional node

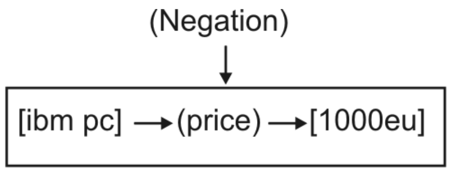


Figure 5-Propositional node with negation

- *Negation*. Propositional nodes allow us to express negation. (see Fig.5)
- *Quantifier*. Conceptual graphs do not explicitly use quantifiers either. Since negation can be handled by using propositional nodes, by utilizing deMorgan's law we should be able to express variables which are universally quantified.
- *Modal logic*. Furthermore, conceptual graphs with propositional nodes may be used to express the modal concepts of knowledge and belief.

The inference rules are shown below:

- *Erasure*. Any conceptual graph enclosed by an even number of negations may be erased. Suppose w is a conceptual graph, it can be erased from the consequent of an implication ¬[u ¬ [v]] to derive . ¬[u w ¬ [v]]
- *Insertion*. Any conceptual graph may be inserted into another graph context which is enclosed by an odd number of negations. Suppose w is a conceptual graph, it can be inserted into the consequent of an implication ¬[u ¬[v w]] to derive ¬[u ¬ [v]].
- *Iteration*. A copy of any conceptual graph C may be inserted into a graph context in which C occurs or in which C is dominated by another concept.
- *De-iteration*. Any conceptual graph which could be the result of iteration may be erased from a conceptual graph context.
- *Double negation*. A double negation may be erased or drawn before any conceptual graph or set of graphs.

Conceptual graphs are equivalent to predicate calculus. There is a straightforward mapping from conceptual graphs into predicate calculus notation. The following is an algorithm for *converting a conceptual graph g into a predicate calculus expression*:

1. Assign a unique variable to each of the n generic concepts in g.
2. Assign a unique constant to each individual concept in g.
3. Represent each concept node by a unary predicate with the same name as the type of that node and with the variable or constant assigned in step 1 or 2 as its argument.
4. Represent each n-ary conceptual relation in g as an n-ary predicate whose name is the same as the relation.
5. Take the conjunction of all atomic sentences formed in steps 3 and 4. Attach an existential quantifier to each variable.

For example, following the 5 steps, the sample conceptual graph converted to logic is:

$$\exists X\ \exists Y\ (donate(X) \wedge computer(Y) \wedge person(john) \wedge university(transilvania) \wedge agent(X, john) \wedge object(X,Y) \wedge recipient(X, transilvania))$$

## 8. USER MODELING AND FLEXIBLE INFERENCE CONTROL

Intelligent agents and knowledge workers share the same problem-solving environment which is modeled by using the discussed techniques. Knowledge workers and other human users are also an integrated part of this environment. We can look on *user modeling* from the perspective of computer-user symbiosis - a desirable environment for decision support. The main consideration is to provide flexible inference control to reduce the work of search in knowledge bases (KB).

There are some common considerations behind real-time expert systems. In order to provide timely response, most approaches have imposed a restricted manner of searching, namely, to restrict the portion of KB to be searched. In the *designated inference engine approach*, each inference engine only processes a particular kind of data, so that only part of the KB will be searched by a particular inference engine. Some other approaches rely on meta-level reasoning and change of focus in searching. To assure flexibility, some degree of domain-dependent control has also been introduced. There are four items - user environments, user modeling, system adaptability and perturbation models - that are all concerned with the role of users in flexible inference control, but each has its unique focus. Flexible inference control is also justified from the perspective of different user groups of user models. This discussion leads to the issue of *system adaptability*.

Another interesting concept is the *perturbation model*. According to the theory of perturbation models, each user is assumed to have a mental model similar to the domain model (the "true" model of the system), differing only in certain perturbations to the domain model. To support such kind of system adaptability, flexible inference control is needed. A more radical approach of flexible inference control requires the user's participation of reasoning.

## 9. CONCLUSIONS

The frame systems are more appropriate in representing structures while conceptual graphs are more appropriate in representing actions (as well as in natural language representation). The knowledge represented by frame systems may also be represented by a conceptual graph, but the frame system representation may be more appropriate. We see a similarity between the object-oriented formalism and the conceptual graph formalism and also we remark a relationship between the notion of class in object orientation and the notion of type in conceptual graphs.

## 10. RESULTS

We tried solving some computational intelligence problems using different programming languages. We transposed representation in Prolog to C++. Prolog can be considered as an implementation of logic as a programming language. It has a declarative reading as well as a procedural reading and class libraries that simulate Prolog can quickly be built. C++ is not as directly suited to symbolic computation, but is an OOPL and computational intelligence programs can be written around the data types. A strong motivation for using C++ for computational intelligence programs is its suitability for use in large software systems.

## 11. REFERENCES

[1] Luger, G. F. and Stubblefield, W. A., *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (3rd ed.), Addison-Wesley Longman, Harlow, England, 1998.

[2] Silberschatz, A., Korth, H. F. and Sudarshan, S., *Database System Concepts* (3rd ed.), McGraw Hill, New York, 1998.

[3] Lockwood, S. and Chen, Z., *Modeling experts' decision making using knowledge charts*, Information and Decision Technologies, 19, 311-319, 1994.

[4] Chen, Z., *User modeling for flexible inference control and its relevance to decision making in economics and management*, Computational Economics, 6, 163-175, 1993.

[5] Ford, K. M., Bradshaw, J. M., *Introduction: Knowledge acquisition as modeling,* international Journal of Intelligent Systems, 8(1), 1-7, 1993.

[6] Rich, E., Knight, K., *Artificial Intelligence* (2nd ed.), McGraw-Hill, New York, 1991.