IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4396

A Methodology for Conceptual Database Design from Natural Language Specifications

Aryya Gangopadhyay, Department of Information Systems
University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250
Phone: (410)455-2620, Fax: (410)455-1073, gangopad@umbc.edu

ABSTRACT

In this paper we describe a structured method for developing a conceptual data model by starting from a functional model expressed in a natural language. We have used the Conceptual Dependency theory for mapping natural language descriptions to conceptual dependency diagrams. We have developed algorithms to convert these conceptual dependency diagrams into unit conceptual dependency tables, which are then merged to represent the whole context of the application. We also show how transactional requirements can be incorporated into the unit conceptual dependency table, and subsequently convert the unit conceptual dependency table into a corresponding conceptual model.

1 INTRODUCTION

The process of designing a database for an enterprise can be divided into conceptual, logical, and physical design phases [14]. The conceptual design phase consists of two steps: view modeling, and view integration. In the view-modeling phase, each user group analyzes its data requirements and expresses them in terms of a local conceptual schema. In the view integration phase, these independently developed views are integrated into an overall enterprise-wide schema.

There are many methods for determining user requirements: interviewing users, studying existing environments, and analyzing functional specifications of the system [26]. In all of these cases, descriptions in natural languages are common and abundant. Interviews are held in natural languages and transcribed in natural language reports, existing environments can be described in natural language descriptions, and functional specifications have natural language definitions and glossary of terms. Developing the conceptual model involves identifying the various data elements from natural language descriptions, and determining their inter-relationships [11,9,12,13]. No systematic method has been suggested in the literature to identify missing pieces of information in the description of user requirements.

In this paper we develop a methodology for designing the conceptual model starting from a functional model expressed in natural language sentences, using the design specification from [17,18,20,22]. We applied the theory of Conceptual Dependencies, developed by Schank [28,27] to interpret natural language expressions. The theory of conceptual dependencies provides a method for developing conceptual representations of natural language sentences and has primarily been used for developing natural language understanding and generating systems. The methodology we describe identifies data elements from the functional model expressed as natural language description, locate missing pieces of information, combine the individual data elements into an overall conceptual schema, and establish object granularity in the data model.

2 METHODOLOGY

In this section we describe the methodology of developing the conceptual schema from natural language specification of functional requirements. We first describe the target conceptual schema to be developed, next we describe the theory of conceptual dependencies, lastly we describe the steps necessary to develop the conceptual schema from the conceptual dependency diagrams (CD).

2.1 Steps to develop conceptual schema from conceptual dependency diagrams

Developing the conceptual schema from conceptual dependency diagrams consists of four steps. Each of these steps is described briefly below.

· Step 1: The first step in our method is to develop conceptual dependency diagrams from natural language specifications. This step is not shown here for space limitations, but is described in [3].

· Step 2: The CDs developed from Step 1 are then decomposed into unit CDs (UCD) where each UCD is made up of an arc and its associated nodes. Each UCD is represented as a tuple in a table (referred to as a UCT). Thus, for each primitive activity there are four UCTs, corresponding to the four arcs associated with it. The algorithm for developing a UCT from a given CD is described in 2.1.1.

· Step 3: The next step is integration or merging of the UCTs. The UCTs corresponding to the arcs of a primitive activity are merged to form that of the primitive activity. Those corresponding to primitive activities are merged to form the UCTs of higher level activities. Since the activities in many functional models are arranged in a hierarchical manner, by successive integration of the UCTs of the child activities, we obtain those of the parent activities. This continues until the UCT corresponding to the root level activity is obtained. An algorithm for integrating UCTs is presented in Section 2.1.2. The UCT is freed from redundancies by merging equivalent concepts. Rules for establishing equivalence among concepts are discussed in Section 2.1.2.

· Step 4: At this point, we convert the UCT to an equivalent ER diagram, which gives us the conceptual model for the entire application. Note that this conversion can be done at any intermediate level as well. Thus, an activity view (ER diagram corresponding to an activity) can be obtained by converting the UCT of that activity to the equivalent ERD. An algorithm for converting a UCT into an ER diagram is presented in Section 2.1.3.

2.1.1 Developing a Unit Conceptual Dependency Table

In this Section, we address Step 2 of our proposed method-

This paper appears in *Managing Information Technology in a Global Economy*, the proceedings of the Information Resources Management Association International Conference.

ology: developing a Unit Conceptual Dependency Table (UCT) from a number of input CDs. In order to merge the CDs corresponding to each arc of the activities in the functional model, we represent each CD in a canonical form, called a Unit CD Table (UCT). A UCT is a table where each tuple represent a Unit CD.

Definition 2.1.1.1 A Unit CD (UCD) is a CD consisting of an arc and its associated nodes.

Definition 2.1.1.2 A UCT is the canonical form of a set of UCDs represented as a 4-tuple *<Arc Type Node1 Node2>* if the arc in the UCD is binary, and a 5-tuple *<Arc Type Node1 Node2 Node3* if the arc in the UCD is a ternary arc.

We divide a CD into two parts: nodes and arcs. A node is a concept in the conceptual dependency theory [27,28]. Thus, PPs, ACTs, PAs, and AAs represent nodes. Arcs include the conceptual dependencies, conceptual cases, and conceptual relations discussed in section 2.2. In order to facilitate our discussion, we classify the various types of arcs described in the CD theory as follows:

1. **Binary Arc**: A binary arc connects two concepts, or two conceptualizations. We distinguish among the following types of binary arcs.

a. **A-arc**. Is a binary arc that connects a PP with a PA, or an ACT with an AA.

b. **C-arc**. Is a binary arc that connects two conceptualizations, such that one is causing the other.

c. **Cl-arc**. Is a binary arc that connects a conceptualization with a PP, such that the PP specifies the location of the conceptualization.

d. **Ct-arc**. Is a binary arc that connects a conceptualization with a PP, such that the PP specifies the time of occurrence of the conceptualization.

e. **I-arc**. Is a binary arc that connects a PP with an M-arc (see below). We notice that in the case of the I-arc, one of the nodes of the arc is itself an arc.

f. **M-arc**. Is a binary arc that connects a PP with an ACT.

g. **L-arc**. Is a binary arc connecting two PPs such that there is a locational dependency between the two PPs.

h. **O-arc**. Is a binary arc connecting a PP with an ACT such that the PP is an objective case of the ACT.

i. **P-arc**. Is a binary arc connecting two PPs such that there is a possessive dependency between the two PPs.

2. **Ternary arc**. Is one that connects three concepts. Examples of ternary arcs are those used in the directive and recipient cases (see discussion in Section 2.2). The State-change arc is also a ternary arc. We distinguish among the following types of ternary arcs.

a. **D-arc**. Is a ternary arc joining two directive PPs with an objective PP.

b. **R-arc**. Is a ternary arc joining the source, recipient, and objective PPs.

c. **S-arc**. Is a ternary arc that describes the state change of a PP, and connects the PP to its initial and final states.

The various arc types, graphical representation, and their description in the CD theory are listed in Figure 1. In order to represent a CD in its canonical form, we first decompose the CD into units CDs. Each of these unit CDs are then represented as a tuple of a UCT.

Now we describe how to derive a unit conceptual table from a conceptual dependency diagram (CD). We first describe the procedure algorithmically and then illustrate the algorithm with an example.

Algorithm 2.1.1: decomposing conceptual dependency diagrams

Input: A CD.
Output: UCT
Procedure:
begin

1. For each C-arc in the CD:

a. Detach the nodes (from the CD) that were connected by the arc.

b. Repeat Step a to the (sub) CDs that evolve from the previous step, until no C-arc exists.

2. Eliminate all I-arcs.
3. Represent all Cl arcs as binary arcs, attaching the ACT with the PP indicating the location of the ACT.
4. For all Ct-arcs, represent the PP indicating the time of occurrence as the AA of the ACT.
5. A Ct-arc represents the time of occurrence of an ACT, which is a modifier of the ACT.
6. For each arc in the CDs obtained from the previous steps:

a. Identify the nodes connected by the arc.

b. Detach the arc and its associated nodes from the CD.

c. Construct a corresponding tuple in the UCT.

end

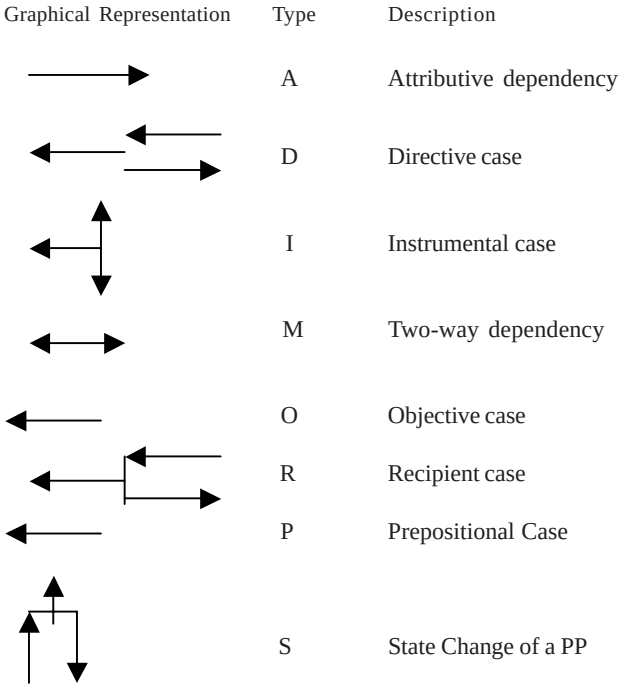

Figure 1: Classification of Arc Types

Arc	Type	Node1	Node2	Node3
(a1)	M	PRODUCE	Employee	
(a2)	M	OPERATE	Employee	
(c)	S	Machine	temp=x	temp=x+t
(d1)	O	PRODUCE	Product	
(d2)	O	OPERATE	Machine	
(e)	D	Product	Shop-floor	FG inventory
(f)	R	Raw material	Employee	Vendor

Table 1. An example Unit CD

2.1.2 Merging Unit Conceptual Dependency Tables

This Section addresses Step 3 of our proposed methodology. The UCTs obtained from the previous step are merged into an integrated UCT. This step is repeatedly applied to the UCTs corresponding to the arcs and lower level activities to obtain that corresponding to higher-level activities. When merging two or more UCTs we need to compare concepts and establish equivalence. This is accomplished by applying the following rules. The equivalence rules have been adapted from [33].

Rules 2.1.2.1: Merging PPs

1. If two PPs A and B have the same domain then they are equivalent.
2. If two PPs A and B have different domains, then

a. If the domain of *B* contains the domain of *A*, then add a containment dependency (the default relationship being ISA) between them. Also, all attributes of B are inherited by A.

b. If the domains of *A* and *B* are overlapping, then create a third PP, *AB*, such that the domain of *AB* is the union of those of *A* and *B*, and the attributes of *AB* are the intersection of the attributes of *A* and *B*. Add two containment dependencies: between *AB* and *A*, and between *AB* and *B*.

c. If the domains of *A* and *B* are disjoint and they have no common attributes, then they are kept separately. If *A* and *B* have some common attributes, they can be merged using the rules for merging PPs with overlapping domains, or kept separate, depending on the object granularity intended for the database. As an example, if *A* is *machine operator* and *B* is *sales manager*, then since *A* and *B* have common attributes (that any employee may have), a third PP Employee can be created, since the domain of Employee is the union of those of machine operator and sales manager.

Rules 2.1.2.2: Merging ACTs

1. Two ACTs connecting equivalent concepts are equivalent if they have only *nonclashing* attributes.

Definition: A *nonclashing* attribute is one that can be present in both ACTs.

2. If the concepts connected by two ACTs are not equivalent, the conflicting nodes are resolved using the rules for merging PPs.
3. If the conflicting nodes are resolved to be equivalent, then test the ACTs for equivalence using rule 1 for merging ACTs.
4. If the conflicting nodes are generalized to a higher class, then replace the PPs of the ACT with the new generalized class, and apply rule 1for testing the equivalence of the ACTs. As an example, let the ACT be "register", and the connected PPs are "undergraduate-student" and "course" in one case, and "graduate-student" and "course" in the other. If the object granularity is to be maintained at the level of "student", then the two ACTs can be merged into one, with the PPs "student" and "course" connected by it. Note that two specialization edges can be created between the PP "student" and the PPs "undergraduate-student" and graduate-student".

Algorithm 2.1.2: CD Merging

Input: Two UCTs (T_1, T_2).
Output: A new UCT, resulting from the merger of the UCTs contained in T_1 and T_2.
Procedure:
begin
For each row *i* in T_1
For each row *j* in T_2
a. **For** each node *k*, node *l* such that k Î *i* and *l*Î*j*
b. **if** node *k*, node *l* are both PPs, then compare them using rules 2.1.2.1
c. **else if** node *k*, node *l* are both ACTs, then compare them using rules 2.1.2.2
d. **If** for all nodes in row *i* there is an equivalent node in row *j* and *vice versa* then merge them into one
e. **Else** append row *i* to T_2
End if
End for
End for
end

EXAMPLE

We use table 1 and 2 to illustrate algorithm 2.3.2. The PPs "Filling line" and "Machine" are merged to "Machine", and "Raw material" and "Material" merged to "Material". No row in table1 is equivalent to any row in table 2, so we append the rows of table 1 to table2. The output is shown in table 3.

Arc	Type	Node1	Node2	Node3
(a)	M	MOVE	Employee	
(b)	O	MOVE	Material	
(c)	D	Material	Filling line	Inventory

Table 2. Unit conceptual dependency table

Arc	Type	Node1	Node2	Node3
(a)	M	MOVE	Employee	
(b)	O	MOVE	Material	
(c)	D	Material	Machine	Inventory
(a1)	M	Produce	Employee	
(a2)	M	OPERATE	Employee	
(c)	S	Machine	temp=x	temp=x+t
(d1)	O	PRODUCE	Product	
(d2)	O	OPERATE	Machine	
(e)	D	Product	Shop-floor	FG inventory
(f)	R	Material	Employee	Vendor

Table 3. Merged table

2.1.3 Conversion of a Unit Conceptual Dependency Table to the Conceptual Schema

This Section addresses Step 4 of our proposed methodology. As in the previous sections, we first describe the algorithm and then illustrate it with an example.

Algorithm 2.1.3: Converting a UCT to a conceptual schema
Input: A UCT.
Output: An ER diagram corresponding to the UCT.
Procedure:
begin
For each row in the UCT:
1. If the node is a PP, represent it as an entity.
2. If the node is an ACT, represent it as a relationship.
3. If the node is a PA or an AA, represent it as an attribute.
4. If the arc-type is A, represent the dependent concept as an attribute of the governor concept.
5. **If** the arc-type is D, then there are four associated PPs:

source, destination, object, and actor.

a. If p_i, p_j, p_k | (p_i, p_j, p_k ® p_l), where p_i, p_j, p_k are the four associated PPs and i ¹ j ¹ k ¹ l, then create a 4-way relationship, connecting the four PPs by the ACT.

b. If the above condition cannot be met, then if, p_i, p_j | (p_i, p_j ® p_k), where p_i, p_j, p_k are any of the three PPs object, source, and destination, and i ¹ j ¹ k, then create a 3-way relationship, connecting the three PPs by the ACT. The actor is connected by a binary relationship with the object.

c. If the above conditions are not satisfied, then create three binary relationships: (actor, object), (object, source), (object, destination).

End if

6. **If** the arc-type is R (or Cl), then there are three associated PPs: sender, recipient, object (actor, object, location).

a. If exists p_i, p_j | (p_i, p_j ® p_k), where p_i, p_j, p_k are any of the three PPs sender, recipient, object (or actor, object, location), and i ¹ j ¹ k, then create a 3-way relationship, connecting the three PPs by the ACT.

b. If the above conditions are not satisfied, then create three binary relationships: (p_i, p_j), (p_i, p_k), and (p_j, p_k).

End if

7. If the arc-type is M, or O, connect the PPs through the ACT.

8. If the arc-type is P, connect the two PPs by a relationship "location", if the dependency is locative, or "possessed-by", if the dependency is possessive.

9. If the arc-type is S, represent the state as an attribute of the PP, and the values of the state as values of the attribute.

End for

end

Example

The ER diagram corresponding to the UCDs in Table 3 is shown in Figure 2, and is obtained by applying algorithm 2.1.3.

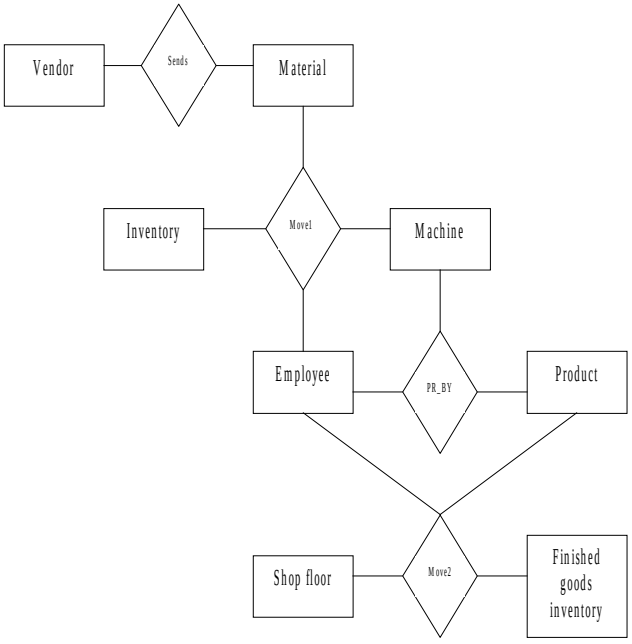

Figure 2. Conceptual model for UCT shown in Table 3.

3. CONCLUSIONS

In this paper we have described a methodology for developing a conceptual data model from functional specifications expressed in natural languages. The importance of developing such a methodology has been emphasized in the literature time and again. The methodology utilizes the theory of conceptual dependencies that has been applied to multiple natural language understanding systems. In addition to identifying the various data items from natural language specifications, the methodology described in this paper also includes integrating individual conceptual dependency diagrams into an overall schema for the application context. A prototype system has been developed that can take a natural language description as input and develops a conceptual schema as output.

The methodology described in this paper can also be useful in schema/view integration in OLTP applications in relational database platforms as well as OLAP and data warehouse designs. Other areas of application include software engineering, machine translation of natural language specifications, natural language interfaces to database applications, and data modeling and representation.

REFERENCES

1. Adam, N. and Gangopadhyay, A. "A Form-Based Natural Language Front-End to a CIM Database", *IEEE Transactions on Knowledge and Data Engineering*, 9(2): 238-250, March/April 1997.

2. Adam, N., Gangopadhyay, A., Clifford, J. "A Form-Based Approach to Natural Language Processing", *Journal of Management Information Systems*, 11(2): 109-135, Fall 1994

3. Adam, N. and Gangopadhyay, A. "An N-ary View Integration Method Using Conceptual Dependencies", in proceedings of the *Hawaii International Conference on System Sciences*, January 1995.

4. Batini, C. and Lenzerini, M. "A Methodology for Data Schema Integration in the Entity Relationship Model," *IEEE Transactions on Software Engineering*, SE-10(4):450-313, 1984.

5. Batini, C. and Lenzerini, M. and Navathe, S. B. "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*,18(4), 1987.

6. Bouzeghoub, M. and Gardarin, G. and Metais, E. "Database Design Tools: An Expert System Approach," in *Proceedings of the International Conference on Very Large Databases*, 1985.

7. Bouzeghoub, M. and Comyn-Wattiau, I. "View Integration by Semantic Unification and Transformation of Data Structures," in *Entity-Relationship Approach: The Core of Conceptual Modeling* Kangassalo, H. (editor), 381-398, 1991.

8. Buneman, P. and Davidson, S. and Kosky, A. "Theoretical Aspects of Schema Merging," in *Lecture Notes in Computer Science* 580. Springer Verlag 1992.

9. Carswell, J. L. and Navathe, S. B. "SA-ER: A Methodology that Links Structured Analysis and Entity-Relationship Modeling for Database Design," in *5th International Conference on ER Approach*, 1987.

10. Chen, P. "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems*, 1(1), 1974.

11. Chen, P. "From Ancient Egyptian Language to Future Conceptual Modeling," in *Conceptual Modeling Current Issues and Future Directions*, *Lecture Notes in Computer Science* 1545, 56-64, 1999.

12. Chen, P. "English Sentence Structure and Entity-Relationship Diagrams," *Information Sciences*, 29:97-149, 1983.

13. Chen, P. P. "English, Chinese, and ER Diagram," *Data and Knowledge Engineering*, 23(1): 5-14, 1997.

14. Elmasri, R. and Navathe, S. B. *Fundamentals of Database Systems*, Benjamin/Cummings Publishing Company, Inc., (3rd Edition) 2000.

15. Feng, J. and Crowe, M. "The Notion of 'Classes of a Path' in ER Schema," in *Advances in Databases and Information Systems*, *Lecture Notes in Computer Science* 1491, 218-232, 1999.

16. Geller, J. and Mehta, A. and Perl, Y. and Neuhold, E. and Sheth, A. "Algorithms for Structural; Schema Integration," in Ng, P. and Ramamoorthy, C. V. and Seifert, L. C. and Yeh, R. T. editors, *Proceedings of the Second International Conference on Systems Integration*, 274-284, 1992.

17. Hatchman, S. "Practitioner Perceptions On The Use of Some Semantic Concepts in the Entity-relationship model," *European Journal of Information Systems*, 4, 31-27, 1995.

18. Kesh, S. "Evaluating the Quality of Entity Relationship Models," *Information and Software Technology*, 25(9), 1995.

19. Lanka, S. "Automatically Inferring Database Schemas," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 317-319, 1985.

20. Maier, R. "Evaluation of Data Modeling," in *Advances in Databases and Information Systems*, *Lecture Notes in Computer Science* 1491, 232-245, 1999.

21. Malhotra, R. and Jayaraman, S. "An Integrated Framework for Enterprise Modeling," *Journal of Manufacturing Systems*, 11(4): 424-311, 1992.

22. Moody, D. L. "Metrics for Evaluating the Quality of Entity Relationship Models," in *Conceptual Modeling-ER '98*, *Lecture Notes in Computer Science* 1507, 211-225, 1998.

23. Navathe, S. and Elmasri, R. and Larson, J., "Integrating User Views in Database Design," *IEEE Computer*, Vol. 19, 50-42, 1984.

24. Navathe, S. and Savasree, A. "A Practical Schema Integration Facility using an Object-Oriented Approach," *Journal of Computers and Software Engineering*, 3:4, 1994.

25. Pernul, G. and Tjoa, A. M. "A View Integration Approach to the Design of Multilevel Secure Databases," in *Proceedings of the 10th International Conference on Entity Relationship Approach*, 1991.

26. Rosenthal, A. and Reiner, D. "Database Design Tools: Combining Theory, Guesswork, and User Interaction," in Lochovsky, F. H. editor, *Entity-Relationship Approach to Database Design and Querying*, North-Holland, New York, 1989.

27. Schank, R. C. "Conceptualizations Underlying Natural Language," in Schank, R. C. and Colby, K. M. editors, *Computer Models of Thought and Language*, 1973.

28. Schank, R. C. "Conceptual Dependency Theory," in *Conceptual Information Processing*, North-Holland/American Elsevier, 22-82, 1975.

29. Shanks, G. G. "Conceptual Data Modeling: An Empirical Study of Expert and Novice Data Modellers," *Australian Journal of Information Systems*, 4:2, 43-73, 1997.

30. Spencer, R. and Teorey, T. and Hevia, E. "ER Standards Proposal," in Kangassalo, H. editor, *Entity-Relationship Approach: The Core of Conceptual Modeling*, North-Holland, New York, 425-432, 1991.

31. Wand, Y, Storey, V. C., and Weber, R. An Ontological Analysis of the Relationship Construct in Conceptual Modeling. *ACM Transactions on Database Systems* 24(4): 494-520, 1999.

32. Yao, S. B. and Waddle, V. and Housel, B. "View Modeling and Integration Using the Functional Data Model," *IEEE Transactions on Software Engineering*, SE-8(4): 531-553, 1982.

33. Young, R. E. and Vesterager, J. "An Approach to CIM System Development whereby Manufacturing People can Design and build their own CIM Systems," *International Journal of Computer Integrated Manufacturing*, 4(5): 208-299, 1991.