1995120957

N95-27378

47253

Automated Database Design from Natural Language Input* p. 16

Fernando Gomez
Department of Computer Science, University of Central Florida
Orlando, FL 32816
gomez@cs.ucf.edu (407) 823-2764

Carlos Segami
Dept of Mathematics and Computer Science, Barry University
Miami Shores, FL 33161

Carl Delaune
NASA Kennedy Space Center

Abstract

Users and programmers of small systems typically do not have the skills needed to design a database schema from an English description of a problem. This paper describes a system that automatically designs databases for such small applications from English descriptions provided by end-users. Although the system has been motivated by the space applications at Kennedy Space Center, and portions of it have been designed with that idea in mind, it can be applied to different situations. The system consists of two major components: a natural language understander and a problem-solver. The paper describes briefly the knowledge representation structures constructed by the natural language understander, and, then, explains the problem-solver in detail.

*This research is being funded by NASA-KSC Contract NAG-10-0120

1 Introduction

In this paper, we describe a system that constructs logical database designs from English sentences entered by users with no knowledge of databases or programming. The logical design used is the entity-relationship model (E-R) (Chen, 1976). The set of user's statements describing a database has been called a *user view* (Navathe and Elmasri, 1986). The techniques for extracting user's views or the relevant components of a logical database from a user are based on elicitation methods. Several methodologies have been developed for aiding the extraction process (Baldissera et al., 1979; Martin, 1981; Ceri, 1983; Albano et al., 1985). More recently, expert systems techniques have been applied to the creation of an E-R model from user's specifications. The VCS system (Storey, 1988) elicits the entities, attributes and relations from the user by asking him/her questions formulated in English. In VCS, the user's replies

are limited to saying "yes/no" and listing the entities, attributes and relations separated by blanks. The approach presented in this paper, however, aims at identifying the entities, attributes and relations from the English descriptions of a problem. For instance, a typical database problem for which our system can build a logical design is the following:

> Each person keeps a record of documents of interest. The source and the time of each document are stored with the location of the document. Documents may be books, identified by author name and title. Documents may be also journal articles, identified by journal volume number, author name and title. Documents may be private correspondence, identified by sender and date.

Our system will identify the entities, attributes, key attributes and relations in this passage, and the hierarchical relations between the entity "document" and its subconcepts "book," "journal article," and "private correspondence." The two main components of our system are a natural language understander (NLU) and a problem-solver. Although research on using natural language processing (NLP) for interfacing databases has been intensive and has achieved certain success (Ballard and Tinkham, 1984; Grosz et al., 1987; Bates et al., 1986), research on the construction of logical databases from natural language has been scarce, but the reader may see (Kersten, 1987; Alshawi, 1985). These earlier attempts are based on syntax. Our approach, however, involves a parse of the sentence, a semantic interpretation of the output produced by the parser, the construction of knowledge representation structures from the logical forms of the sentence, and the integration of these structures into memory. Figure 1 depicts the main components of the natural language understander module. This model of comprehension of expository texts has been under development for some years now (Gomez, 1985; Gomez and Segami, 1991; Gomez et al., 1993). More recently this model has been applied to the acquisition of knowledge from encyclopedic texts (Gomez et al., 1994) and is the same model being applied here as the front-end to the problem-solver.

The key idea in our approach is to use the final knowledge representation structures as the input to the problem-solver, rather than to use the syntactic output of the parser. The construction of the final knowledge representation structures is done as follows. The semantic interpretation phase, if successful, has built a relation and a set of thematic roles for each sentence. Let us call the thematic roles of the relation the entities for that relation. All the n entities of a n-*ary* relation are represented as objects in our language, and links are created pointing to the representation of the relation, which is represented as a separate structure called an *a-structure*. For instance, for the sentence *A company sells books of history to customers*, the NLU builds a 3-ary relation with "sell" as the relation and "company," "books of history," and "customers" as entities or arguments of the relation. These three arguments will be represented as separate objects in our representation. The words "relation" and "entities" as used in the preceding paragraph should not be confused with the notions of "relation," and "entities" in the E-R model. Although not identical to the E-R model, this representation is very close to it, making it relatively easy for the the problem-solver to decide which entities and relations in our representation stand for entities, attributes and relations in the E-R model.

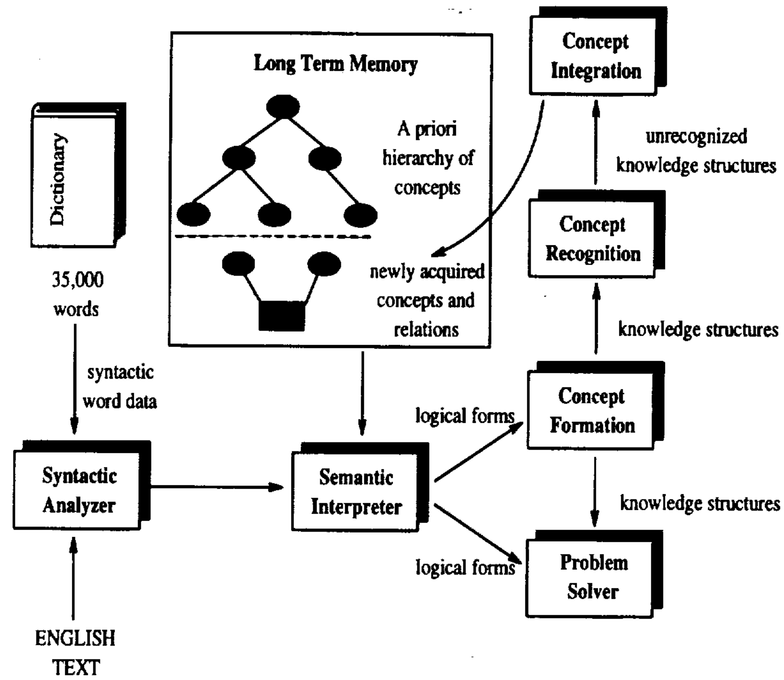

Figure 1: Main Components of the Natural Language Understanding Module

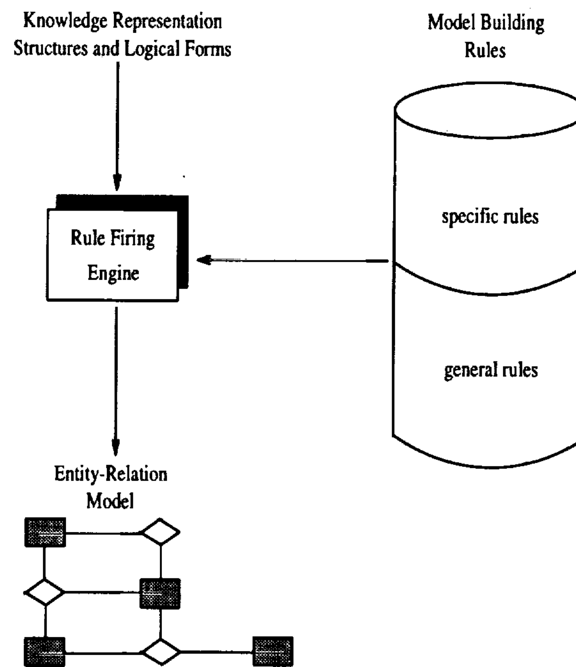

Figure 2: Main Components of the Problem Solver

Figure 2 depicts the main components of the problem-solver. There are two major sources of knowledge used by the problem-solver: the logical database under construction (LDB) and the knowledge structures being built by the NLU. The key idea in the problem-solver has been to decouple the rules that recognize relations, entities and attributes on the basis of the semantics of concepts and the relations built by the NLU from those rules that base their recognition on the entities, attributes and relations already in the LDB. The former rules are called specific rules because they depend on the semantics of the verbs and concepts in the sentence, and the latter are called generic rules because they are independent of the semantics of the sentence.

This paper is organized as follows. The next section describes the knowledge representation structures used by the problem-solver. The remainder of the paper explains the problem-solver, with two major sections describing in detail the specific and generic rules, and their role in constructing an E-R model from the user's sentences. In the last section, we give our conclusions, point out some of the limitations of the system, and future research. An appendix containing an annotated sample session with the system, which is written in Common Lisp and runs on Sparc workstations, ends the paper.

2 Knowledge Representation Structures

Each sentence entered by the user defines one or more conceptual relations. Conceptual relations can have one, two or more arguments, and are classified, accordingly, as unary, binary or n-ary. Also, conceptual relations can be classified as actions or descriptions, depending on the type of their verbal concept.

Nominal concepts that refer to physical or abstract objects are represented as frame-like structures, called *object-structures*. A sample *object-structure* corresponding to the concept "company" is shown below:

```
(company
  (is-a (organization))
  (buy (item ($more (@a6731))))
)
```

The slots in *object-structures* correspond to conceptual relations, which are also represented as frame-like structures, called *a-structures*. In the example above, the second slot corresponds to an instance of the conceptual relation "buy" ("a company purchases items from a number of suppliers"), represented by the *a-structure* @a6731 shown below:

```
(@a6731
  (instance-of (action))
  (args (company) (item) (supplier))
  (pr (buy))
  (actor (company (q (all))))
  (theme (item (q (?))))
  (from-poss (supplier (q (some))))
  (time (present))
)
```

The "args" slot in this structure contains the arguments of the relation, the "pr" slot contains the verbal concept, and the rest of the slots are the semantic cases, also called thematic roles, of the relation. The "q" slot stands for quantifier, and contains the value of the quantifier for that concept. The value of the quantifier may be not only "all" and "some," but also "most," "many," "few," etc. A question mark means that the value of the quantifier is unknown. See (Gomez and Segami, 1991) for a detailed discussion of these quantifiers, and the meaning of these structures expressed in first order predicate calculus (FOPC).

Restrictive modifiers, i.e., complex noun groups, restrictive relative clauses, or nouns modified by prepositional phrases, are represented by an object structure characterized by the presence of a "characteristic features" slot, called a *cf-slot.* The content of the *cf* slot identifies this concept uniquely by providing the necessary and sufficient conditions that define it. The structure is identified by a dummy name (a gensym). Thus, in the sentence *The person who detects the problem writes a problem report*, the restrictive relative clause "the person who detects the problem" is represented by the object structure:

```
(@x5354 (cf (instance-of (person))
            (@a5679)))
```

This structure contains the two characteristic features of the concept: "x5354 instance-of person", and "x5354 detect problem". Note that the second characteristic feature is represented by the *a-structure* @a5659 shown below. What appears in the cf-slot is simply the name, a gensym, of the *a-structure.*

```
(@a5679
  (instance-of (cf-structure))
  (args (@x5354) (problem))
  (pr (detect))
  (actor (@x5354 (q (constant))))
  (theme (problem (q (?))))
  (time (present))
)
```

Thus, a cf-slot contains one is-a/instance-of slot plus one or more names of *a-structures*. We see, then, that the representation of the concepts and relations underlying a sentence consists of a collection of *a-structures* and *object-structures*. As an example, for the sentence *The person who detects the problem writes a problem report*, the NLU builds the following representation structures:

```
(problem-report
  (is-a (report))
  (write%by
    (@x5354 ($more (@a5757))))
)

(@a5679
  (instance-of (cf-structure))
  (args (@x5354) (problem))
  (pr (detect))
  (actor (@x5354 (q (constant))))
  (theme (problem (q (?))))
  (time (present))
)

(@x5354
  (cf (instance-of (person))
    (@a5679))
  (detect (problem ($more (@a5735))))
  (write
    (problem-report
      ($more (@a5757))))
)

(@a5735
  (instance-of (action))
  (args (@x5354) (problem))
  (pr (detect))
  (actor (@x5354 (q (constant))))
  (theme (problem (q (?))))
  (time (present))
)

(problem
  (detect%by
    (@x5354 ($more (@a5735))))
)

(@a5757
  (instance-of (action))
  (args (@x5354) (problem-report))
  (pr (write))
  (actor (@x5354 (q (all))))
  (theme (problem-report (q (?))))
  (time (present))
```

3 The Problem-Solver

The problem-solver is a rule-based system that identifies relations, entities and attributes based on the representation structures built by the NLU and on the current state of the database design. Essentially, the problem solver does its work by accessing the structures that represent conceptual relations, that is, *a-structures*. *Object-structures* are considered only when they define hierarchical relations and in order to access the *a-structures* within cf-slots. The algorithm implemented by the problem-solver consists of two passes. All structures are examined in the first pass, where some structures may result in the creation of database relations, entities or attributes, others structures may cause no action by the problem-solver, and, finally, other may be saved to be considered in the second pass, after the problem-solver has had a chance to gather possibly pertinent information from other structures.

Two distinct sets of rules comprise the problem-solver, generic rules and specific rules. Specific rules are tried first. If they do not succeed, then the generic rules are tried. Specific rules take advantage of the semantic cues in a conceptual relation, when such cues are relevant to the database design. These rules are, therefore, attached to verbal concepts and are fired when the verbal concept in the *a-structure* being considered has rules attached to it. Examples of specific rules are those that construct hierarchical relations among entities, or those that identify key attributes. Generic rules, on the other hand, are fired regardless of the verbal concept in a conceptual relation. They base their actions on the arguments of the relation and on the elements currently defined in the database design. They are in turn classified as unary, binary and n-ary rules and are applied to unary, binary and n-ary conceptual relations, respectively.

A main driver in the problem-solver controls the order in which the representation structures are examined and the order and kinds of rules that are applied in each case. The first structure examined by the problem solver is always the structure that represents the main clause in the sentence (the main relation). The problem-solver then descends to the structures representing the arguments of the main relation and to explanatory relative clauses, if any. The arguments of the main relation may result in some action by the problem-solver only if their *object-structure* contains a cf-slot, that is, only if the argument is described by a complex noun phrase. In this case, the problem-solver acts on the *a-structures* that are part of the cf-slot of the argument. Let us consider a simple example. Suppose the following two sentences are read:

> An organization keeps track of customers, identified by customer id. The name and address of customers are stored.

The first sentence consists of a main clause and a subclause. The main clause introduces the conceptual relation "organization keep-track-of customer" and the subclause, an explanatory relative clause, introduces the relation "customer identified-by customer-id". The problem-solver deals first with the main relation. Since no specific rules are attached to "keep-track-of," a binary generic rule defines "organization" as an entity and "customer," tentatively, as an attribute of "organization." The problem solver then examines the arguments of the main relation, "organization" and "customer." Because both arguments are represented by *object-structures* without cf-slots,

they lead to no action on the part of the problem-solver. Next, the problem-solver examines the conceptual relation introduced by the explanatory relative clause. The verbal concept in this relation has a specific rule attached to it that identifies "customer-id" as a key attribute of "customer." Because "customer" is currently defined as an attribute in the database design, the rule redefines it as an entity, and uses the previous clause to define a database relation. Thus, after the first sentence the problem-solver has built two entities, "organization" and "customer," a key attribute for "customer," "customer-id," and a database relation, "organization keep-track-of customer."

Next, the second sentence is read. This sentence defines a unary conceptual relation. Its verbal concept has specific rules attached to it, which try to identify the arguments of the relation as database attributes. In this case, "name" and "address" are identified as attributes of "customer." We now discuss in detail the different kinds of rules in the problem-solver.

4 Specific Rules

As described above, specific rules are defined for a verbal concept when its semantics indicate that an action specific to the concept must be performed by the problem-solver. Such is the case, for example, with verbal concepts that define hierarchical relations among entities, or those that define key attributes.

Hierarchical Relations Hierarchical relations among entities are introduced by the *is-a* verbal concept. Apart from sentences that explicitly define *is-a* relations, such as *A manager is an employee*, this relation also results from other constructions. For example, for the paragraph:

> Each person keeps a record of documents of interest. Documents may be books, identified by author name and title, journal articles, identified by journal volume, number, author name, and title, and private correspondence, identified by sender and date.

the problem-solver creates the entities "document," "book," "journal article," and "private correspondence," and it establishes the conceptual relations:

- "book is-a document"
- "journal article is-a document"
- "private correspondence is-a document"

These conceptual relations are not translated into database relations, but are maintained by the problem-solver to keep track of the inheritance of attributes among entities.

Verbal Concepts that Introduce Attributes Some verbal concepts strongly suggest that the arguments in the conceptual relation describe attributes of entities. Some specific rules are attached to these verbal concepts in order to identify the attributes and their corresponding entities. The entities may or may not be explicitly identified in the relation. Consider, for example, the sentences: *The source, the time, and the location of each document are stored, The hour and the length of use are recorded, The organization keeps a record of the addresses of the suppliers.* All these sentences are associated with the "store-information" verbal concept, and the arguments in these relations, ("the source of each document," "the time of each document," "the location of each document," "the hour," "the

length of use," "the addresses of the suppliers") all seem to describe attributes of entities. Whether or not they are taken as attributes depends on the current state of the database under construction. If the argument describes a property or characteristic pertaining to a concept that has been defined as an entity by previous statements, then this property is taken as an attribute of the entity. Such is the case, for example, with "the location of the document," if "document" has previously been defined as an entity. If this is the case, then "location" is taken to be an attribute of "document." Thus, after reading the second sentence in the paragraph:

> Each person keeps a record of documents of interest. The source, the time, and the location of each document are stored.

the problem-solver identifies "source," "time," and "location" as attributes of "document." In this example, the identification of attributes and entities by the problem-solver is possible because all three arguments of the conceptual relation are represented by *object-structures* with a cf-slot. An examination of the *a-structures* referenced in the cf-slot allows the problem-solver to reach its determination. The same mechanism is used to identify "address" as an attribute of "supplier" from the sentence *The organization keeps a record of the addresses of the suppliers.*

A different situation is illustrated by the sentence *The hour and the length of use are recorded.* Here, the arguments of the relation, "hour" and "length of use," do not explicitly link these possible attributes with any concept, that is, with any previously defined entity. The problem-solver first tries to recognize these concepts by examining the attributes and entities already identified. If this fails, an interaction with the user is started, in which the problem-solver inquires about entities that might be associated with the arguments of the relation.

A similar mechanism is used for conceptual relations with the "interest-of" verbal concept, such as, "the registration number, the registration termination and the address of a registration office in each state are of interest." The actions taken for this primitive are the same as the actions for "store-information."

Verbal Concepts that Define Key Attributes

Key attributes are typically introduced by the verb "identify" in the passive form, as in *Items are identified by item type*, or *A person, identified by a person id, can own any number of vehicles.* Thus, either the main clause is passive, or it contains an explanatory relative clause in the passive form. Many times, however, key attributes are also introduced by restrictive relative clauses, i.e., *Each vehicle is registered in one or more states identified by state name.* The distinction is important for the problem-solver because the representation structures built for the two cases are different. As we saw above, an argument in a conceptual relation described by a noun restricted by a relative clause is represented by an *object-structure* with a cf-slot. An examination of the *a-structures* in the cf-slot leads us to the "identify by" relation. On the other hand, when an argument in a relation is described by a noun followed by an explanatory relative clause, the representation of the argument does not contain a cf-slot. Instead, the "identify by" relation appears as a conceptual relation in the *object-structure* of the argument. For this reason, after examining the main relation the main driver looks for explanatory relative clauses in the sentence and passes the corresponding *a-structure* to the rule-firing engine.

Other constructions that lead to key attributes result from certain adjectives: *Each major has a unique name*, *Each building in an organization has a different building name*, *The meeting rooms have their own room number.* In all these cases, the verbal concept is "property-r," and the second argument of the conceptual relation is an instance of the LTM (long-term memory) category "name" (names of things). Thus, a specific rule is attached to "property-r" which examines the representation of the second argument of the relation to verify that it is an instance of "name" modified by the property "unique." Note that the representation of "unique name" constructed by the NLU is:

```
(@x5476 (cf (is-a (name)) (@a5482)))

(@a5482
  (instance-of (cf-structure))
  (args (@x5476) (unique))
  (pr (property-r))
  (descr-subj (@x5476 (q (all))))
  (descr-obj (unique (q (?))))
)
```

5 Generic Rules

The second category of rules that comprise the problem-solver are the generic rules. As noted above, these rules are not associated with any particular verbal concept and do their work based only on the arguments of the conceptual relation and the current state of the database design. The steps taken by these rules differ significantly, depending on whether they are unary, binary or n-ary rules. Generally, unary rules result in the definition of attributes; binary rules may define attributes, entities and relations; while n-ary rules result in the definition of database relations. Typically, most sentences in a database description introduce binary relations. Unary relations normally derive from sentences in the passive form, with verb phrases such as, "are stored," "are recorded," "are of interest," etc.; although we can find sentences like *There are six warehouse locations.*

Unary Rules

When a conceptual relation has a single argument, three cases must be considered: the argument has already been defined as an entity; it has been defined as an attribute; or it does not exist in the database being designed. In each of these cases, the conceptual relation may or may not introduce constraints. These situations are summarized in the following table:

```
         Argument          Constraint
         --------          ------
case1:   Entity             Yes/No
case2:   Attribute          Yes/No
case3:   Does not exist     Yes/No
```

In the first case, the system interacts with the user to inquire if another entity in the database constitutes a second argument of the conceptual relation. If so, a database relation is created. Otherwise, no action is taken. In the second case, the system inquires if the argument is an attribute of an existing entity. If so, the argument is defined as an attribute of the user-supplied entity. Otherwise, no action is taken. In the third case, if no entities or attributes are currently defined in the database, the argument is defined as an entity. Otherwise, an interaction with the user is started. In all these cases, if the conceptual relation introduces constraints, these constraints are added to the corresponding relation.

Binary Rules

These rules are applied to conceptual relations with two arguments. For each argument the possibilities are: it exists in the database as an entity; it exists as an attribute; or it does not exist in the database.

	Argument 1	Argument 2	Relation	Constraint
	-----	-----	-----	-----
case1:	Entity	Entity	Yes	Yes/No
case2:	Entity	Entity	No	Yes/No
case3:	Attribute	Entity	No	Yes/No
case4:	Does not Exist	Entity	No	Yes/No
case5:	Entity	Attribute	No	Yes/No
case6:	Attribute	Attribute	No	Yes/No
case7:	Does not Exist	Attribute	No	Yes/No
case8:	Entity	Does not Exist	No	Yes/No
case9:	Attribute	Does not Exist	No	Yes/No
case10:	Does not Exist	Does not Exist	No	Yes/No

Figure 3: Binary Rule Cases

If the two arguments exist in the database as entities, then the corresponding relation may or may not be defined in the database. These cases are summarized in Figure 3.

The first column indicates whether the first argument of the conceptual relation exists in the database as an entity, attribute, or whether it does not exist. The second column applies similarly to the second argument. The third column indicates whether the relation already exists between the two arguments. The fourth column indicates whether the conceptual relation defines constraints.

In each of these cases the problem-solver defines entities, attributes or relations, updates relations, or adds constraints to a relation. Let us consider case 2. Suppose that "company" and "books" are entities in the database, and that the relation "sell" does not exist in the database. If the user enters the sentence *The company sells books*, the problem-solver defines the database relation "sell" with arguments "company" and "books." Similarly, if the user enters *The company sells books*, and "company" and "books" do not exist in the database design, then the problem-solver creates the entity "company" and defines "book" as an attribute of "company." In each of the ten cases, the actions taken by the problem-solver are the following:

Case 1 Update the relation between argument 1 and argument 2. Some new information may be present in the conceptual relation, such as, quantification.

Case 2 Build a new relation for argument 1 and argument 2.

Case 3 Convert argument 1 into an entity and build a new relation for argument 1 and argument 2.

Case 4 If the relation is *1:n* (meaning the quantifier of argument 1 is 1 and the quantifier of argument 2 is greater than 1), then create a new entity for argument 1 and build a new relation for argument 1 and argument 2. Else, add argument 1 as an attribute of argument 2.

Case 5 Convert argument 2 into an entity and build a new relation for argument 1 and argument 2.

Case 6 Convert both argument 1 and argument 2 into entities and build a new relation for argument 1 and argument 2.

Case 7 If the relation is *1:n*, then create a new entity for argument 1, convert argument 2 into an entity, and build a new relation for argument 1 and argument 2. Else, convert argument 2 into an entity and add argument 1 as an attribute of argument 2.

Case 8 If the relation is *1:n*, then create a new entity for argument 2, and build a new relation for argument 1 and argument 2. Else, add argument 2 as an attribute of argument 1.

Case 9 If the relation is *1:n*, then convert argument 1 into an entity, create a new entity for argument 2, and build a new relation for argument 1 and argument 2. Else, convert argument 1 into an entity, and add argument 2 as an attribute of argument 1.

Case 10 If the relations is *1:n*, then create a new entity for argument 1, create a new entity for argument 2 and build a new relation for argument 1 and argument 2. Else, create a new entity for argument 1, and add argument 2 as an attribute of argument 1.

In each of the previous cases, if the conceptual relation defines some constraints, these constraints are added to the database relation.

N-ary Rules

These rules differ from unary and binary rules in that the final result is always an n-ary database relation for the supplied arguments. The following table contains the possible cases.

	Entities	Relation	Constraints
case1:	Yes	Yes	Yes/No
case2:	Yes	No	Yes/No
case3:	No	No	Yes/No

The first column indicates whether all of the arguments of the conceptual relation exist as entities in the database. The second column indicates whether or not an n-ary database relation exists for the given arguments. The third column indicates whether or not one or more constraints are implied by the conceptual relation.

The problem-solver actions are as follows:

Case 1 Update the relation between the given arguments.

Case 2 Build a new relation for the given arguments.

Case 3 Create a new entity for each of the given arguments which does not have an associated entity in the database and build a new n-ary relation for the given arguments.

As before, in each case if the conceptual relation defines some constraints, these constraints are added to the database relation.

6 Conclusion

We have described an approach to the automated construction of logical database designs for small application domains. The method hinges on using as input for the problem-solver elaborate knowledge representation structures constructed from the logical form of the sentences. Because these structures have a close relation to the representation used in the E-R model, a set of generic rules for the problem-solver can

be systematically derived from these knowledge representation structures and the state of the logical database under construction.

One of the limitations of the system is that, if the NLU is unable to fully interpret a sentence, the problem-solver is not even activated. The user is then asked to rephrase the sentence for which no semantic interpretation was found. But, in many cases there is sufficient information in the partial semantic interpretation for identifying the entities, attributes and relations. In order to make the system more robust, we need to pass whatever partial information the NLU has constructed to the problem-solver. In other words, to make the problem-solver work with less-than-ideal semantic interpretations becomes an imperative for achieving a robust system that does not fail on seemingly easy sentences.

Because the method is based on semantic interpretation, a user needs to convey to the system some background knowledge about the words he/she is using to describe the database application so that the NLU can produce a semantic interpretation. Hence, a major unfinished goal of this work is to design and implement a knowledge acquisition interface by means of which end-users can convey the background knowledge needed by the system to construct a database model for the user's application. We have done some initial investigation of this problem and, in most cases, this is going to require only a mouse click on the part of the user to select one concept among a set of concepts presented by the system. This is possible because the system already operates with a rich ontology of concepts. For instance, suppose that a user wants to write a description of a database including the word "shuttle." He/she will be asked to choose between the three possible meanings of "shuttle": 1) a vehicle to transport things, 2) an instrument when playing badminton, or 3) a reel. This component is essential for the system to be transported across domains. The goal is to allow the user to tune the system to each specific area of application, without the intervention of programmers, knowledge-engineers or linguists.

The initial goal of this research was to design a problem-solver that would identify relations, entities and attributes with little or no help from the user, and this paper has provided a detailed description of the problem-solver. However, if one brings the user into the loop, the system described in this paper is greatly enhanced. The user can refine the final design of the database by clicking in the entities, attributes and relations. This clicking may result in deleting wrongly identified entities, or attributes, and rearranging some of the entities, relations and attributes. The nature of this interaction will be the object of future reports.

References

Albano, A., De Antonellis, V., & Di Leva, A. (Eds.) (1985). *Computer-Aided database design.* North-Holland

Alshawi, H. (1985). Creating relational databases from English texts. In *Proc. of the 2nd IEEE Conference on Artificial Intelligence Applications*, 449-454.

Ballard, B. and Tinkham, N. (1984). A grammatical framework for transportable natural language processing. *Computational Linguistics*, 10, 2, 81-96.

Baldsissera, C., Ceri, S., Palegatti, G., & Bracchi, G. (1979). Interactive and formal specification of user's views in data base design. In *Proceedings of the International Conference on Very Large Data Bases*, 262-272.

Bates, M., Moser, M., & Stallard, D.(1986). The IRUS transportable natural language database interface. In M. Evens (Ed.) *Expert Database Systems*, Benjamin/Cummings, 617-631.

Ceri, S. (Ed.). (1983) *Methodologies and tools for data base design.* North Holland Publishing Company.

Chen, P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1.1, 9-36.

Gomez, F., and Segami, C. (1989). The recognition and integration of concepts in understanding scientific texts. *Journal of Experimental and Theoretical Artificial Intelligence.* 1,51-77.

Gomez, F., Segami, C., and Hull, R. (1993). Prepositional attachment, prepositional meaning and determination of primitives and thematic Roles. UCF Technical Report.

Gomez, F., and Segami, C. (1991) Classification-Based reasoning. *IEEE Transactions on Systems, Man and Cybernetics*, 21,3, 644-659.

Gomez, F., Hull, R., & Segami, C. (1994) Acquiring knowledge from encyclopedic texts. In *Proceedings of the ACL 4th conference on applications of natural language processing.* Morgan and Kaufmann.

Grosz, B., Appelt, D., Martin, P., & Pereira, F. (1987). TEAM: an experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32, 2, 173-243.

Kersten, M.L. (1987). A conceptual modelling expert system. In Spaccapietra, S. (Ed.). *The entity-relationship approach.* North Holland, 35-47.

Martin, J. (1981). *An end user's guide for data bases.* Pretince-Hall, Englewood Cliffs, N.J.

Navathe, S.B., & Elmasri, R., & Larson, J. (1986). Integrating user views in database design. *Computer*, Jan., 50-62.

Storey, V. (1988). *View Creation.* ICIT Press.

APPENDIX

Sample Session with the Problem Solver

>>> a problem report is written if an anomaly is detected during an operation.

REMARKS: The input sentence is first processed by the NLU, where it goes through the phases of parsing and interpretation, formation of concepts, recognition of concepts, and long-term memory integration of concepts. As an illustration, we show the output of the parser and the representation structures built for the current sentence:

Parser Output:

```
g5301
(subj ((parse ((udt a) (adj problem)
       (noun report)))
       (ref (indefinite))
       (plural nil)
       (interp (problem-report
         (q (all))))
       (semantic-role (theme)))
 verb ((aux (is))
       (main-verb write written)
       (tense pres) (voice passive)
       (num sing) (prim (write)))
 conj ((if) (interp (if (q (?)))))
 sub-cl ((parse ((g5442)))
         (sub-clause (g5442))
```

```
 (interp (proposition
          (q (?))))))

g5442
(subj ((parse ((udt an)
       (noun anomaly)))
       (ref (indefinite))
       (plural nil)
       (interp (anomaly (q (all))))
       (semantic-role (theme)))
 verb ((aux (is))
       (main-verb detect detected)
       (tense pres) (voice passive)
       (num sing) (prim (detect)))
 prep ((parse (during ((udt an)
          (noun operation))))
       (ref (indefinite))
       (plural nil)
       (interp (operation (q (?))))
       (attach-to (verb (strongly)))
       (semantic-role (at-time))))
```

Representation Structures:

```
(problem-report
   (is-a (report)))
(operation
   (is-a (thing))
     (related-to (@a5702)))
(@a5702
   (args (anomaly) (operation))
   (pr (detect%by))
   (theme (anomaly (q (all))))
   (at-time (operation (q (?))))
   (instance-of (proposition))
   (time (present)))
(@a5757
   (args (problem-report))
   (pr (write%by))
   (theme (problem-report
     (q (all))))
   (instance-of (action))
   (time (present)))
```

REMARKS, these structures are passed to the Problem Solver:

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a12754 (args (problem-report))
  (pr (write%by))
  (theme (problem-report (q (all))))
  (instance-of (action)))

firing default-a-structure-delay-
  integration rule
```

PROBLEM SOLVER PASS NUMBER 2

```
Integrating structure:
(@a12754 (args (problem-report))
  (pr (write%by))
  (theme (problem-report (q (all))))
  (instance-of (action)))

firing unary-case-3-a rule
creating an entity
Entity:  problem-report
```

REMARKS: Because the structure @a12754 represents a unary relation, the problem solver delays its processing until the second pass. In the second pass, unary rule 3 fires, which defines the entity problem-report.

```
::: next statement
  ( 'x' to exit, 'help' to see menu)


>>> each problem report is
identified by a unique number.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a13079 (args (problem-report)
          (@x13008))
  (pr (identified-by))
  (descr-subj (problem-report
    (q (each))))
  (descr-obj (@x13008 (q (?))))
  (instance-of (description))
```

```
firing prim-implies-key rule
adding a key attribute to an entity
Entity:  problem-report
Attribute:  @x13008
```

REMARKS: a specific rule identifies @x13008 (unique number) as a key attribute of problem report.

```
::: next statement
  ( 'x' to exit, 'help' to see menu)
```

```
>>> each problem report contains
the name of the person who detected
the problem.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a13864 (args (problem-report)
           (@x13792))
  (pr (consist-of))
  (descr-subj (problem-report
    (q (each))))
  (descr-obj (@x13792 (q (?))))
  (instance-of (description)))
```

```
firing consist-of-first-arg-entity
  rule
adding an attribute to an entity
Entity:  problem-report
Attribute:  @x13792
```

REMARKS: a specific rule identifies @x13792 as an attribute of problem-report. @x13792 represents the concept "name of the person who detected the problem."

```
::: next statement
  ( 'x' to exit, 'help' to see menu)
```

```
>>> each problem report contains
the location of the procedure.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a14428 (args (problem-report)
           (@x14356))
  (pr (consist-of))
  (descr-subj (problem-report
    (q (each))))
  (descr-obj (@x14356 (q (?))))
  (instance-of (description)))
```

```
firing consist-of-first-arg-entity
  rule
adding an attribute to an entity
Entity:  problem-report
Attribute:  @x14356
```

REMARKS: @x14356 represents the concept "location of the procedure."

```
>>> each problem report contains
the name of the procedure that was
being run when the anomaly was
detected.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a15516 (args (problem-report)
           (@x15444))
  (pr (consist-of))
  (descr-subj (problem-report
    (q (each))))
  (descr-obj (@x15444 (q (?))))
  (instance-of (description)))
```

```
firing consist-of-first-arg-entity
  rule
adding an attribute to an entity
Entity:  problem-report
Attribute:  @x15444
```

REMARKS: @x15444 represents the concept "name of the procedure ...."

```
::: next statement
  ( 'x' to exit, 'help' to see menu)
```

```
>>> a problem report is classified
as "open" while the problem remains
unsolved.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a16385 (args (problem-report)
               (open))
  (pr (designate%by))
  (theme (problem-report
    (q (all))))
  (designation (open (q (constant))))
  (instance-of (action)))

firing binary-case-8-c rule
adding an attribute to an entity
Entity:  problem-report
Attribute:  open

::: next statement
  ( 'x' to exit, 'help' to see menu)


>>> a problem report is classified
as "closed" when the problem is
solved.
```

PROBLEM SOLVER PASS NUMBER 1

```
Integrating structure:
(@a17090 (args (problem-report)
               (closed))
  (pr (designate%by))
  (theme (problem-report
    (q (all))))
  (designation (closed
    (q (constant))))
  (instance-of (action)))

firing binary-case-8-c rule
adding an attribute to an entity
Entity:  problem-report
Attribute:  closed

::: next statement
  ( 'x' to exit, 'help' to see menu)
```

```
>>> (entities)

The following entities were created:
Entity Name
  problem-report
Immediate Parent Entities

Key Attributes
  number:
    @x13008  (cf (is-a number)
                 (@a13014))
Non-Key Attributes
  name:
    @x13792  (cf (is-a (name))
                 (@a13798))
  location:
    @x14356  (cf (is-a (location))
                 (@a14362))
  name:
    @x15444  (cf (is-a (name))
                 (@a15450))
  open
  closed
```

REMARKS: After reading the paragraph, the problem solver has created the entity problem-report, with one key attribute and five other attributes.