AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

by

DAVID ELDEN LEASURE

B. A., Kansas State University, 1982

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

Table of Contents

LD
2668
.T4
1984
L42
c. 2

A11202 958365

List of Figures appearing in the Thesis

Figure Number and Title Page

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

Introduction to the Problem

Chapter 1

The design of commercial data base applications typically involves a design team. Because of the large number of data elements, the potential for conflicting and inconsistent views of the data is high. This paper explores the feasability of using a computer to assist in avoiding design problems.

Although differences exist between manufacturers, most commercial database systems can be viewed abstractly using the ANSI SPARC DBMS Model [JARD77]. The model is useful because it allows ordered and simplified problem solving by separating out the unneeded details at each level. The three levels of the model are the external schema, the conceptual schema, and the internal schema. The external schema corresponds to the user view. It is a subsetting of the total data base model (described by the conceptual schema) which allows a user to concentrate on the portion of the data base which concerns him. The conceptual schema represents the entire description of the data base at a level separate from the physical implementation. Each of the external schemata is mapped to a portion of the conceptual schema. The internal schema is a mapping from the conceptual schema to physical devices in the system. Generally the user is protected from this level of the data base. This paper will concentrate only on the top two levels.

To cope with the growing complexity of data bases, designers have developed several design methodologies. Although the methods vary in their particulars, a general philosophy has evolved. This philosophy is composed of four components [TEOR82],

1) Requirements formulation and analysis
2) Conceptual design
3) Implementation design
4) Physical design.

Of these four steps, the first two are the focus of this paper. (See figure 1.1 for a comparison of the design steps with the levels of data base description.)

Step 1, **Requirements**, is a statement of the scope of the model, the general information and processing requirements for the organization being modelled, and the translation of those requirements to machine readable form. The scope and requirements are built up by a series of interviews between analysts and employees within the organization. These interviews range from the top management down to the clerical help. The machine readable form is usually a set of local views representing the various functions of the departments and a list of the entities involved in the local views. Quite often this list contains redundancies in the form of homonyms, syntactic forms which are equivalent but have different semantics, and in

Figure 1.1. Comparison of design steps to ANSI SPARC data base description levels [TEOR82] [JARD77]:

DATA BASE DESIGN TERMS	ANSI SPARC DATA BASE LEVELS OF ABSTRACTION
REQUIREMENTS FORMULATION AND ANALYSIS: Development of local views.	EXTERNAL SCHEMA: Views of specific applications.
CONCEPTUAL ANALYSIS AND DESIGN: Consolidation of local views into central model.	CONCEPTUAL SCHEMA: model from which external schema are taken.
IMPLEMENTATION DESIGN: Refinement of central model into machine processible schemata.	
PHYSICAL DESIGN: mapping of conceptual schema to physical storage devices.	INTERNAL SCHEMA: mapping of conceptual schema to physical storage devices.

the form of synonyms, different syntactic forms which have equivalent semantics.

Step 2, the **Conceptual Design,** consists of consolidating the various local views into a unified central view. The consolidation is done by identifying commonalities in form and function within the local views and merging these similar parts into a central form. Currently, the methods for deriving the central form are more like an art form than a design methodology. Among the many problems which occur at this stage of the design, TEOR82 identified the following problems as the most difficult to deal with in consolidation of the local views:

1) Name Inconsistency: the existence of homonyms and synonyms in the separate designs.
2) Identification differences: the use by different units of the organization of different keys to access the same data.
3) Aggregation inconsistencies: different groupings of dissimilar data items.
4) Mutual subsets: different views may handle only subsets of a data item. Often this distinction is implicit and must be discovered by the designer.
5) Conflicting update requirements: more than one view can allow for insertion or deletion of a data item, leaving the way open for data inconsistencies.
6) Conflicting integrity constraints: different departments may have different domains or allow different combinations of data items to occur.

Complications increase when more than one designer is involved in the interviews, because of communication breakdown. The problem becomes even more difficult when separate information systems already exist for each department and must be integrated. Even worse, the problem could occur across a geographically distributed environment of separately operating databases. Consolidation is a problem even after the information system is defined the first time because of the addition of new views as a business evolves.

Many factors contribute to the above design problems. TEOR82 associates the design difficulty with inadequate semantic description methods. Another problem is the willingness to accommodate existing, poorly designed usages into the new design. Ignoring the lesson of abstraction has also hurt the design process. Too often the representation details are not separated from the design details. Code forms such as M-male, F-female may also be represented as 1 and 0 and are carried along by the design team, while the preferable method is to have a syntax independent representation which says that there are two sexes to each person, male and female.

The lack of a well defined semantic notation also implies the lack of a well defined set of rules for combining different user views into workable schema. Using formal methods allows for machine assistance in the design process to manage complexities and to perform consolidations.

Current Technology:

Based upon the previously outlined findings, little attention has been paid to the above problem. Nevertheless, several related tools and projects are worth examining for their approaches to aiding data base design and data modelling.

Some of the more primitive methods for detecting synonyms and homonyms are syntax based. HUBB79 describes a system which displays the entity list to the user in KWIC (KeyWord In Context) form and then asks the user to examine entities with similar words for possible redundancies. Figure 1.2 gives an example list from a KWIC index to identify potential homonym and

Figure 1.2
This example illustrates the use of a KWIC index to identify potential synonym or homonym problems. In this case, the "Date" identifiers may represent homonyms or synonyms, while "Name" groups suggest possible synonym problems. What is not shown, is the possibility of "Address" and "Location" being synonyms which need to be combined to one form. Also, there is no way to tell by inspection whether the two "Rate"'s are homonyms or separate concepts.

```
      Address
  End Date
Start Date
      Date In
      Date Out
      End Date
      First Name
      Last Name
      Location
      Name
 Last Name
First Name
      Pay Rate
      Rate
      Rate
  Pay Rate
      Start Date
```

synonym problems. This method is unsafe because as the number of analysts and entities grows, the complexity increases greatly. Information which seemed adequate to describe local views separately is now insufficient to determine uniqueness or sameness between views.

Some of the most obvious tools available have been developed for use in software engineering. TEOR82 points out the utility of one in particular, the Problem Statement Language/Problem Statement Analyzer. PSL/PSA is a tool used to automate the production and management of documentation for the design of a generalized information system [TEIC77]. It collects and maintains all of the information normally produced during such a design. PSL/PSA succeeds compared to manual methods. Because of its limited goal of automating manual methods, too little information is collected about the data semantics to describe the local views in sufficient detail for consolidation.

Another available tool for data base design is the data dictionary. These systems allow for the storing of data definitions, usually in syntactic terms, but also in relation to other data items. While they aid in the management of certain data description functions, they are better suited to handling data descriptions which have already passed the consolidation stage and contain none of the described inconsistencies. A data dictionary system described by FISH81 works to correct this failing by collecting and maintaining synonym information, but does nothing to identify unknown inconsistencies. Additionally, data dictionary systems provide no functions for the description of procedural requirements [TEOR82].

Both software engineering systems and data dictionaries share the problem of communication with users [TEOR82]. Neither system is particularily friendly to novice users since each usually has a language unique to itself. In order for design tools to be effective, they must allow the designers the freedom to think about the problem, instead of thinking about how to communicate the problem to the computer. In addition, these tools still tend to be overly restrictive in their requirements, basing the description of data and procedures on syntactic methods.

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

That a more semantic representation of data is needed for data base modelling is not a new realization. Whether dealing with data base design or with actual modelling, current technology is lacking. This is evident by the number of researchers working on extending existing models, such as SCIO79's effort to extend Codd's relational model, and by the number of research projects developing new knowledge representations. CODD80 reports that as of 1979, over 40 logical data models (models used at the conceptual level of data base systems)had been developed. More have been added. In the more general area of knowledge representation, BRAC80 reports that more than 80 groups are working on knowledge representation schemes. Still further evidence is the inclusion of over 40 position papers listing many more references on the subject of conceptual modelling and data bases in ZILL80; all of them implying the need for more inclusion of semantic information.

One of the better defined data models is the Semantic Data Model [HAMM81]. SDM is intended to be used as the conceptual schema in a database system. Borrowing from work on frames in artificial intelligence, SDM organizes entities into classes. These classes are then interrelated through various definitional mechanisms including set algebra and predicate calculus. SDM provides for the modelling of concrete objects, events, categories and other higher level entities, and names. Included in the model are the access and manipulations which are allowed over the data base. Unfortunately, although SDM represents a great improvement over current commercial systems, it falls short of describing enough of the semantic components needed for data base design as explained in the next paragraph.

Of the research projects examined so far, each either deals with the consolidation problem syntactically or ignores the problem altogether. Because of the emphasis on syntactic representation, each fails to provide for enough semantic description [TEOR82]. SOWA80 describes six parts needed in any conceptual model or knowledge representation scheme for complete semantic description of data.

1) A Type Hierarchy: Types are categories which data items fit into by virtue of their characteristics. A type hierarchy allows items to be described at appropriate levels of generality and provides for default inheritence for subtypes. **TRUCK-DRIVER is an EMPLOYEE is a PERSON is an ANIMAL is a LIVING-THING is an ENTITY** is one example type hierarchy, where each type is successively more general than the preceeding one. Types embody the downward inheritance mechanism described by CARB80.

2) Functional Dependencies: Functional dependencies describe how items may be found in a data base. They discriminate keys from non keys, independent variables from dependent variables. This usage is from Codd's relational

model. It should be extended to also show the functionality of the dependencies: the relationship of a key type to a non key type may be 1 to 1, 1 to many, many to many, or even many to one. For example, a common functional dependency is the unique determination of a person's NAME from his SOCIAL-SECURITY-NUMBER, or the NAME's of all his children. The first example is 1 to 1, the second, 1 to many.

3) Domain Roles: These descriptions show what role the functional dependency plays. It is not enough to say that NAME functionally determines NAME's; the knowledge that NAME is the FATHER of NAMEs and that NAME's are the CHILD's, is also needed.

4) Definitions: Definitions are similar to Aristotle's genus and differentia where an object is defined by saying for example, that an EMPLOYEE is a PERSON (type hierarchy) with the restriction (or differentia) that EMPLOYEE performs WORK for a COMPANY. The type WORK is defined as performing an ACT for PAY. Definitions list only the necessary conditions for an entity's classification in that type.

5) Schemata: Each definition lists the necessary conditions for classification of entities. More is needed however, to show the conventional roles played by members of a type. Schemata describe the usual associations with other objects

and common value ranges. For instance, employees are usually associated with skills, departments, pay rates, and managers, although these are not necessary characteristics.

6) Procedural Attachments: Some way is needed of showing how some data items are derived in a data base. Often it is necessary to borrow built-in procedures from the data base environment. A simple example of this is the determination of a PERSON's AGE from the difference of the system generated current DATE and the stored BIRTH-DATE. Age would not otherwise be representable because of its continuously changing nature.

7) Inferences: The model should be provided with manipulation rules and rules from which statements about the data items can be made based on their occurances with other items or upon their extensions (values). Constraints upon the values would be described here since the only other part specifying values is the schemata, but those values are only intended to describe common ranges.

Of the many models available for semantic description of data, only a few satisfy the above criteria. One of the most complete implementations is a development from the University of Toronto called TAXIS [MYLO80] [SOWA80]. TAXIS is a programming language for the design and implementation of interactive information systems, such as airline reservation systems. It is based on several concepts from different fields. From artificial intelligence, TAXIS uses semantic networks. From programming languages, it borrows the concept of the abstract data type and exception handling. And from database theory, it has borrowed the relational data model. Three types of objects are used in TAXIS: tokens, classes and meta classes, along with the rules to combine them into type structures. Using these objects the aggregation and generalization hierarchies of SMIT77 can be constructed as well as 'instance of' relationships. Actions and constraints may also be built into hierarchies. Exception handling is also well defined. TAXIS has the strength that it is a functioning system designed specifically for data base semantics and design. Some drawbacks are that TAXIS necessarily deals with representational issues and it is limited to three levels of type hierarchy. Further, there is no mention in the literature of any attempt by TAXIS to deal with the consolidation issues, a deficiency shared by all of the design systems in the literature.

As mentioned previously, no design systems in the literature deal with the consolidation problem. The consolidation problems persist in any design effort. The solution is the adaptation of a language/representation scheme which satisfies the seven criteria of SOWA80 into a design system. One such system might have the following design:

The system shown in figure 1.3 is composed of two main repositories for information about the data base design. These repositiories are the local views and the data base system data description, the external and conceptual schema. The first repository contains information collected about the design problem in the semantic representation scheme. The last contains this same information in the language of the data base management system. Two modules communicate with the designer, the graphic display formatter and the design dialogue handler. The graphic display editor converts internal representation of the local or consolidated views into graphical representations on the designers screen. The use of graphics should help to overcome disadvantages with past systems which required expert knowledge of the language of representation in order to communicate with the system. The design dialogue module works at the direction of the consolidation engine to communicate in natural language with the designer concerning the data base design. Again, the provision of a natural language interface is included to ease the burden of non-expert design analysts. The consolidation engine contains rules which direct the construction of the consolidated view from the local views. The rules also provide for the detection and avoidance of the six inconsistencies noted by TEOR82. Finally, so that no work should have to be redone manually, a module for producing a usable set of external and conceptual schemata is included. In principle,

Figure 1.3. A proposed system to manage the design of data bases.

```
+----------+               +------------+
| DISPLAY  |               | DISPLAY    |
|   WITH   +---------------+ FORMATTER  |
|  GRAPHS  |               |            |
+----------+               +-----+------+
                                 |
                                 |
                        +--------+--------+
                        |    SEMANTIC     |
                        | REPRESENTATION  |
                        |  OF DATA BASE   |
                        +---+----+--------+
                            |    |
                            |    |
                            |    |  +----------------+
                            |    |  | CONSOLIDATION  |
                            |    +--+                |
                            |       |     ENGINE     |
                            |       +--------+-------+
                            |                |
                            |                |
                            |       +--------+------+
                            |       |    DESIGN     |
                            |       |   DIALOGUE    |
                            |       |    HANDLER    |
                            |       +---------------+
                            |
                            |
                   +--------+---------+
                   |    RULES FOR     |
                   |  DERIVING  THE   |
                   |    DATA BASE     |
                   |   EXTERNAL AND   |
                   |    CONCEPTUAL    |
                   |     SCHEMATA     |
                   +----------+-------+
                              |
                              |
                   +----------+---------+
                   | DATA BASE SYSTEM   |
                   | DATA DESCRIPTION   |
                   |     STORAGE        |
                   +--------------------+
```

this module should simply translate from the design system's internal representation to the data description language of any particular vendor's data base management system.

Solution:

The detailed design and implementation of a system with the capabilities described is a large task and should be painstakingly researched from the beginning. Nothing of this size should be undertaken without first establishing the feasability of each part. The selection of a representation language which satisfies the seven criteria for data semantics is the first goal. No other piece of the system can be designed without it, because of its centrality in the overall design. SOWA80 points out that many of the AI knowledge representation languages (also known as semantic networks and associative networks) are general enough to implement such a language, even though most do not satisfy the seven criteria without enhancement. The representation problem is not to find the one language, but to find one which is acceptable, and then to use it to show its feasability in the consolidation problem. Afterwards, the search for the language of implementation can proceed based on factors such as availability of current systems to build upon. The claim that any such language will do is supported by the work of the IBM Research Institute group on

Data Base Semantics. They are working on the implementation of the semantic graphs of SOWA84 (which are known to satisfy the seven criteria) using Heidhorn's NLP system [SOWA80]. Sturzda's implementation of a semantic network using an IBM IMS data dictionary also supports this statement [STUR83]. As mentioned before the problem is not one of implementation, rather the demonstration of the usability of a representation language to represent the local views, to allow for inferences to construct a consolidated view, and to allow for the detection and avoidance of the six problem areas during consolidation. These same rules developed for consolidation can also be used on a smaller scale in the construction of the local views from pieces brought together during the requirements analysis.

The demonstration of the representation language to handle the above problem is the primary goal of this research. Once the feasibility of the language is established, the rest of the system will fall in place. Each part has already been shown to be workable. The generation of displayable graphic representations from the linear forms of a semantic network has been implemented for the AI knowledge representation language KL-ONE [SOWA84]. The translation to and from natural language and semantic networks has been shown in many forms by SCHA77. Other groups have shown this translation to be feasable with other representations. The problem of translating the representation language into the external and conceptual schema is a variation of translating from one formal language to another; it is the work of compiler theory. As an example of a similar translation, SOWA84 describes the neccessary transformations to produce first order predicate calculus from conceptual graphs. All of the information needed for such a translation to the schema will already be available, although the mingling of conceptual and implementational levels by many data base system vendors could require the solicitation of additional implementation dependent details, such as file names and access strategies.

Methodology

Chapter 2

The solution outlined in the introduction identifies as the primary problem to be solved before a data base design system can be built, the selection of a representation language. The problem is further decomposed into two tasks, the selection of a language which satisfies SOWA80's seven criteria and the demonstration of this language's ability to serve in the consolidation process. The purpose of this paper is the accomplishment of these two objectives. To this end, this chapter establishes the methodology to be followed. These subtasks have been identified:

1. Select a language for representation.
2. Demonstrate the seven points by showing examples of each in the representation of local views.
3. Demonstrate the language's ability to represent local views by generating local views from English-like descriptions such as might be collected by the design analysts.
4. Develop a preliminary (possibly incomplete) set of rules to detect synonyms between (or even within) the various local views.

5. Develop a preliminary (possibly incomplete) set of rules to detect homonyms between (or even within) the various local views.
6. Devise a way to represent the consolidated, integrated model.
7. Develop a preliminary (possibly incomplete) set of rules for consolidating the local views into an integrated model.
8. Identify the synonyms and homonyms already present in the local views for use in validating task 9.
9. Using subtasks 4, 5, and 7 perform the consolidation, showing the point of synonym and homonym detection.
10. Validate task 9 with the output of task 8.

Each of the above tasks will be expanded upon in the following paragraphs.

The selection of a representation language should be based on two criteria. First, the language should satisfy the seven points of SOWA80 without extension. Because the demonstration of feasibility is separate from implementation, no extra effort should be made in extending current representation systems unless a suitable language cannot be found. As mentioned previously, implementation may proceed with a language which is available, easier to implement, and possibly less clear. The choice of the language for argumentation cannot, however, be difficult to work with. The second selectional criterion is that the language be easily learned and talked about, in order to aid argumentation.

The language must allow abstraction, the suppression of detail in favor of communicating at a higher level of meaning. In a representation language, abstraction would be performed by organizing related groups of details into named units, similar to modules in programming languages. When necessary, the detail could be recalled by replacing the group name with its expansion.

The language must have a similar representation for all the descriptive components. This restriction disallows non formal representation schemes such as those which use LISP code to represent things not otherwise representable in the model. Not only does such an "escape" hinder communication of the model's intent by requiring knowledge of LISP, but it also suggests a basic weakness in the model itself.

The language must use a limited number of basic forms. The definition of higher forms should be in the lower forms. With this restriction, readers are able to understand the essentials quickly, and to extend their knowledge as necessary.

SOWA80's seven points must be illustrated in the chosen language. Even if the language is said in the literature to satisfy the criteria, explicit proof is reasuring and serves as a basis for argumentation. The best method is to show each of the seven steps in an example by itself. The more specific points will be built upon more general ones. In addition, an understanding of the seven points themselves will be gained.

Although a language satisfies the description criteria, it may not be able to model local views of data. For example, a system which requires all of the data base information to be present at one view, does not lend itself to the design process. The specification of local views is usually an incremental process; therefore, a system which requires the entire model to be present before definition, is not usable. The entity relationship diagram [CHEN76] is one such system. To illustrate the languages ability to model local views, English-like descriptions of local views for an example problem will be formed. These descriptions will then be transformed into their forms in the representation language. This step is part of the data base design system and will eventually need to be formalized, but will only be dealt with informally here. Most important is the language's ability to model the local views. The local views will include problems with homonyms and synonyms for the sake of demonstration.

Rules for detecting synonyms are necessary. As with the transformation from English to internal representation, these rules are an important part of the design system and will need to be formalized. This paper will develop a set of rules but will make no claim of completeness. The intent of the rules is to show the possibility of detection. The generation of a complete set of rules, in all likelihood, is implementation dependent and best left for later. This fact is suggested by the rules in the natural language understanding program, PAM [SCHA77]. PAM's rules borrow their structure directly from the forms of the representation language [SCHA77]. Rules for detecting homonyms follow the same argument. Additionally, rules for resolving synonym and homonym problems automatically are probably possible, but left for the implementation.

Some representation may need to be found for the consolidated representation. The word "may" is used because it may not be necessary to represent the central model if consolidation can be performed on a pay as you go basis. The best system would be one which gave immediate feedback to the designers on the validity and problems of the just-entered local view. The only restriction this would enforce is that the most important views be added first. In this way, the most important views establish their priority and dominance in naming conventions. The only time a consolidated view would be necessary is during the actual running of the data base. The conceptual schema of the "real" data base is generated by a different set of rules from the semantic representation. If no way is found of incrementally specifying the local views, however, then the consolidated model is necessary.

Even if the consolidated model is found to be unnecessary, the consolidation rules will be needed. Either way, their purpose is to take local views which containing possible data inconsistencies and manipulate them at the direction of the designer and design rules to produce the consolidated view. This manipulation could include generating or modifying new definitions and schemata, or adding and deleting parts of the structure. Most likely, the structural manipulation rules will come from the representation language itself, while the guiding rules which direct them will be generated as necessary. The guidance rules will be dependent on whatever form is selected for the consolidated model.

After the three sets of rules are developed, the consolidation takes place. This point of the research is the most important, since it establishes or disproves the thesis and the feasibility of the conceived design system. The result of this task is the method in which the three sets of rules are applied. It is proposed that two views be integrated at a time, this being conceptually simpler. Even if a consolidated view is unnecessary, the process is the same. The selected local view will be compared using the preceeding rules to either the central structure or the other selected view. This consolidation will be verbally described, and the point where data inconsistencies are first detected will be noted. The process will be repeated for all local views. For this paper, the demonstration of the consolidation for two local views is sufficient.

Once finished with consolidation, the method will be validated by comparing the intentional set of inconsistencies with the set of those found. These tasks will be iterated (starting as far back as necessary) until the two sets agree.

The end result will be the demonstration of the representaion language's ability to model enough semantic information for consolidation, the generation of a preliminary set of rules for inconsistency detection and consolidation, and the development of a general methodology for applying those rules. In addition, a set of test data will exist to aid in debugging any subsequent implementaion.

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

Data Base Semantics with Conceptual Graphs

Chapter 3

As mentioned in chapter 1, the most important thing about the semantic representation language chosen for this preliminary work is that it satisfy SOWA80's criteria. To this end, conceptual graphs (CG), a representation language developed by Sowa and the Project on Data Base Semantics at the IBM Systems Research Institute [SOWA80], has been chosen as the modelling language. The choice was made for several reasons: CG is easy to learn and understand, CG is graphically oriented and conveys information well, and most importantly CG has already been shown to satisfy the seven criteria [SOWA84]. This chapter will present an overview of CG with each of the seven criteria discussed and illustrated with examples. The discussion will be based on the seven points, and will follow this order, type hierarchies, domain roles, definitions, schemata, functional dependencies, procedural attatchment and inferences.

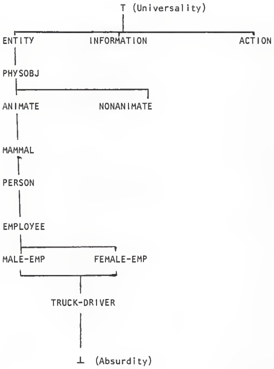
A type can be thought of as a category into which data items can be classified by virtue of their characteristics. The type hierarchy organizes these categories into a type lattice which explicitly shows the lines along which lower types may inherit properties of their more general supertypes. At the top of the hierarchy is the most general type, called universality.

All types are subtypes of universality. At the bottom of the hierarchy is the most restrictive type called absurdity, which is a subtype of all other types. The most interesting types are in between the two type extremes. Using the mechanisms described later under definitions, the type lattice may be extended to include new types made from the combination of old ones or from retrictions on existing types. Figure 3.1 illustrates one possible type hierarchy. Notice that all type labels are written in upper case. EMPLOYEE inherits all of the attributes of PERSON except any which may be explicitly excluded by the definition of EMPLOYEE. Types in CG correspond to concepts. A concept is rpresented by a box or brackets with a type label inside.

Certain extensions may be added to show additional information about a type. A type by itself in a concept box suggests that any object which is of this type may play this role. This same thing could also be expressed by concatenating a colon and an asterisk to the end of the type label, PERSON:*. To refer to some specific instance of the type, a variable may be added after the asterisk. Thus [PERSON:*x] is the same person as [PERSON:*x], but not the same as [PERSON:*y]. Other symbols denoting quantification will be discussed under functional dependencies.

Domain roles show how two CONCEPTS are related. Domain roles are represented in CG by conceptual relations. Conceptual relations are labels inside of ovals or parentheses. Conceptual

Figure 3.1. Example Type Hierarchy.



relations are always placed between two concepts with a directed line. No two concepts can ever be connected, except via a conceptual relation. The arrow determines the ordering of the relation. Consider figure 3.2. In this example (CHLD) is the conceptual relation, while [PERSON] and [BABY] are concepts. Conceptual relations may be thought of as

the relation of a concept1 is a concept2

Using this guideline, the above conceptual graph becomes:

A child of a person is a baby.

Note also that conceptual relation labels are restricted to four or fewer letters by convention.

Definitions are the means of defining new type labels. The definitional mechanism is based on the idea that the definition should only specify the necessary conditions of that type. This model is best exemplified by the biologists' taxonomy of all life. A genus specifies the characteristics of a certain type of animals, individual species within that genus share all of the listed characteristics of that genus except for the restrictions which are in the definition of the species. Details which are apt to vary among individuals of that species are not included in the definition. Canis lupis is differentiated from other members of the canine genus by his coloring, size and hunting habits. That fact that some individuals may have darker or lighter coats depending on the region they habit does not constitute a defining characteristic for the species. The extra

Figure 3.2 "A child of a person is a baby"

[PERSON] → (CHLD) → [BABY]

Figure 3.3 Type definition for RENT.

```
type RENT(X) is


[GIVE:*x] -

          (OBJ)  → [ENTITY],
          (OUR) → [TIME-PERIOO]
          (AGNT)  → [PERSON-BUSINESS:*a]
          (COST) → [MONEY:*@b]
          (RCPT) → [PERSON-BUSINESS:*c],
[GIVE] -
          (RCPT) → [PERSON-BUSINESS:*a]
          (OBJ)  → [MONEY:*@b]
          (AGNT)→ [PERSON-BUSINESS:*c].
```

information is not left unaccounted for, however; the schemata section takes care of such embelishments. Definitions in CG are defined using graph notation as shown in figure 3.3. This definition says the necessary and sufficient characteristics for the type RENT are that there exists an OBJ which is an ENTITY, which is given by a PERSON or BUSINESS to another PERSON or BUSINESS in exchange for an agreed upon sum of MONEY. The type definition (not shown) for GIVE states that the OBJ is owned (POSS) by the AGNT and that the given OBJ is actually the transfer of possesion. Figure 3.4 shows another example type definition. Although this example does not place the type BUSINESS-EST at the front of the definition, the meaning is still clear because the 'x' relates the words together. AGNT stands for agent, in this case the lessor. RCPT stands for recipient and DUR, duration. The notation '{*}' is introduced in this example, and is called the referent. It shows that the referent of the type is a plural object, as in an arbitrary number of motel rooms or customers. Definitions are also used to introduce new conceptual relations into the system. Labels in a conceptual relation are built up using the LINK relation in figure 3.5. Recall that the 'x' and 'y' are used to identify general yet separate instances of the type with which they are associated.

Schemata are used to add the extra information to the type definition which is left out. Schemata are necessary because the

Figure 3.4 Type definition for HOTEL

type HOTEL(X) is

[RENT:*y] -

(AGNT)→ [BUSINESS-EST:*x]
(OBJ)→ [HOTEL-ROOM]
(DUR)→ [TIME-PERIOD]
(RCPT)→ [PERSON: {*}].

Figure 3.5 Relational definition of CHILD(X,Y)

relation CHLD(X,Y) is

[PERSON:*x]→(LINK)→[CHILD]→(LINK)→[PERSON:*y] real world seldom fits nicely into a type scheme alone. By using schemata we can capture much of the default information, such as common values, common associations and any other information which does not fit inside the definitional mechanisms for types. The schemata implement the concept that an object can be described by all of its extensions, or all of its prototypes. This notion was put forth by Wittgenstein in 1953 [SOWA84]. Since then it has made its way into various AI representation languages such as KL-ONE [BRAC79] and KRL [BOBR77]. Schemata are created as shown in firgure 3.6. CONT stands for 'contains' and PART signifies 'has part'. This example must be a schema definition because not all HOTEL-ROOMS have bathrooms (European hotels often have only one bathroom per floor), nor do all hotel rooms have COLOR-TELEVISON. This schema shows common associations, but schemata can also show common or default values, as the figure 3.7 illustrates. This schema states that one can assume that Motel rooms along the highway cost around 35 dollars. The symbol '@' is shorthand for the QTY in the graph of figure 3.8

Functional dependencies show the access paths and quantification of the concepts in a conceptual graph. The most common form of functional dependency is the strict definition from relational data base methodology. In the restricted case, Y is said to be functionally dependent upon X if X uniquely determines Y. NAME in chapter 1 was shown to be functionally

Figure 3.6. Schema definition for HOTEL-ROOM.

```
schema HOTEL-ROOM(X)  is

[HOTEL-ROOM:*x] -
          (PART)→[CLOSET]
          (PART)→[BATHROOM]
          (CONT)→[DESK]
          (CONT)→[COLOR-TV]
          (CONT)→[DRESSER]
          (CONT)→[BED]
          (CONT)→[TELEPHONE] .
```

Figure 3.7. Schema definition for MOTEL-ROOM.

```
schema MOTEL-ROOM(X)  is

[MOTEL-ROOM:*x] -
          (LOC)→ [LOCATION:highway]
          (COST)→ [MONEY:@35 dollars] .
```

Figure 3.8 Alternate schema for MOTEL-ROOM.

```
schema MOTEL-ROOM(X) is

[MOTEL-ROOM:*x] -

          (LOC)→ [LOCATION:highway]
          (COST)→ [MONEY:dollar] -
                   (QTY)→ [NUMBER:35].
```

dependent upon SSN. In graph form the dependency is shown in figure 3.9

This graph states that for all SSN's, there exists a unique person with a unique name. The symbol '∀' is the universal quantifier which contributes the phrase 'for all' to the figure 3.9. The symbol 'El' is the unique existential quantifier which contributes the phrase 'there exists' to figure 3.9. Other dependencies are also possible. The multi-valued dependency is expressed using the generic set notation, namely '{*}'. An example of this is given by figure 3.10. The graph states: 'for each parent there exists zero or more children'. Thus far, the relationships '1 to 1' and '1 to m' have been shown. A more complex relation 'm to n' exists, although it is not given a name in relational database methodology. 'm to n' is expressed in the graph of figure 3.11 This graph states that an arbitrary number of HOTEL's are owned by an arbitrary number of PERSON's.

Functional dependencies are a description of how data items are accessed in a system, rather than a description of their status in the real world, although quantification does express their existence. Functional dependencies are necessary to describe how the information is used inside of a data base system. Because access and manipulation functions often have no corallary in the real world, the next component of a data semantics description language, procedural attachment, is

Figure 3.9 Example functional dependency, 1 to 1.

[SSN: ∀] ⟵ (ATTR) ⟵ [PERSON: E^1] ⟶ (NME) ⟶ [NAME: E^1]

Figure 3.10 Example dependency, 1 to n.

[PARENT: ∀] ⟶ (CHLD) ⟶ [CHILD: {*}]

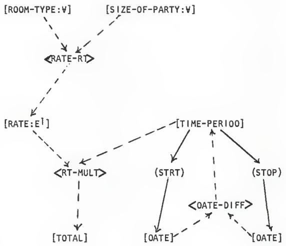
Figure 3.11 Example dependency, n to m.

[HOTEL: {*}] ⟵ (POSS) ⟵ [PERSON: {*}]

necessary. Procedural attachment is shown using a notation called actors. Actors are place holders for functions that are normally only part of the data base. They are represented in CG by labels within diamonds or angle brackets. Their use enhances the description of functional dependencies and makes possible the representation of certain virtual (computed) data. The graph in figure 3.12 shows a typical use of actors and functional dependencies to show how the total cost of a room is derived from information existant in the data base. In this example, the RATE for a room is functionally dependent on the type of room and the number of occupants. The actor <RATE-RT> is a retrieval function which takes as input a ROOM-TYPE and a SIZE-OF-PARTY and returns a RATE. Because the workings of this function have not been specified the behaviour of the data base has been described without making premature decisions about implementation In fact, this function could either implemented arithmetically or with look-up tables. The next function is not so flexible, because its use is rather more obvious. <RT-MULT> multiplies a RATE with a TIME-PERIOD to produce the TOTAL cost of a room.

The ability to specify constraints and make inferences from the data is a key function in data base systems. In CG, these constraints and inferences are specified with inference rules. The rules take the form of graphs and are interpreted by the same system, unlike other AI languages which use "escapes"

Figure 3.12 Example use of actors and functional dependencies.



to code segments called demons. The demons are actually coded in LISP or some other programming language. The inferences in CG are built by using common predicate calculus operators in CG form. Some operators are not, however, immediately obvious. Because any other primitive (non-quantifier) symbolic logic operator can be specified using the operators NOT and AND, CG has adopted them as the core from which all other logic operators are specified. NOT is represented by the symbol '¬', either in front of an entire graph, a single concept, or a type label. AND is less obvious, but intuitively clear. Two graphs representing inferences placed next to each other are said to be in conjunction. Other operators are more straight forward. By De Morgan's law, the definition of OR as a conceptual relation becomes the graph in figure 3.13, and implication is defined in figure 3.14,

which is another way of saying ¬X OR Y. Using the preceeding operators, it is possible to specify additional constraints on type referents (a referent is the extension, or actual data value) which is not specified in the schemata and type definitions. In addition, default inferences can be specified as well as derived. If further relations need defining, it is possible to go beyond first order predicate calculus, as in the case of a figure 3.15. In this example the "difference" (inequality) relation is defined. In CG any constraints on the referents or types in a definition or schema are shown by

Figure 3.13 Definition of OR.

relation OR(X,Y) is

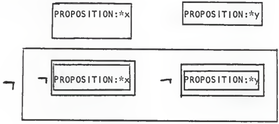


Figure 3.14 Definition of IMPLICATION

relation IMP(X,Y) is

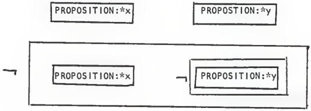


Figure 3.15 Definition of difference relation.

relation DFFR(X,Y) is

[T:*x] [T:*y]

¬ [[T:*x = T:*y]]

(Where T is universality)

placing the constraint graph next to the other graph. An example constraint is illustrated in figure 3.16. In this complex example, a motel room type is reservable if there exists a number p and a number q such that p vacanct rooms of that room type is greater than q reserved rooms of that type for a given duration.

One more thing needs to be described about CG, the legal manipulations. Without these, no derived inferences would be allowed upon the graphs. The manipulation rules are divided into two parts, rules on basic graphs structures and rules specifically designed for logical connectives.

There are four manipulative rules for working with and deriving new graphs. All are based on the concept of the conceptual canon. The canon is a group of graphs which are known to be well formed. Well formed graphs are "canonized" by actual observation of the graph in the real world ("truth"), or by derivation from other well formed graphs. Using the canon and the following inference rules, the only new graphs which can be derived will be well formed. The first rule is the COPY rule, illustrated by figure 3.17. It states that a copy of any well formed graph will be itself a well formed graph. The second rule, RESTRICT (illustrated in figure 3.18), states that any type may be replaced by a subtype. Thus ROOM could be replaced by MOTEL-ROOM in a graph. Third, JOIN (shown in figure 3.19), allows any concept c in graph u to be deleted while moving all

Figure 3.16 Schema for ROOM.

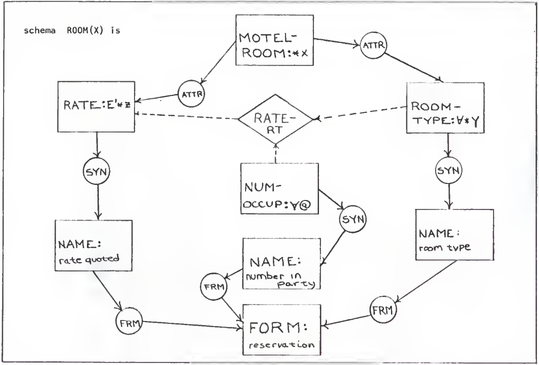


Figure 3.16 continued.

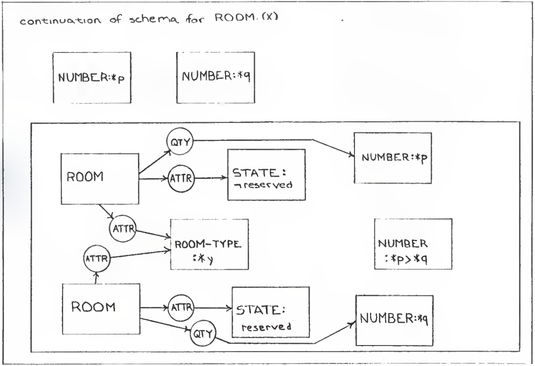


Figure 3.17 Example of the copy rule.

[FATHER]→(CHLD)→[PERSON]

[FATHER]→(CHLD)→[BABY]
[FATHER]→(CHLD)→[BABY]

Figure 3.18 Example of the restrict rule.

[FATHER]→(CHLD)→[BABY]

[FATHER]→(CHLD)→[BABY]

Figure 3.19 Example of the join rule.

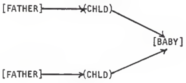


Figure 3.20 Example of the simplify rule.

[FATHER]→(CHLD)→[BABY]

of the arcs connected to it to the identical concept d in graph v. SIMPLIFY (illustrated by figure 3.20) states that relation r and all of its arcs may be deleted from a graph if the duplicate relation q conects the same concepts as r.

Additional rules in CG exist, which deal in more depth with inferences, tenses and modalities. The will not be used in the consolidation examples and are left out of this discussion.

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

Construction and Consolidation of Local Views

Chapter 4

The consolidation step in conceptual database design is performed after the collection of local views. It is performed from three points of view, identity, aggregation, and generalization [TEOR82]. Identity consolidation is concerned with discovering identical data elements and consolidating the local views on these elements. Aggregation and generalization consolidation are based on SMIT77. Aggregation is the grouping of dissimilar entities which belong together under a single class. Aggregation consolidation is the discovery during consolidation of entities in different views which belong together and grouping them in the consolidated model under a single heading. Generalization is the grouping of similar entities into a more general, less restrictive class. Generalization consolidation is the discovery of a supertype for related entities in separate views, and their grouping together under that supertype.

Because the design of any large data base is likely to be done by more than one analyst, because different departments have adopted different views of data, and because variances in style and design documentation occur, consolidation is not without its problems. As mentioned in chapter one, the following six problems occur during the consolidation process:

1. Naming inconsistencies
2. Identification inconsistencies
3. Aggregation inconsistencies
4. Mutual subsets
5. Conflicting update requirements
6. Conflicting integrity constraints

Given a uniform design style, problems 2, 5, and 6 are mostly a problem of resolution; detection is straightforward. Problems 1, 3, and 4 are problems of detection as well as resolution. Problem 1, naming inconsistency, is addressed mainly by identity consolidation. Problem 3, as the name suggests, is dealt with during aggregation consolidation. Problem 4, mutual subsets, is dealt with during generalization consolidation. Although no distinct ordering of the three consolidation types is required, identity consolidation is considered first, with the other two invoked as required.

Successful consolidation depends on the successful generation/collection of local views. TEORY lists three properties of the local views which must be satisfied before consolidation can begin:

1. The local view is complete for the user's needs.
2. All objects in the local views are uniquely named.
3. No synonyms exist within the local view (no redundancy).

Under manual methods, the above conditions are the best that one can hope for. Under machine assisted methods, it is possible to do quite a bit of aggregation and generalization consolidation as well. More will be said on this in the examples to follow.

Chapter 3 demonstrated that the system of conceptual graphs satisfies SOWA80's seven criteria for a semantic representation language. What is left to be shown is that the graphs can represent enough information for the detection and resolution of consolidation problems 1, 3, and 4. This chapter will use the partial design of a database for a motel as a base for demonstrating a solution to the problem. The problem is taken from actual documents used by a motel; these documents contain many design flaws. This chapter will demonstrate solutions to the consolidation problems of naming and aggregation inconsistency and mutual subsets. Because the success of consolidation depends so heavily on the local views satisfying TEOR82's three criteria above, the generation of local views will also be demonstrated. As much of the three inconsistencies as possible will be solved in the generation of the local views. This chapter is organized into four sections. Section 1 will detail the assumptions about the system environment. Section 2 will describe the design problem and demonstrate the generation of the local views. Section 3 will demonstrate the consolidation of those local views. Section 4 will summarize the results.

SECTION 1: ASSUMPTIONS ABOUT THE DESIGN SYSTEM ENVIRONMENT.

The proposed system in figure 1.3 is composed of several components which require a good deal of built-in knowledge about the problem area. As mentioned in chapter 1, most of these components have been successfully demonstrated in other programs. Because of this success, little need be said about their inner workings. However, one sub-component, the local view generator, is especially crucial to the consolidation process and should be discussed.

Successful machine consolidation of the local views depends not only on the properties of completeness, uniqueness, and non-redundancy, but also on the uniform translation and representation of the local views. SCHA77 has shown that it is possible to derive a single internal representation for several differing surface expressions or sentences. The internal representations of sentences like "John hit Mary" and "Mary was hit by John" are the same. Both Schank's system of conceptual dependency and Sowa's system of conceptual graphs are semantic networks. The assumption that Sowa's conceptual graphs derive consistent internal forms for surface expressions is based on the close similarity to Schank's work. Without this assumption, the system cannot work.

In addition to the derivation assumption, several assumptions about the knowledge of the problem that the local view generator brings with it need to be stated. The construction of the local views is a smaller problem than the understanding of text because there exist a limited number of goals, a limited business vocabulary (and hence a specified domain of discourse), the opportunity to ask questions of the design analyst to clarify ambiguities, and the opportunity for the design analyst to modify on a limited basis the internal representation of any local view. As much detail can be provided as is neccessary for understanding. Additionally, new concepts may be added to the system as usage patterns develop, thus making the system more and more intelligent with use and conserving of the intellectual effort by the design analysts. The main assumption is that the system already has a number of common business verbs defined as well as the normal group of basic nouns, modifiers, and connectives.

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

SECTION 2: Rules for generation of the local views and consolidation.

The process of interviewing is used to construct the local views. The system needs some additional techniques beyond natural language in order to form the conceptual graph form of the local views. In addition, rules by which consolidation is performed are necessary. Following is a list of rules by which the conceptual graph form of the design problem is derived. These rules, coupled with the basic inference rules described in chapter 3, will be used in section 3 and 4 to construct the local and consolidated model.

Rule 1.

Make use of existing design information as much as possible. This includes any information such as forms. Experience shows that forms are readily understandable by the lay and specialist alike. Because they embody some design information already, they ease the analysis burden.

Rule 2.

For each form, try to find an action verb which captures the essence of the form. Using verbs gives the system the expectations of finding the concepts which are linked to the verb. Use type definitions first, as they represent the necessary and sufficient conditions, then use the schemata. The verb analysis approach is common in natural language processing, but its use in database design is not established. Nevertheless, it is an extremely useful way of predicting the fields to come and insuring that all needed information is specified.

Rule 3.

Check for homonyms and synonyms by checking the deep representations for similarity in forms. Because uniformity is enforced at several levels, these deep forms should be readibly matchable. The restriction operation (chapter 3) is extremely useful for moving up and down the type lattice.

Rule 4.

Enforce uniformity in the construction of the local views by constantly checking for the current field's membership in schema and type definitions. When membership occurs, consider the definitions as possible predictors of coming fields, and as templates for generalization and aggregation operations.

Rule 5.

Uniformity is further enforced by checking at the end of a form entry for any schemata which are partially matched. The unmatched fields can pin point weaknesses in the information and discover non-visible fields which in a manual system were done in the user's head, but nevertheless exist.

Rule 6.
Discover generalization possibilities by finding a supertype which contains the elements of the field in question, which has other subtypes which do not conflict with the elements held in common. As an example, figure 4.20 shows vehicle having the elements which are associated with car on the registration form. Car (shown in figure 4.21) inherits these elements from vehicle by restriction and join. Truck (figure 4.22) also inherits these elements, although truck does not appear on the form. Because truck does not conflict in the use of these elements (they are only mentioned in the common supertype), vehicle is a likley generalization candidate.

The following rules are performed only during consolidation.

Rule 7.
Treat two local views at a time like a single view.

Rule 8.
Perform consolidation on the organizations most important views first.

Rule 9.

When identity appears to be found, check for indications that one view might preceed the other operationally. This is an indication of data sharing and thus, identity among elements. Additionally, descriptors of purpose collected during the local view stage can be used to add weight to the decision. Preponderance of evidence is a deciding factor in whether or not to ask for permission to consolidate.

Rule 10.

Generalization works much like rule 9 with the additional need to check usage conflicts. Although employee and customer may both be subtypes of person, they do not necessarily belong together in a generalization, unless there are similarities in usage and no conflicts. The further up the lattice one has to go is an indicator of inappropriateness. The closer the common supertype is, the more likely it is a generalization candidate.

Rule 11.

Aggregation consolidation follows the same guidelines as rule 8. Aggregation should only be considered if the schemata suggest it, as was the case in the local views, and if the other local views in the system do not counterindicate by having overlapping, mutually antagonistic members or usages.

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

Section 3: GENERATION OF TWO LOCAL VIEWS.

A motel operation uses many of the normal business functions. It provides services, requires an inventory, and requires employees. To simplify the following discussion, only two functions of a motel will be analyzed, reservation and registration. In spite of a seeming simplicity, these functions are quite complex and will provide the basis for a strong demonstration of the proposed method by requiring the use of all features of conceptual graphs and by demonstrating with numerous examples the detection of homonyms, synonyms, aggregation and generalization problems.

The first view to be dealt with is the function of the reservation of a room for a customer. The conversion of this view to graph form will be explained in detail using a simulated conversation between the analyst and the design system. The problem will be demonstrated in this way because of the difficulty involved in referring to fields on the forms and the function being performed. In this way, system action will be separated from action by the analyst. Most of the conversation will center on the physical objects which are used in the manual motel system, i.e., forms. The form for a reservation is shown in figure 4.1. The following conversation shows the output from the system in upper case and the response of the analyst in

Figure 4.1 Reservation Form

RESERVATION

Date:
Name: Phone:
Type of room:
Number in Party:
Rate quoted:
Depart: / / Arrive: / / Num. days:
Hold:
Credit card:
Card number: Expiration date:

lower case. The commentary will be shown inside of Pascal comment brackets using mixed case.

WHAT IS THE NEXT FORM?

{At this point in the conversation, the system already knows about the various forms used by the motel and their inter-relationships (Rule 1). What is not known is the contents of each, nor how the various fields relate to information needed in the data base.}

the reservation form.

WHAT IS THE PURPOSE OF FORM:RESERVATION?

it is used to reserve a room for a customer.

{The system can now set up certain inferences about the forthcoming form definition (Rule 2). 'RESERVE' is the keyword in the previous phrase. Using the definition of RESERVE given in figure 4.2, the system expects to find fields or references to the PERSON-BUSINESS who owns the PHYS-OBJECT, the PHYS-OBJECT, the TIME-PERIOD, and the PERSON-BUSINESS who is to possess the object (Rule 4). So far, the system has the motel as the owner, a customer as the recipient, and the motel room as the object. Figure 4.3}

WHAT IS THE FIRST FIELD?

date.

{Three dates are unresolved in the definition of TIME-PERIOD in RESERVE (Rule 4). More information is needed to resolve the ambiguity.}

WHAT IS THE PURPOSE OF DATE:DATE?

{Notice how the system gives a type to the entities. This extra information is used to show that the system is understanding the short answers.}

it is the date the form is filled out.

THEN THE DATE:DATE IS THE CREATION DATE OF THE FORM:RESERVATION?

{The system has made a guess based on the definition of FORM

Figure 4.2 Type definition of RESERVE.

```
type RESERVE(X) is

[GIVE:*x] -

          (AGNT)──→[PERSON-BUSINESS]
          (DUR)───→[TIME-PERIOD] -

                     (STRT)───→[DATE:*y],
          (OBJ)────→[ENTITY]
          (RCPT)───→[PERSON-BUSINESS],

[DATE:*now],

[DATE:*now <*y].
```

Figure 4.3 RESERVE filled in with known information.

```
[RESERVE] -

          (AGNT)——→[MOTEL]
          (DUR)———→[TIME-PERIOD] -

                    (STRT)——→[DATE:*y],

          (OBJ) ———→[MOTEL-ROOM]
          (RCPT)———→[CUSTOMER],

[DATE:*now],

[DATE:*now < *y].
```

which states that for all forms there exists a unique creation date. In addition the date in the constraint for time period is filled in (Rule 4 and join). Figure 4.4}

yes.

WHAT IS THE SOURCE OF DATE:DATE?

{The system asks for source to aid in determining the functional and computational dependencies in the system. In this case, date is a system supplied date and is represented by an actor.}

the date is supplied at creation by the system.

WHAT IS THE NEXT FIELD?

name.

WHAT IS THE PURPOSE OF INFORMATION:NAME?

it is the name of the customer.

WHAT IS THE SOURCE OF INFORMATION:NAME?

the desk clerk.

{The response of "desk clerk" tells the system that this field has an input, not derived, source. Presumably the actors (desk clerk, maid, owner) of the motel are known from previous discussion in the requirements analysis. }

{The system has a schema (figure 4.5) for the business use of a customer's name. It has been satisfied by the input. The generalization consolidation principle is at work here, grouping the three fields under their least common supertype, as in figures 4.6 and 4.7. (Rules 4 and 7)}

MAY I GENERALIZE?

yes

{'yes' should almost always be the response to generalization and aggregation consolidation questions. Just because a conceptual model grows in the number of entities does not mean that the physical storage will also grow. It is dependent only on the physical design.}

NEXT FIELD?

Figure 4.4 Creation date of RESERVATION.

```
[DATE:E^1*creation] -

        (CRTN)——→[FORM:*reservation]
        (SYN)——→[NAME:date] -

                (FRM)——→[FORM:*reservation].
```

Figure 4.5 Schema for NAME.

```
schema NAME(X) is

[PERSON] -

        (NME)——→[NAME:*x] -

                        (PART)——→[FIRST-NAME]
                        (PART)——→[MIDDLE-NAME]
                        (PART)——→[LAST-NAME].
```

Figure 4.6 Intermediate form of NAME.

```
[CUSTOMER:*reservation] -
        (NME)——→[NAME] -

                        (PART)——→[FIRST-NAME]
                        (PART)——→[MIDDLE-NAME]
                        (PART)——→[LAST-NAME].
```

Figure 4.7 Synonyms for name.

```
[FIRST-NAME]———→(SYN)———→[NAME:first]———→(FRM)———↘
[MIDDLE-NAME]——→(SYN)———→[NAME:middle]——→(FRM)———→[FORM:reservation]
[LAST-NAME]————→(SYN)———→[NAME:last]————→(FRM)———↗
```

phone.

WHAT IS THE PURPOSE OF OBJECT:PHONE?

it is the number of the customer's phone.

{Here the analyst has corrected a problem in communication about the purpose of the field PHONE. A schema could now be developed to cover this situation in the future. The plausability of PHONE meaning NUMBER instead of an actual PHONE would be strengthened by usage. Figure 4.8}

WHAT IS THE SOURCE OF NUMBER:PHONE?

the desk clerk.

NEXT FIELD?

type room.

WHAT IS THE PURPOSE OF INFORMATION:TYPE ROOM?

it is the type of motel room the customer wishes to reserve.

{Figure 4.9.}

WHAT IS THE SOURCE OF INFORMATION:TYPE ROOM?

the desk clerk.

NEXT FIELD?

number in party.

WHAT IS THE PURPOSE OF NUMBER: NUMBER IN PARTY?

it tells how many customers will use the room. this is used in determining the rate.

WHAT IS THE SOURCE?

the desk clerk types it in.

{The system has a tag on the field RATE because of its mention above. The expectation will be filled later.}

NEXT FIELD?

rate quoted.

Figure 4.8 Representation for phone.

```
[CUSTOMER:*reservation] -

          (POSS)→[PHONE] -

                    (ATTR)→[PHONE-NUMBER] -

                              (SYN)→[NAME:phone] -

                                        (FRM)→[FORM:reservation].
```

Figure 4.9 Representation for room type.

```
[MOTEL-ROOM:*reservation] -

          (TYPE)→[ROOM-TYPE:*reservation] -

                    (SYN)→[NAME:type room] -

                              (FRM)→[FORM:reservation].
```

WHAT IS THE PURPOSE OF INFORMATION:RATE QUOTED?

it tells how much the room will cost per night.

WHAT IS THE SOURCE?

the rate is determined by the number of occupants and the type of room.

{The system has to deal with two name ambiguities here. Number of occupants is determined to be the same as number in party because of the definitions of occupants and abscence of other verbal clues. Rate, because of a definition of distance per time, is not as clear.}

IS INFORMATION:RATE THE SAME AS INFORMATION: RATE QUOTED?

yes

{Figure 4.10.}

WHAT IS THE NEXT FIELD?

arrive.

{Although ARRIVE is an action verb, the system knows that forms do not record the particular instance of an action. Instead, because an action verb is specified, the system knows to look at the schemata and definition for the verb to predict coming fields and to allow them a place. Potentials enabled by arrive are an actor, a source, a location, and a time (Rule 2).}

WHAT IS THE PURPOSE OF INFORMATION: ARRIVE?

it is the date the customer arrives.

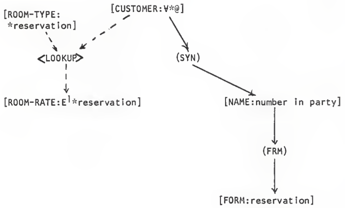
{The system has decided arrive is a time/date field. The question is where the date belongs in the model. Three dates are currently in the model, start, stop, and creation. One of these might correspond to arrival-date. Creation date is ruled out because of its well defined nature.}

IS IT THE START OR THE STOP DATE?

the start date of time-period

{The answer could have been no to both, but the system has been fortunate this time.}

Figure 4.10 Procedural attachment to Room rate and number in party.



WHAT IS THE SOURCE?

the desk clerk

NEXT FIELD?

depart

WHAT IS THE PURPOSE OF DEPART?

it is the date that the customer leaves on.

IS IT THE STOP DATE?

yes.

WHAT IS THE SOURCE?

the desk clerk

NEXT FIELD?

total.

WHAT IS THE PURPOSE OF NUMBER:TOTAL?

it is the number of days the customer wishes to occupy a room.

WHAT IS THE SOURCE?

it is the difference between the start and stop dates.

{The system creates figure 4.11 as its understanding of the problem. Note that no mention is made about the implementation of the date-difference function. It is only known that it exists. The generality allows the design to proceed unhindered by implementation details.}

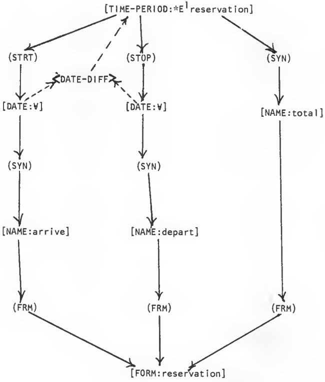
NEXT FIELD?

hold

WHAT IS THE PURPOSE OF INFORMATION:HOLD?

{Although hold is a verb, the definition of the act of an agent possessing an object does not offer much help in making preliminary inferences. Another inference that can be made about verbs is that they can be used as boolean in condition-action

Figure 4.11 Representation of TIME-PERIOD.



pairs where the verb is involved in the action.}

if the room is held, then it is reserved for the entire date of arrival. if not, then the reservation is terminated at 7pm on the day of arrival.

WHAT IS THE SOURCE?

desk clerk.

WHAT IS THE NEXT FIELD?

credit card

WHAT IS THE PURPOSE OF VALUE-OBJECT:CREDIT CARD?

if the room is held and the customer does not arrive by the arrival date + 1 then the card is billed at the quoted rate for one day. if the room is not held the credit card serves nothing.

WHAT IS THE SOURCE?

desk clerk

{The analyst has overlooked that the type or name of the credit card is to go in the preceeding field. Because the definition of credit card exists in the system, however, this oversight will be discovered (Rule 5).}

WHAT IS THE NEXT FIELD?

date.

WHAT IS THE PURPOSE OF INFORMATION:DATE?

{The model has acquired a new, unfilled DATE concept from the definition of CREDIT-CARD.}

it is the expiration date of the credit card.

{Suspicion about the date confirmed!}

WHAT IS THE SOURCE?

the desk clerk.

WHAT IS THE NEXT FIELD?

card number.

IS THIS THE CREDIT CARD NUMBER?

{The system has strong evidence to support filling the NUMBER slot (Rule 4).}

yes.

WHAT IS THE SOURCE?

the desk clerk.

WHAT IS THE NEXT FIELD?

that was the last one.

ALL FIELDS ARE ENTERED?

{The system has to confirm that the last answer was a sentence and not a field.}

yes

{The system now goes through the form in LIFO order, requesting more information on the unfilled slots and missing functional dependencies.}

{The first problem is that the system is unresolved with the CREDIT CARD field and the slot CREDIT CARD:TYPE (Rule 5).}

DOES THE CREDIT CARD FIELD REPRESENT THE CREDIT CARD:TYPE?

yes.

{This is an aggregation example. The three fields are related by function, but not by type (Rule 4).}

MAY I COMBINE INTO FIGURE 4.12?

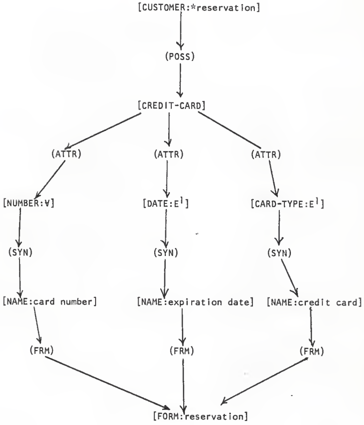
yes

{It is important to have the functional dependencies by which a form is retrievable. These are gathered now, because all fields on the form are known.}

WHAT FIELDS UNIQUELY DETERMINE FORM:RESERVATION?

name, arrival, departure, and type of room.

Figure 4.12 Representation of CREDIT CARD.



NEXT FORM?

guest registration

{This form is shown in figure 4.13. (Rule 1).}

WHAT IS THE PURPOSE OF FORM:GUEST REGISTRATION?

it is used to rent a room for a guest.

{The system selects the MOTEL-REGISTER verb (figure 4.14) based on two keys of register and rent (Rule 2). Alternatively, if no MOTEL-REGISTER verb were present, the verb RENT could be used as well. Figure 4.15.}

WHAT IS THE FIRST FIELD?

last name

WHAT IS THE PURPOSE OF NAME:LAST NAME?

to record the name of the guest

{having the outline for MOTEL-REGISTER, the system begins to clarify some of the slots (Rule 2).}

DO YOU MEAN CUSTOMER?

yes

{The system knows two locations for a person subtype, AGNT and RCPT. AGNT is filled with the default MOTEL, which disallows guest filling in here. The other possibility is CUSTOMER}

WHAT IS THE SOURCE?

the desk clerk

NEXT FIELD?

first name

PURPOSE?

it is the first name of the customer.

Figure 4.13 Registration form.

GUEST REGISTRATION

____________________/____________________/__________
Last name First name Middle

____
Street

____
City

_______________________/______________________
State ZIP

Number of persons:______

Make of car:______ License No.:__________ State:__________

Unit number:_______

Rate:______

Date in:___/___/___

Date Out:___/___/___

Days Occupied:_______

Total Days:_________

Total:______________

Extras:____________

TAX: ____________

PAY: ____________

Figure 4.14 Type definition for register.

```
type MOTEL-REGISTER(X) is

[RENT:*x] -

          (AGNT)——→[MOTEL]
          (OBJ) ——→[MOTEL-ROOM]
          (RCPT)——→[CUSTOMER]
          (COST)——→[MONEY:*@]
          (DUR)———→[TIME-PERIOD].
```

Figure 4.15. Type definition for RENT.

```
type  RENT(X) is
[GIVE:*x]  -
          (OBJ)→[ENTITY]
          (DUR)→[TIME-PERIOD]
          (AGNT)→[PERSON-BUSINESS:*a]
          (COST)→[MONEY:*@b]
          (RCPT)→[PERSON-BUSINESS:*c],
[GIVE]  -
          (RCPT)→[PERSON-BUSINESS:*a]
          (OBJ)→[MONEY:*@b]
          (AGNT)→[PERSON-BUSINESS:*c].
```

SOURCE?

the desk clerk

NEXT FIELD?

middle initial

WHAT IS THE PURPOSE OF INFORMATION:MIDDLE?

it is the initial of the middle name

{On some words the system may not understand (have a definition of), it may solicit more information.}

WHAT IS 'INITIAL'?

it is the first letter of something.

THANK YOU. WHAT IS THE SOURCE?

the desk clerk

{This is the same generalization problem encountered in the reservation example because the three names, although they are of the same type, are treated differently. The system contains a schema definition for NAME which belongs to a PERSON, shown in figure 4.5 (Rule 4). Because CUSTOMER is a subtype of PERSON the system can restrict PERSON to CUSTOMER in 4.5 and then join to form 4.16. It must ask permission though, since it is not always appropriate to consolidate in this way. The consolidation is triggered because all three fields have been matched. An important point about the existance of schema to trigger generalization consolidation is that it enforces uniformity in the design.}

MAY I COMBINE 4.5 INTO 4.16? {Principle of restrict/join}

yes

NEXT FIELD?

street

WHAT IS THE PURPOSE OF INFORMATION:STREET?

it is the street address where the customer lives

SOURCE?

Figure 4.16 Representation for customer name.

```
[CUSTOMER:*registration] -

        (NME)→[NAME:*] -

                              (PART)→[FIRST-NAME] -

                                       (FRM)→[FORM:registration],
                              (PART)→[MIDDLE-NAME] -
                                       (FRM)→[FORM:registration],
                              (PART)→[LAST-NAME] -
                                       (FRM)→[FORM:registration].
```

the desk clerk

NEXT FIELD?

city

WHAT IS THE PURPOSE OF INFORMATION:CITY?

it is the city where the customer lives.

SOURCE?

the desk clerk.

NEXT FIELD?

state

WHAT IS THE PURPOSE OF INFORMATION:STATE?

it is the state where the customer lives

SOURCE?

the desk clerk

NEXT FIELD?

zip code

WHAT IS THE PURPOSE OF NUMBER:ZIP CODE?

it is the zip code for the address of the customer.

SOURCE?

the desk clerk

{This is an aggregation problem. The unifier in this case is address. The action is again keyed by the filling of the available address slots (Rule 4).}

MAY I COMBINE 4.17 WITH 4.18 TO OBTAIN 4.19?

yes

NEXT FIELD?

Figure 4.17 Representation of address.

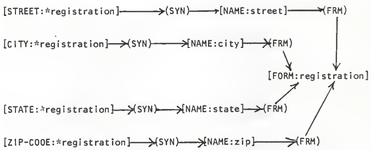


Figure 4.18 Schema for address.

```
schema ADDRESS(X) is
[ADDRESS:*x] -
          (PART)→[STREET]
          (PART)→[CITY]
          (PART)→[STATE]
          (PART)→[ZIP-CODE].
```

Figure 4.19 Combined form of address.

```
[CUSTOMER:*registration] -
          (ATTR)→[ADDRESS] -
                              (PART)→[STREET]
                              (PART)→[CITY]
                              (PART)→[STATE]
                              (PART)→[ZIP-CODE].
```

number of persons

WHAT IS THE PURPOSE OF NUMBER:NUMBER OF PERSONS?

it is the number of persons who will use the room. it is part of the rate calculation.

SOURCE?

the desk clerk

NEXT FIELD?

make of car

WHAT IS THE PURPOSE OF INFORMATION:MAKE OF CAR?

it is the type of car the customer drives.

{There is confusion here because the system has a schema for VEHICLE which has several slots. MAKE and TYPE are not among them. Figure 4.20. (Rule 4, uniformity)}

I UNDERSTAND MANUFACTURER, COLOR AND STYLE OF CAR. DOES IT CORRESPOND TO ONE OF THESE?

yes, to maufacturer, color and style.

MAY I SEPARATE?

{In the interest of uniformity, the system tries to keep all concepts translating to the same form.}

yes

SOURCE?

the desk clerk

NEXT FIELD?

license

WHAT IS THE PURPOSE OF INFORMATION:LICENSE?

it is the number of the license plate on the car. it is used to identify the car in case of problems.

{this is another example where extra information should be

Figure 4.20 Schema for vehicle.

```
schema VEHICLE(X) is

[VEHICLE:E^1*x] -
          (ATTR)→[STYLE]
          (ATTR)→[MANUFACTURER]
          (CHRC)→[COLOR]
          (PART)→[LICENSE-PLATE] -
                              (CHRC)→[NUMBER:∀]
                              (CHRC)→[STATE:∀].
```

Figure 4.21 Type definition for car.

```
type CAR(X) is
[VEHICLE:*x] -
          (PART)→[WHEEL:@4],
¬ [[VEHICLE:*x]→(PART)→[TRUCK-BED]].
```

Figure 4.22 Type definition for TRUCK.

```
type TRUCK(X) is

[VEHICLE:*x] -

          (PART)→[TRUCK-BED].
```

broken down into its component pieces. Also the use of the phrase "used to identify" shows the functional dependency of CAR on LICENSE NUMBER. CAR is not yet a part of the model, however, it is another form of the aggregation consolidation problem (Rule 4).}

DOES LICENSE CORRESPOND TO LICENSE NUMBER?

yes

SOURCE?

the desk clerk

NEXT FIELD?

state

WHAT IS THE PURPOSE OF INFORMATION:STATE?

it is the state the license is issued from.

SOURCE?

the desk clerk

NEXT FIELD?

unit number

WHAT IS THE PURPOSE OF NUMBER:UNIT NUMBER?

it is the number which identifies the room rented to the guest

SOURCE?

the desk clerk

{The system knows that vehicle is not yet filled out in schema as much as it could be. It will wait untill all fields are in before reporting on the chance that not all fields on the form that concern car have been reported (Rule 5).}

NEXT FIELD?

rate

WHAT IS THE PURPOSE OF INFORMATION:RATE?

it is the price of the room per night.

SOURCE?

it is calculated based on the room type.

{The system can find no reference to ROOM-TYPE; assumes it is coming.}

NEXT FIELD?

date in

PURPOSE OF DATE:DATE IN?

it is the arrival date of the customer.

SOURCE?

the desk clerk

NEXT FIELD?

date out

WHAT IS THE PURPOSE OF DATE:DATE OUT?

it is the departure date of the customer.

SOURCE?

desk clerk

NEXT FIELD?

days occ.

WHAT IS THE PURPOSE OF INFORMATION:DAYS OCC.?

it is the number of days the room will be occupied.

SOURCE?

it is the difference between date out and date in.

NEXT FIELD?

total days

PURPOSE OF INFORMATION:TOTAL DAYS?

it is the number of days the room is occupied

SOURCE?

it is the difference between date out and date in.

{The structure for this field is equivalent to the structure for days occ. By rule 3, these turn out to be synonyms}

WHAT IS THE DIFFERENCE BETWEEN IT AND DAYS OCC.?

none.

THEN IT IS REDUNDANT?

yes.

{Figure 4.23}

REMOVED!

NEXT FIELD?

total

WHAT IS THE PURPOSE OF INFORMATION TOTAL?

it is the amount of money the customer must pay for the room.

WHAT IS THE DIFFERENCE BETWEEN IT AND RATE?

rate is per day cost, total is all days combined.

SOURCE?

it is calculated: total days * the rate.

{Figure 4.24}

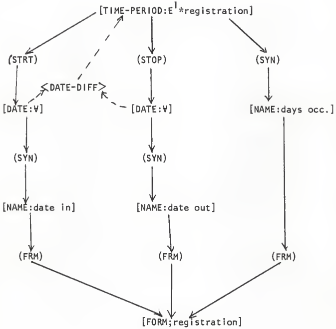
NEXT FIELD?

extras

PURPOSE OF INFORMATION:EXTRAS?

it is any charges not counted in the total.

Figure 4.23 Registration time period.



SOURCE?

desk clerk.

NEXT FIELD?

tax

WHAT IS THE PURPOSE OF TAX:TAX?

tax is the amount to charge for tax purposes.

SOURCE?

{A non visible field is "discovered" here. A pre tax total and a tax rate are used to calculate the tax. The system does not know if they are fields or not defined until all other fields are in (Rule 5).}

tax is (total + extras) * tax-rate

NEXT FIELD?

pay

PURPOSE OF WAGE:PAY?

it is not a wage, it is a cost. It is the total of all costs.

{System not clear on what is happening, but the source will clear things up.}

WHAT IS THE DIFFERENCE BETWEEN TOTAL AND PAY?

it includes the total.

SOURCE?

it is the total + the extra + tax

{Figure 4.25}

NEXT FIELD?

done!

{Some things have been left hanging, namely the missing room-type and generalization on the car (Rule 5).}

Figure 4.24 Source of money (TOTAL).

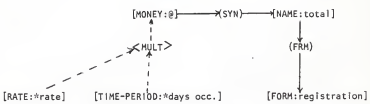
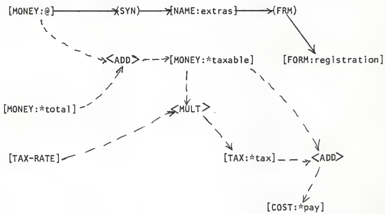


Figure 4.25 Calculation of pay.



WHEN YOU ENTERED RATE, YOU MENTIONED ROOM-TYPE. WHERE DOES ROOM-TYPE FIT?

it is calculated but not entered.

HOW IS IT CALCULATED?

it is the result of a function on unit-no.

{this is another way the system discovers hidden, yet very real, information. Figure 4.26}

{Rule 6 is at work in the following example.}

THE INFORMATION YOU HAVE GIVEN ME FOR CAR FITS NICELY INTO THE MORE GENERAL CATEGORY OF VEHICLE. WOULD YOU LIKE TO REPLACE CAR WITH VEHICLE AND INCLUDE TRUCK IN THIS FIELD?

yes.

WHAT FIELDS UNIQUELY DETERMINE AN INSTANCE OF THIS FORM?

customer, date in and unit number.

For the purposes of this paper, the local views are now defined. In an actual design environment, much more information would likely be collected. This information would include domains of the fields, which could be general like NUMBER, or contain a specific set of members, as in a Pascal enumerated type. The information could also include update authority, integrity constraints, and condition action sequences. More was not collected in these examples because it is not necessary for solving the three consolidation problems.

Figure 4.26 Representation for motel room.

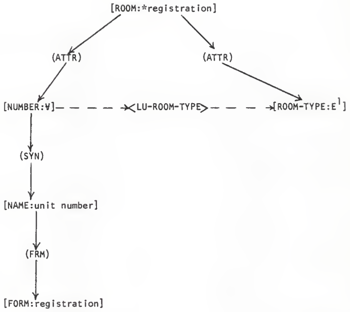
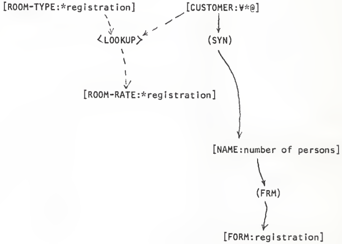


Figure 4.27 Representation of rate and number of persons.



AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

SECTION 4: CONSOLIDATION OF THE LOCAL VIEWS

Consolidation is the next step in the design process. Consolidation normally takes place in order of most important views first and is done pairwise [TEOR82]. Because this example deals with only two local views, these considerations are irrelevant. The views will be compared for the three types of consolidation, identity, aggregation, and generalization, but no attempt at resolving any conflicts will be made. The resolution is the job of the analyst and other system rules.

Some of the fields in the local view are immediately seen as candidates for identity consolidation. Comparing figure 4.1 to 4.13, the following fields are potential candidates:

RESERVATION	figure	REGISTRATION	figure
date out	4.11	depart	4.23
date in	4.11	arrive	4.23
num. days	4.11	days occ.	4.23
rate quoted	4.10	rate	4.27
number in party	4.10	number of persons	4.27

The consolidation on these data items is straightforward and follows from rule 9. A visual inspection of the conceptual graph representations for these items shows equivalent forms. The SYN relation and the referents are ignored in the comparison process to separate syntax from semantics. One pair of fields does not have as strong of structural similarities, "number in party" and "number of persons." For these to be detected requires accessing the representation of the analyst's definition. The definitions have all been retained as information but are not needed as often in the identity consolidation process. "Number in party" was defined with "it tells how many customers will use the room...," and "number of persons" was defined with "it is the number of persons who will use the room..." These sentences are syntactically different, but semantically the same. Owing to the translation process, the internal representations of the two are equivalent. If the field had been more complex, the analyst's explanation may have been different enough to foil the detection of potential identity. The analyst's role plays a large part in determining the success of the consolidation, but the system still relieves much of the design analysis burden.

The problem of homonyms occurs during identity consolidation. Because semantic information is collected, the problem of syntactic equivalence between different data items is not important. There should be no restriction against the same name existing in separate views, as long as context makes it clear to anyone using the fields what their meanings are. Semantic homonyms are a different matter. Structures which appear the same, such as two different time-periods, should not be considered for identity consolidation. The same clues which are used to determine synonyms are used to rule homonyms out. Context is important;; going further out in the items which are associated with the concept in question should answer some questions. If the concept is procedurally derived from other concepts, then checking the definitions of the constituents is excellent practice; they must be equivalent. Information about how the forms are used can be a deciding factor, dissimilar forms are more likely to have dissimilar concepts. In the end, some quantitative measure might be the best measure. The confidence level could be reported to the analyst with the final decision coming from him. Even expressing indecision about the consolidation is better than the pair going unnoticed into the physical design where data integrity is dependent on minimizing the data redundancy.

The potential for aggregation consolidation is detected when different associations for equivalent concepts are discovered (rule 11). From the local view of reservation comes figure 4.28, which shows the aggregation members collected under customer. Figure 4.29 shows the same for the registration view. These views have the name and address concepts in common. Credit card and phone of reservation and car of registration are the differences. It is these differences which are candidates for consolidation. The question is whether they should be. For the physical model it is important that storage be efficient in space and access time, which could mean storing all of the five subgroups together in one record, even though for the local views, car would only be visible to registration. For the conceptual model, however, the important point is that all

Figure 4.28 Aggregation of customer from reservation.

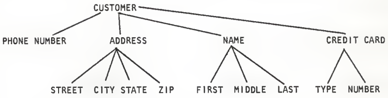
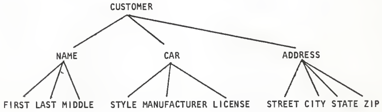


Figure 4.29 Aggregation of customer from registration.



information which might need to be included in a local view, be included in a local view. For example, it might be best to include credit card and phone in the registration in case damages or valuables were discovered after checkout, while it seems pointless to include car information in the reservation view. The system must give the designer the opportunity to choose or have higher level decision making power available. In either case, the system must detect the potential for aggregation. Presumably, once the potential is detected, the model can be marked for later reference during the conversion to the conceptual schema.

Generalization potential is detected when there is a general category, such as name, to which similar elements belong, such as first, last and middle names (rule 11). This situation was seen several times during the collection of the local views. The need for generalization is indicated by a concept's location in the type lattice. The more specific a concept (and the more it participates in the parent role of other generalizations and aggregations), the more likely a candidate it is for generalization. Just as strong an indication of the need for generalization is the existence of a not far removed supertype which has many attributes while its subtypes have few of their own. This case was seen in figures 4.20, 4.21 and 4.22. Another example is employee which has many attributes, together with all of the possible subtypes of employees such as truck driver, janitor and executive. It may be desirable to either supress these occupations distinguishing features or generalize them as pay type and job code. No candidates for generalization consolidation exist in the local views, but the process of deciding to consolidate or not is very similar to the preceeding aggregation discussion. Under manual methods, generalization would have taken place during consolidation on the concepts name and address.

Unlike manual methods, much of the work associated with consolidation is performed during collection of the local views. The work can be done because the schema and type definitions act as a different local view with which the view in question may be consolidated. Performing the bulk of the consolidation work at the time local views are collected has many benefits. It spreads the consolidation workload out across time. Immediate feedback is given at the time of local view definition; the same designer is present to be questioned, which is not the situation in the manual case. The same immediacy pays off because a problem in one view that is not detected until consolidation time can cause the entire process to be halted until a solution is found.

The need for a consolidated model has not fully been removed, but the representation has changed. Instead of a huge intertwined conceptual graph, the referents on the concepts are changed to show sameness and non sameness. The customer in registration and reservation could both become CUSTOMER:*motel.

A restaurant customer would remain CUSTOMER:*restaurant. It is better to represent the consolidated model in this way because abstraction and modularity, which are widely recognized in software engineering as desirable, meaning enhancing concepts, are preserved.

Chapter 5
Summary of Results

In chapter 2, the guidelines for the thesis were laid out. The success of the thesis depends on the satisfactory completion of these guidelines. In addition to fulfilling the guidelines, some results which were not expected were realized. This chapter is organized in three sections, results contracted, contributions, and new directions.

SECTION 1: Contracted Results

This section will address each of the objectives in chapter 2, stating to what degree each has been completed.

1. Select a language for representation.

The selected language is SOWA84's conceptual graphs. It was selected on it's readability, modularity, and hoped-for completeness with respect to 2 below. The pertinent features are described in chapter 3.

2. Demonstrate the seven points by showing examples of each in the representation of local views.

This point is formally addressed in chapter 3. Numerous examples are given in section 3, chapter 4, the local views. The language fully satisfies the seven criteria. The figures which represent each of the seven points are listed below:

Point	Figure
1. Type Lattice	3.1
2. Functional dependencies	3.9, 3.10, 3.11
3. Domain Roles	3.9, 3.10, 3.11
4. Type definition	3.3, 3.4
5. Schemata	3.6, 3.7, 3.8
6. Procedural Attatchment	3.12
7. Inferences	3.13 - 3.15 and 3.17 - 3.20

3. Demonstrate the language's ability to represent local views by generating local views from English-like descriptions such as might be collected by the design analysts.

This point is satisfied in section 3, chapter 4. Not only were local views generated from English like descriptions, but also from business forms. First a set of rules was developed to govern generating the local views. These rules depend heavily on the existence of a natural language interface which maps semantically equivalent, differing syntax surface expressions to the semantic expressions which have the same expressions in conceptual graphs. This ability was proven by SCHA77. Another component that this step is dependent on is a dictionary of business terms. A more complete discussion of the generation of local views can be found in sections 2 and 3 of chapter 4.

4. Develop a preliminary (possibly incomplete) set of rules to detect synonyms between (or even within) the various local views.

This point is well satisfied in section 2 of chapter 4. No attempt was made to prove completeness, and it is doubtful that completeness can be established due to the diversity inherent in database designs. The use of each of these rules is noted where it occurs in section 3, chapter 4.

5. Develop a preliminary (possibly incomplete) set of rules to detect homonyms between (or even within) the various local views.

This point is satisfied with an incomplete set of rules in section 2 of chapter 4. Homonyms at the syntactic level are eliminated by the rules governing local view generation. Semantic homonyms (equivalent representational forms for non-equivalent concepts) are covered by rules which consider the context. Again the point of application in section 3 and 4, chapter 4 is noted.

6. Devise a way to represent the consolidated, integrated model.

The consolidated representation using referents in the local view is covered in section 4, chapter 4. Concepts which are meant to be the same, even though they appear in different views are given unique referents. The customer in figure 4.16 is the same customer as the one in figure 4.19 because of the registration referent. The decision to not come up with an integrated model was made in the interest of readability, modularity, abstraction, and hence, efficiency.

7. Develop a preliminary (possibly incomplete) set of rules for consolidating the local views into an integrated model.

This criterion is satisfied mostly by the rules governing the construction of local views. Other rules are noted in the later half of section 2, with their application noted in section 4, chapter 4. By putting more effort into the local views, less effort was needed during consolidation. The rules governing each are discussed in section 2, chapter 4.

8. Identify the synonyms and homonyms already present in the local views for use in validating task 9.

This step is outlined in section 4, chapter 4, just prior to consolidation.

9. Using the three sets of rules above, perform the consolidation, showing the point of synonym and homonym detection.

The consolidation was performed in three separate steps, identity, aggregation and generalization consolidations. The process is written up in section 4, chapter 4. The summary of the results follows point 10 below.

10. Validate task 9 with the output of task 8.

All known inconsistencies were discovered. More could possibly exist, but were not discovered. The generalization potential of car (figure 4.20-4.22, in the registration view was not suspected by the author until analysis for the local view was performed. Analysis on other fields strengthened the author's initial assumptions about the suspected inconsistencies. A weakness was found concerning concepts which do not have strong structural associations with other concepts. In this case, the definitions by the analysts had to be referenced. This reliance on the analyst is the weak point because of the potential for greatly different definitions. Nevertheless, enough of the other consolidation potentials were discovered to make the system worthwhile, especially since the limitation is a lesser version of one that exists in the manual system. The potential for this weakness in the conceptual graph structure is not very great because of the independence of the model from surface expression. Both points could be bolstered by taking more input during the local view collection.

In conclusion, the feasibility of using CG as the basis for an computerized design tool and the feasibility of the design tool is demonstrated by the satisfaction of the above criteria.

Section 2: Contributions

The promised contribution (as described in section 1, chapter 5) of this work is the demonstration of feasibility and development of a preliminary methodology for machine aided data inconsistency avoidance. In addition, several other contributions were realized.

The desirability of analyzing local views from an action verb point of view contributes to design methodology. The benefits of doing this are that there is a measurable standard against which to check the design and the associated concepts of the verb can help suggest design approaches in analyzing the functions of the view. This benefit is especially true because it relies on a language system which is very familiar to all analysts, English. Traditional methods of design analysis emphasize data or function analysis. Verb analysis is close to function analysis, but comes with built in data needs and usually breaks functions down into primitive components; a much easier level to deal with because of simplicity.

The other main contribution of this work is the discovery that many of the traditional consolidation inconsistencies are eliminated during collection of the local views. The situation is analogous to the cost savings achieved from finding program errors in the design stage instead of the programming stage. The reasons for the inconsistency avoidance are given in section 4, chapter 4.

Section 3: New Directions

The most obvious work left undone is the construction of a prototype of the proposed system described in chapter 1. Some of the components involve little more than adapting current programs, while others require in-depth analyses and testing. One place to start would be the development of a more complete rule set for the generation of local views. Another direction would be to take a language which has a strong natural language processing background like Schank's conceptual graphs and extend it to satisfy SOWA84's criteria. In this way, advantage could be taken of the many existing software tools.

An unrelated activity would be to test the hypothesis that verb analysis is superior to other forms of analysis. The verbs dealing more with abstract entities and functions may prove more difficult.

Another application would be to test the feasibility of this system for the design of procedure intensive applications. The framework of C.G. seems to handle procedure description as well as data description.

An expected, though not immediate contribution of this work is the protection of intellectual investment. Because the proposed system is dynamic, each organization can start with a minimal system and "educate" it on their style and policy. Analysts who have worked for the company for years can introduce their style into the system and have it be enforced, while it educates newer analysts.

VII Bibliography

[BRACH80] Brachman, R. and B. Smith. "Special Issue on Knowledge Representation" in SIGART, No. 50, Feb. 1980.

[CHEN76] Chen, P. P. S. "The Entity-Relationship model -- Toward a unifeid view of data." ACM Trans. Database Systems 1:1 (1976).

[CODD80] Codd, E. F. "Data Models in Database Management" in [Zill80]

[HAMM81] Hammer, M and D. McLeod. "Database Description with SDM: a Semantic Database Model." ACM Trans. Database Systems 6:3 (1981).

[HUBB79] Hubbard, George U. Computer Aided Design Vol 13, No. 3. May 1979 IPC Business Press Ltd.

[JARD77] Jardine, Donald A. editor. The ANSI/SPARC DBMS Model. North Holland Publishing Co. New York (1977)

[MYLO80] Mylopoulos, John, Phillip A. Bernstein, and Harry K. T. Wong. "A Language Facility for Designing Database-Intensive Applications" in ACM Transactions on Database Systems, 5:2 (1980).

[SCHA77] Schank, Roger and Robert Abelson. Scripts Plans Goals and Understanding. Lawrence Erlbaum Associates, Inc. 1977.

[SCIO79] Sciore, Edward. "Improving Semantic Specification in a Relational Database" in Data Base Management ACM (1979)

[SMIT77] Smith, J. M. and D. C. P. "Database Abstractions: Aggregation and Generalization" in ACM Transactions on Database Systems, Vol 2, No. 2, 1977.

[SOWA80] Sowa, J. F. "A Conceptual Schema for Knowledge Based Systems" in ZILL80.

[SOWA84] Sowa, J. F. Conceptual Structures. Addison-Wesley 1984.

[STUR83] Sturzda, Paltin. "From Data Base to Knowledge Base, Artificial Intelligence with an IBM Data Dictionary" in Intnl. Conf. Computer Capacity Management (5th, 1983 New Orleans).

[TEIC77] Teichrow, Daniel and Ernest A. Hershey, III. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Transactions on Software Engineering. Vol. SE-3, No. 1. January 1977.*

[TEOR82] Teory, Toby J. and James P. Fry. *Design of Database Structures*. Prentice *Hall, Inc.* 1982

[ZILL80] Zilles, Stephen N. and Michael L. Brodie, eds. *Workshop on Data Abstraction, Databases and Conceptual Modelling*. Association for Computing Machinery, 1980.

[ZILL80a] Zilles, Stephen N. "Workshop Summary" in ZILL80

AVOIDING DATA INCONSISTENCY PROBLEMS IN THE CONCEPTUAL DESIGN OF DATA BASES: A SEMANTIC APPROACH

by

DAVID ELDEN LEASURE

B. A., Kansas State University, 1982

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Abstract:

Many problems occur in data base design during consolidation of local views. Among these problems are the failure to detect synonyms and homonyms in the entities, to detect aggregation inconsistencies and to detect generalization inconsistencies. Many authors argue for a semantic representation language to serve as a basis for machine assisted consolidation, but no suitable representation has been demonstrated. SOWA78 states that a representation scheme which includes a type hierarchy, functional dependencies, domain roles, definitions, schemata, procedural attachment, and inference mechanisms is sufficiently rich to model data base semantics. This paper argues that such a representation scheme can also serve as the representation language for a data base design system which collects local views and performs consolidation while avoiding data inconsistency problems.