ADDSS: A system for automatic database schema design based on the binary-relationship model

Peretz SHOVAL and Moshe EVEN-CHAIME
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer Sheva, 84105 Israel

Abstract. This paper presents the system ADDSS that has been developed to assist the database designer designing a database schema. A distinction is made between the stage of information structure analysis in which the information structure of the system is defined according to its user information needs, and the stage of database schema design in which the record types of the database and the relationships between them are defined. In the first stage a conceptual schema is obtained, represented as an information structure diagram (ISD), and in the later stage the ISD is used to derive the database schema in the form of a data structure diagram (DSD).

ADDSS automatically creates the database schema out of a conceptual schema which is expressed as an ISD of the binary-relationship data mode. The resulting schema consists of normalized record types, according to the relation model, along with hierarchical/set relationships between 'owner' and 'member' record types, as in the CODASYL/Network model. ADDSS applies algorithms to convert the conceptual schema into the database schema. It is implemented on a micro-computer under MS-DOS using dBASE III.

Keywords. Binary-relationship model, conceptual schema, database schema design, data structure diagram, information structure analysis.

1. Introduction

A data model is an abstraction device which enables to view, define and understand the data structure of a certain reality. A data model is composed of a set of concepts and constructs which are used to describe the contents of data and the relationships among them within the database. This description is termed schema. A schema describes both the types of facts included in the database and the rules or constraints by which those facts are allowed to exist [19].

The ANSI/SPARC report [20] introduced a 3-schema framework for database architecture in which the conceptual schema comprises a central description of the various contents that may be in a database. The ISO/TC97 report [21] made a distinction between two principal purposes of the conceptual schema:

(a) to describe the universe of discourse ('enterprise model'), and
(b) to control the descriptions in the information base ('information/data base model').

The first purpose implies that the conceptual schema is formulated independently of computer implementation, whereas the second purpose requires that the conceptual schema formulation be directed towards the computer oriented data structures. This led to a distinction between two levels in that schema, each of which is dedicated to one of the purposes [15]:

0169-023X/87/$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)
The top level is the conceptual schema (CS), a result of the information analysis stage of system development. The CS can be described by an information structure diagram (ISD). The conceptual primitives of the CS and its ISD are not of the computer world. They consist of elements such as object types or entities, attributes, roles, relationships, and so on—depending on the semantic model used for conceptualization.

The second level is a database schema (DS) whose purpose is to control the description in the information base. It has to translate the CS computer-free description into computer oriented data structures and constructs. The DS can be represented via a data structure diagram (DSD). Its vocabulary consists of terms such as record types, fields, keys, and relationships between record types.

At the next level there are the (original) three-schemata which are DBMS specific descriptions, dependent on the DBMS model, and expressed in the specific data definition language. Now, the original conceptual schema of ANSI/SPARC is just a central description of the data structures and constructs (e.g. record types, fields, domains, sets etc.) as implemented in the specific DBMS.

The above classification implies a 5-schema framework for database architecture of which the original ANSI/SPARC 3-schemata are at the DBMS implementation level, and on top of them there are the above conceptual schema, at the analysis level, and the database schema, at the design level. Figure 1, which is taken from [15], summarizes the above terminology and classification. It shows the three main stages in information system development, the data oriented activities in each stage, the results of the activities in terms of schemata, and their representation.

The above introduction clarified the roles and purposes of the various schemata in system development, emphasizing the importance of and the difference between the conceptual schema and the database schema. In this paper we describe the software system termed ADDS that was developed by the authors to automatically generate a database schema out of a conceptual schema. The system accepts as input a description of the CS that is created in the analysis stage. This CS is of the binary-relationship model which is implemented.

![Fig. 1. Stages in database design (source: [15]).]
according to the information analysis (IA) methodology that was developed by Nijssen and others (see [14] and [22]). ADDS uses this CS to produce a DS which is composed of normalized record types, or relations, according to the relational data model [4, 5], and of hierarchical/set relationships between ‘owner’ and ‘member’ record types, as in the CODASYL/Network model [18]. The purpose of ADDS is to serve as a decision support tool for the database designer. It was developed within the framework of a more extensive project which involves the development of several integrated ‘software engineering’ tools that are intended to automate all the stages of system analysis and design. Among those tools is a system for information structure analysis [7] and a system for architectural design [16].

The rest of this paper is structured as follows: Section 2 outlines, in brief, some of the principles of the binary-relationship model and the IA methodology on which ADDS is based, and it describes the technique by which the database schema can be derived from the conceptual schema. In Section 3 the ADDS system is presented. After a general overview of the system its various functions and steps are detailed along with an example. In Section 4 we summarize and outline some further research and development issues.

2. The binary-relationships model and the transition from a conceptual schema to a database schema

The binary-relationship (BR) model and the IA methodology for creating a conceptual schema are described, amongst others, in [14, 15, 21, 22]. The BR model can be viewed as a type of a spectrum of entity-relationship (ER) models [1]. In [2] Chen proposed a framework for classifying ER models, making a distinction between two major types of models: GERM (general ER models), which enables one to define n-ary relationships, and BERM (binary ER models), which is restricted to binary relationships only. According to his framework the BR model is classified as ‘BERM without attributes’.

The BR model has been chosen as the basis for developing ADDS for several reasons. This model has a simpler structure than the general ER model—there are two concepts only: entity (i.e. object type) and binary relationships—thus avoiding the confusion that one may find in the ER model in distinguishing between things which are entities and things which are not, or between facts which are relationships and facts which are attributes [11]. The BR model is connected with the IA method for deriving a conceptual schema from ‘simplified’ natural language sentences (as will be shown). Moreover, we have connected and integrated the IA method with other methods for systems analysis and design [8, 17]. By this integration the use of ADDS is not limited just to database design, but it can be incorporated within the other activities and stages of system development.

In this section we only mention and show examples of some of the important elements of the BR model and the IA method. We use a variant of the model termed ‘nested BR’ as described in [15]. According to the IA method, the development of a conceptual schema starts with ‘examples’ of user information needs (such as reports and forms) which are expressed in natural language. A semantic analysis process applied on the natural language sentences results in ‘elementary sentences’, each of which is composed of two ‘object-types’ (OT) related to each other via two ‘roles’. An OT may have one or more ‘object-type-name’ (OTN) which is a way the OT is referenced or sometimes identified. The type of relationship (or dependency) between the two OTs in a sentence may be 1:1, 1:n or m:n. An m:n relationship may in itself be defined as a (new) OT and be related to another OT. In such a case ‘nested’ elementary sentences are created.
Elementary sentences can be described diagrammatically using a simple set of symbols: a circle denotes an OT and a rectangle denotes a role of an OT. The OTN is written (in parenthesis) within the OT's circle. An OTN which is also an identifier of its OT is underlined in order to distinguish it from other (non-identifying) OTNs. Hence, an elementary sentence is described by two circles representing OTs and two adjacent rectangles representing their two roles. Above these rectangles the type of dependency between the OTs is defined: a 1:1 dependency is denoted by two (separate) lines above the two roles, a 1:n dependency is denoted by a single line above the role of the determinant OT (i.e. the OT which functionally determines the other OT\(^1\)), and an m:n dependency is denoted by a long line above the two roles. When an m:n relationship is by itself defined as an OT, the two rectangles are circled like an OT, thus creating the 'nesting' of sentences.

Figure 2 shows two examples of elementary sentence diagrams. The first diagram shows a 1:n dependency between the OTs Order and Supplier. The relationship is termed Order-To-Supplier. The sentence can be 'read' in two ways: "Order is ordered from Supplier", or "Supplier is supplying Order". The second diagram in that figure shows a nesting of two elementary sentences: the OTs Part and Request are related in m:n dependency, and that relationship, termed Part-in-Request, is by itself defined an OT, and is in turn related (in 1:n dependency) to the OT Quantity.

\[\text{Fig. 2. Examples of elementary sentence diagrams.}\]

\(^{1}\) 1:n dependency denotes a functional dependency between the two OTs. One of them is the determinant, and the other is the dependent.
All the user information needs can be represented in a collection of 'examples' and each 'example' can be analysed, transforming it from a natural language description to a set of elementary (or nested elementary) sentence diagrams. When all the elementary diagrams are combined, an information structure diagram is obtained. (It should be noted that this diagram does not include other possible rules and constraints that a CS may have. A discussion of such constraints is beyond the scope of this paper and can be found in [21] and [22].) To supplement our short description of the IA method, we show in Fig. 3 an ISD which describes the information structure of an organization dealing with employees, projects, parts, requests, orders, suppliers, and so on. This example ISD will serve us for the rest of this paper (to keep it simple we do not show the role names within the rectangles).

We turn now to describe the technique for database schema design. Recall that the sought

Fig. 3. Information structure diagram (ISD).
database schema consists of normalized record types on the one hand, and of hierarchical/set relationships between those record types on the other hand. The combination of the two, as it was described in [15], provides a database schema that can be easily and ‘naturally’ implemented using either a relational or a CODASYL/Network DMBS (at the least). Figure 4 shows the DSD which is the database schema corresponding to the ISD of Fig. 3. This DSD can be derived from the ISD utilizing the two techniques that have been described in [15]. One converts the ISD into a set of normalized relations, and the other generates the hierarchical/set relationships between those relations.

The ISD is converted into a set of schemata of normalized relations by applying a ‘grouping’ algorithm of the IA method. This algorithm examines all sentences in the ISD, each of which is viewed as a binary relation, and it ‘builds-up’ record types by grouping together relations as based on the types of dependencies that are defined between the OTs. The ‘grouping’ algorithm guarantees that no functional and multivalued dependencies violating 4NF are introduced into the created relations. Listed below are the main steps of the algorithm:

A pass on all the OTs of the ISD is performed, and for each OT the types of its dependencies vis-à-vis the other OTs to which it is related are examined:

![Data structure diagram (DSD)](image)

Fig. 4. Data structure diagram (DSD).
- if an OT determines another OT (1:n dependency), then a new record type is defined, the determinant OT's role becoming the key field and the dependent OT's role becoming an attribute (non-key field) of that record;
- if that OT determines more OTs (in other sentences), then the other OTs' roles are grouped as attributes of the same record type;
- if an OT is m:n related to another OT, a new record type is defined in which the roles of both associated OTs become the key fields. If that relationship is also defined a 'nested' OT which is related to other OTs, the above rules are applied depending on the types of dependencies in these relationships;
- if there is a 1:1 dependency, that is, either one of the OTs determines the other one, one of the OT's roles is selected as a determinant and the other one becomes the dependent. Then the relationship is treated as a 1:n dependency. (It will be shown later on in Section 3 that ADDS assists the designer in selecting the preferred determinant OT);
- once these rules are applied to all OTs and relationships in the ISD, the roles which represent fields in the created record types are replaced by their associated OTNs, so that the OTNs become the domains of the fields;
- if an OT has more than one OTN, its OTN which is a unique identifier is selected as the domain of the field in all the record types in which that OT participates, while the other OTNs are added as attribute fields to the record type in which that OT is the key field;
- if an OT does not have an OTN (this is the case with an m:n relationship which is by itself a 'nested' OT), the role is replaced by the OTNs of the associated OTs.

The above algorithm generates normalized relations/record types, but it does not define 'owner' and 'member' types and relationships. This can be achieved using a technique that was developed by Shoval [15] and which is based on the 'essential ISD'. An essential ISD is an ISD which consists of 'key concepts' and the relationships between them, where a key concept is either one of the following:
(a) a determinant OT (i.e. an OT which has at least one dependent OT),
(b) an OT which has more than one OTN,
(c) an OT which is m:n related to another OT,
(d) a 'nested' OT (i.e. an m:n relationship),
(e) one (arbitrary) OT of a 1:1 relationship.
Hence, the essential ISD consists of OTs each of which is an indicator for a record type in the database schema. The essential ISD enables us to derive easily a 'skeleton DSD', which consists of the record types and the hierarchical links between them: every OT in the essential ISD defines a skeleton of a record type, and every relationship between OTs defines a directed link between an 'owner' and 'member' record type. The distinction between owner and member is made according to the types of dependency between the OTs in the essential ISD, as follows:
- in a 1:n sentence the dependent OT denotes the owner record type, and the determinant OT denotes its member record type;
- in an m:n sentence, where the relation between two OTs is defined as a 'nested' OT, each of the two 'original' OTs denotes an owner record type, and their nested OT denotes a member record type of the two owners.
(Note that there are no 1:1 sentences in the essential ISD).

To complete these explanations, Fig. 5 shows the essential ISD which is derived from the ISD of Fig. 3, and its respective skeleton DSD. It should be noted that his involves the same record types as in Fig. 4 (the complete DSD).

Based on the foregoing short description of the models and methods for conceptual schema and database schema design, we detail in the next section our ADDS system which automatically transforms the former into the latter.
3. ADDS: Functions and operations

The purpose of ADDS is to assist the database designer designing the database schema after the conceptual schema has been defined and expressed in the form of an ISD. ADDS accepts a description of the ISD and produces descriptions of the database schema. Since the system is to be used as a working tool by designers, we decided to develop it on a micro-computer under MS-DOS and to program it with the dBASE III (Ashton-Tate) application generator and DMBS, so as to enable the application of the system on various micro-computers.

Other work on this kind of problem, i.e. on the translation of a semantic data model into a database schema, has been mainly related to the general ER model, beginning with Chen [1] who showed that his ER diagram can be transformed into relational model relations and into
a data structure diagram of the network model. Chen's original transformation into the relational schema did not produce 3NF relations [13]. Later work on this topic resulted with various algorithms for a proper transformation; see for example [3, 6, 9].

An earlier work on the same problem, converting an ISD that is created according to the IA method into a normalized database schema, resulted in the CDC product IAST [10, 12]. In IAST the underlying BR model is somewhat different from the BR model we use. In the first case there is a separation in the ISD between the OT (termed NOLOT: Non Lexical OT) and its OTN (termed LOT) and therefore there are two types of relationships: a relationship between two OTs, termed 'idea', and a relationship between an OT and each of its OTNs, termed 'bridge'. In our case there is just one type of relationship (equivalent to 'idea') and the OTNs are marked within the circles of their OTs. Another difference between the ISDs of IAST and ADDS is that in the first case there is no 'nesting' of relationships, i.e. a relationship cannot be defined an OT. If a relationship has to be defined as an object type it is represented as a new OT, which is separately related to either of its 'original' OTs. To indicate this fact a 'uniqueness constraint' is defined between those relationships. In our view, the ISD on which ADDS is based is far more simple and easy to 'read' and understand. Obviously, a different transformation mechanism is needed for translating our ISD into a database schema.

One of the problems with which the IAST transformation method is concerned is of checking the 'referencibility' of an OT and the search for its 'best' unique identifier. This problem is crucial there because in their ISD there are 'compound' OTs which are actually relationships between other OTs, and which may not have OTNs of their own. Therefore it is possible to have an OT with no unique identification. In our case this problem is minimized, because every OT is either 'atomic', in which case it must be entered with an identifying OTN, or it is a 'nested' OT—originally a relationship—for which it is guaranteed to find a unique identifier (a concatenation of OTNs identifying the 'original' OTs). It has to be noted that in the case of IAST some additional features are included in the data model,
such as the classification of OTs into types and subtypes, and the inclusion of more types of constraints (e.g. 'subset constraint', 'exclusion constraint'). These features enable one to define more integrity constraints or rules in the database schema (usually creating 'subset' links between the record types). These integrity rules are implemented in a specific DBMS which IAST creates, termed IMF (running on CDC Cyber mainframes). As said, ADDS runs on a micro-computer under dBASE III. Although it does not utilize those 'extra' types of constraints, it succeeds to generate the normalized record types and the hierarchical/set relationships between them—as described earlier.

The major functions of ADDS are shown in Fig. 6 which is the main menu. Complementing it is Fig. 7 which is the macro flowchart of the system. In general, ADDS works as follows:

Fig. 7. ADDS: Macro flowchart.
The user (i.e. the database designer) inputs the description of all the elements of the ISD into two main files: one file, 'OTNs', accepts the description of the OTs and their OTNs, and the other file, "Relationships", accepts the descriptions of the relationships, including the associated OTs and the types of dependencies. In addition to these two files there is a file "OT-Descriptions" which enables one to describe or define the meaning of the OTs, and a file "OT-Synonyms" which is used to define synonymous names for OTs so as to enable the use of different (synonymous) OT names by various users/designers who may participate in the design. Since ADDS enables the creation of not only a global database schema, but also sub-schemata for various user views, it is possible to indicate for every OT, OTN and relationship to which user view it belongs. Figure 8 shows the DSD of ADDS, detailing its four files and the fields in each of them.

After entering the ISD description, ADDS performs some checks on the entered data to make sure that the ISD is correct (as will be explained in Section 3.1). Some of these checks can be done interactively and others in batch mode. Once all errors are corrected the system, together with the designer, handle the 1:1 dependencies, converting them to 1:n dependencies (see Section 3.2). During this process the system presents to the designer the different record types and relationships that may be obtained if either of the OTs associated in the 1:1 dependency is defined as a determinant OT, and it also recommends the 'better' solution to him. At the end of this step ADDS is ready to generate the database schema (see Section 3.3). As already mentioned, it is possible to generate a sub-schema for every user view, considering only the OTs, OTNs and relationships that were defined for a specific view, or to generate a global-schema. The resulting schema (or schemata) is stored in files which can be reported in two formats (see Section 3.4): as a detailed schema or as a 'skeleton' schema. In addition to these major report types ADDS can also produce a data dictionary which details all the information on the ISD, and it can also answer queries concerning specific details of it.

In the rest of this section the major steps and functions of ADDS are described. The description is followed by examples which are based on the ISD of Fig. 3.

3.1. Accepting the ISD description (selections 1–3 in main menu)

The ISD description is entered into the four files as were shown in the DSD of Fig. 8. (This is done using the friendly dBASE-III editor.) The first file to be edited is "OT-Descriptions" and the second is "OT-Synonyms". In both cases ADDS checks the correct-
ness of the entered data on-line. For the "OT-Descriptions" file ADDS verifies that there are no null-values in the "OT" (key) field, and for the "OT-Synonyms" file ADDS verifies that every value in the "OT" field appears in the "OT-Description" file.

After editing these two files the used may enter the other two files, "OTNs" and "Relationships". These are checked in batch mode. Here are some of the verifications that ADDS performs:

(a) **Valid values in fields:**
- there are no null-values in the fields "OT", "OTN", "OT1", "OT2", "Relationship name" and "Type of dependency";
- the only accepted values in the field "Type of dependency" are 1:1, 1:N, N:1 and M:N (where 1:N indicates that "OT1" is the determinant and N:1 indicates it on "OT2");
- every value in the fields "OT", "OT1" and "OT2" appears in the "OT-Descriptions" file;
- every "relationship name" (in the "Relationships" file) which is also marked as a 'nested'-OT appears in the "OT-Descriptions" file.

(b) **Redundancies**: There are no redundancies in key fields (the key of the "OTNs" file is "OT" plus "OTN"; the key of "Relationships" is "Relationship name").

(c) **Correct definitions**: Every OT (which is not a 'nested' OT) has one and only one OTN which is marked as an identifier. An "OT" may not also be a 'nested' OT.

---

**Fig. 9.** Two files of the ISD description; (a) "OTNs"; (b) "Relationships".
3.2. Handling 1:1 Dependencies (selection 4 in main-menu)

Before ADDS can generate the database schema, the relationships with 1:1 dependencies (in the "Relationships" file) have to be converted into 1:N or N:1 dependencies (i.e. one of the associated OTs has to be defined a determinant of the other OT). Theoretically, either OT can be selected as a determinant, but the selection may have different consequences in terms of different record types and relationships that may be generated in the schema, depending on additional relationships and dependencies that exist between the associated OTs and other OTs in the ISD. For example, if OTs A and B are 1:1 related, and if A is additionally a determinant of a third OT C, whereas the second OT, B, is not, then it would be preferred that the OT A be selected as the determinant, so that only one relation will be created with the first OT A as key field and the other two OTs B and C as (non-key) fields. It would otherwise have required the creation of two relations: one involving A (as key field) and C, and the other involving B (as key field) and A. Additionally, there would have been a link between the two relations, as based on the common field A.

ADDS assists the designer in making the ‘better’ selection by going through the 1:1 dependencies in the “Relationships” file and examining the relationships of each of the involved OTs with other OTs. It evaluates and determines the different record types and hierarchical links that may be created if either of the OTs is defined the determinant, and it recommends a preferred option. To do so, ADDS evaluates the two options by applying the conversion algorithms on the involved relationships, thus creating two alternative sets of record types and links. The two sets are presented to the user/designer along with a recommendation on the preferred one. ADDS enables the designer to overrule its recommendation or alternatively to adopt its recommendation by default. ADDS generally recommends the option which creates less record types and more links between them. This is under the assumption that the less record types in the schema the simpler it is, and the more links among the record types the easier it is to join them (in the case of the relational model) or to navigate among them (in the case of the network model). As will be shown later, these two criteria are sometimes conflicting.

ADDS’ evaluation and recommendation is based on the analysis of various possible combinations of relationships and dependencies of either of the two 1:1 related OTs with other OTs in a given ISD. Many different such combinations are possible; some of them are discussed here. For two OTs A and B (which are in a 1:1 relationship) the following additional relationships are considered:

(a) none of them, or one of them only, or both of them, are also determinants of other OTs, or have more than OTN;

(b) none of them, or one of them only, or both, are m:n related with other OTs.

These possibilities constitute ten different combinations of relationships, as shown in Table 1 (theoretically there are 16 combinations, but 6 of them are symmetric when OTs A and B are interchanged).

Table 1 can be viewed as a decision-table. For each of the columns (combinations) there is a recommendation concerning which OT, A or B, should be selected as the determinant OT. When ADDS discovers a 1:1 relationship in the ISD, it looks for the other relationships that the involved OTs have, and according to that it identifies a column (combination) in the table matching that pattern. Then it presents the two ‘solutions’ (on screen), and it recommends according to the last row in the table. (Note that in some of the columns, where there is a symmetric situation of the two OTs, the system is indifferent.) Space limitations
Table 1. Combination of relationships.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>1 A is determinant</td>
<td>+ + + + + + +</td>
</tr>
<tr>
<td>2 B is determinant</td>
<td>+ + + + + + +</td>
</tr>
<tr>
<td>3 A is (m:n) related</td>
<td>+ + + + + + +</td>
</tr>
<tr>
<td>4 B is (m:n) related</td>
<td>+ + + + + + +</td>
</tr>
<tr>
<td>ADDS recommends</td>
<td>- A - A A B A - B -</td>
</tr>
<tr>
<td>as determinant:</td>
<td></td>
</tr>
</tbody>
</table>

make it impossible to show and discuss in detail the ‘solutions’ that are created for all the combinations shown in the table (a detailed discussion on this topic will be reported separately); here we discuss only a few of them. The combination expressed in column 2 of Table 1 was discussed earlier in this section.

Combination 3 in the table represents the case where both OTs A and B are determinants of other OTs; hence for each of them there will be a record type in the schema. Therefore, if A is selected as a determinant, B will be a (non-key) field in its record type (in addition to having its ‘own’ record type). If B is selected, the opposite will happen. Obviously this is a symmetric situation and ADDS is indifferent to the two solutions (the designer may be indifferent as well).

In combination 5 OT A is both a determinant and in an \(m:n\) relationship, whereas B has no other relationships. Figure 10 details the situation: it shows the partial ISD and then the two alternative solutions which are presented to the user. ADDS recommends solution A.

In combination 9 OT A is both a determinant and in an \(m:n\) relationship, whereas B is only \(m:n\) related. Figure 11 shows the partial ISD and the two alternative solutions, respectively. In this case ADDS recommends B, even though this solution involved 4 record types (as opposed to 3 only if A is selected); this is because in the B solution the record types

![partial ISD](image)

![partial schema: solution A](image)

![partial schema: solution B](image)

Fig. 10. Combination 5 of Table 1.′
are 'better' linked than in the A solution. (Recall that the designer can overrule the recommendation.)

These were just a few examples from the decision-table (Table 1). Note that the table can be extended to include more columns, i.e. more combinations of relationships and dependencies between the OTs A and B and other OTs. (As mentioned previously, more on this subject will be reported separately.)

At the end of this stage all 1:1 dependencies are converted into 1:N or N:1 dependencies. Actually they are marked 1:I and I:1, respectively, in order not to destroy the original/real type of dependency.

3.3. Database schema design (selection 5 in main-menu)

ADDS enables the creation of a global database schema or sub-schemata for user views. In the first case, all the OTs, OTNs and relationships in the ISD are considered, whereas in the second case only those that belong to the specified user view (as marked in the DSD description) participate in the design process.

A schema design process is done in batch mode, and the resulting schema is stored in a file. Following is a concise structured description of the design algorithm implemented in ADDS, as based on the description in Section 2 (comments and explanations are given in parentheses):

**Begin**

1. **Definition of the record types according to the relationships in the ISD**

Open "Schema" file; (this file will store the record types created during the design process)

While not end of "Relationships" file:

If dependency type is N:1 or 1:N or I:1 or 1:I, then:

(I is treated as N. Its origin is from the transformation of 1:1 dependency)

If there is not yet a record-type for the determinant-OT (in the "Schema" file), then:
Define a new record-type in the "Schema" file;
Set Record-type name = determinant-OT;
Set Key-field name = determinant-OT;
Set Key-domain = determinant-OT;
  (the system assigns the name of the determinant-OT to the mentioned constructs.
   These names will be changed later on)
Endif;
Add the dependend-OT as a field of the determinant-OT's record type;
Set Field-name = relationship name
  (the system assigns the name of the relationship to the non-key field)
Set Field-domain = dependent-OT;
Endif;
If dependency type is N:M ('nested' OT) then:
  If there is not yet a record-type for the 'nested' OT (in the "Schema" file), then:
    Define a new record-type in the "Schema" file;
    Set the Record-type name = 'nested'-OT
      (the name of the record type which results from an m:n dependency is the name of
       the relationship)
  Else:
    Define (existing) Key-field = Key-field-1;
    Add Key-field-2;
      (for a record type which results from an m:n dependency the system defines two
       key fields, which are OT1 and OT2 of the involved relationship)
  Endif;
Set Key-field-1 name = OT1;
Set Key-field-2 name = OT2;
Set Key-field-1 domain = OT1;
Set Key-field-2 domain = OT2;
  (the system assigns the names of the two OTs to the two key fields, respectively)
Endif;
End-While;

(2. Definition of record types for OTs with more than one OTN)
While not end of "OTNs" file:
  (this file keeps the various OTNs of the OTs)
  If the OT has more than one OTN then:
    If there is not yet a record-type for that OT (i.e. if that OT is not a determinant) then:
      Define a new record-type (in the "Schema" file);
      Set Record-type name = OT;
      Set Key-field name = OT;
      Set Key-field domain = OT;
        (the name of the OT is assigned to the mentioned constructs)
    Endif;
  Endif;
End-While;

(3. Definition of SET relationships between record-types: the following procedure is sufficient to determine the hierarchical 'owner'/member' links between the record types)
While not end of "Schema" file:
Set TYPE = record-type name;
For all cases where: field-domain = TYPE and record-type name $\neq$ TYPE:
  (if the name of a field’s domain is the same as the name of a given record type, but
  which is not the name of the record type to which that field belongs, the system
determines that the given record type is the ‘owner’ of the record type to which the
above field belongs)
Set Owner = TYPE
  (marks the field through which the ‘member’ is linked to the ‘owner’ record type)
End-While;

(4. Decomposition of “nested” OT fields: this procedure takes any field in the schema which
was originally a ‘nested’ OT and it decomposes it into its ‘originating’ OTs)
Set replacement-sign = ‘T’;
While replacement-sign = ‘T’:
  Set replacement-sign = ‘F’
  While not end of “Schema” file:
    If Field-name = (any) relationship-name in the “Relationships” file, then:
      Set Field-name = OT1;
      Set Field-domain = OT1;
      Add a new field;
      Set (new) Field-name = OT2;
      Set (new) Field-domain = OT2;
      (if the original field was defined a key field or if it defined a SET relationship, those
definitions are inherited to the newly defined fields)
      Set replacement-sign = ‘T’;
    Endif;
  End-While;
End-While;

(5. Domain replacement of identifier OTNs: the OTN which is an identifier becomes the
domain of every field in the schema)
While not end of “Schema” file:
  Replace: Field-domain = identifier-OTN (from the “OTNs” file);
End-While;

(6. Adding fields for non-identifier OTNs: all non-identifying OTNs of an OT are added as
new fields in the record type where that OT is the key field)
While not end of “OTNs” file:
  If the OTN is not an identifier, then:
    Add a new field to the OT’s record-type (in the “Schema” file);
    Set Field-name = OT;
    Set Field-domain = OTN;
  Endif;
End-While;

Note. ADDS assigns names to record-types, keys and fields, as based on names of OTs,
OTNs and relationships in the ISD. At the end of the design process the user can change
names, if he wishes, by editing the schema file.
3.4. Reports, queries and data-dictionary (selection 6-7 in main menu)

The database schemata are stored on files. It is possible to produce two types of report for every database schema created by ADDS: a complete-schema and a skeleton-schema.

(a) *The complete-schema report.* This report details the record types and for each of them it details its name, the field names and field domains, and indicates the key field(s). It also indicates foreign-key fields through which record types are defined as “members” of other record types. The name of the “owner” record type is shown next to the foreign-key. Figure 12 shows the complete database schema report for our example. Note the similarity with the complete DSD in Fig. 4.

```
orders00.dbf

*** Automatic Database Design system ***
Design Summary

<table>
<thead>
<tr>
<th>Field name</th>
<th>Domain</th>
<th>Key</th>
<th>Owned by record</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Record name: PROJECT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manager</td>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: REQUEST</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request</td>
<td>No.</td>
<td>Primary</td>
<td></td>
</tr>
<tr>
<td>Date of Request</td>
<td>mn.dd.yy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initiator</td>
<td>code</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: ORDER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order</td>
<td>No.</td>
<td>Primary</td>
<td></td>
</tr>
<tr>
<td>Order to Supplier</td>
<td>code</td>
<td></td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>Date of Order</td>
<td>mn.dd.yy</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: PART IN ORDER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order</td>
<td>No.</td>
<td>Primary</td>
<td>ORDER</td>
</tr>
<tr>
<td>Request</td>
<td>No.</td>
<td>Primary</td>
<td>PART IN REQUEST</td>
</tr>
<tr>
<td>Part</td>
<td>No.</td>
<td>Primary</td>
<td>PART IN REQUEST</td>
</tr>
<tr>
<td>Quantity in Order</td>
<td>No of unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: PART IN REQUEST</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request</td>
<td>No.</td>
<td>Primary</td>
<td>REQUEST</td>
</tr>
<tr>
<td>Part</td>
<td>No.</td>
<td>Primary</td>
<td>PART</td>
</tr>
<tr>
<td>Quantity in Request</td>
<td>No of unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: SUPPLIER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplier</td>
<td>code</td>
<td>Primary</td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>Supplier</td>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address of Supplier</td>
<td>No.st.city</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: BRAND</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplier</td>
<td>code</td>
<td>Primary</td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>Part</td>
<td>No.</td>
<td>Primary</td>
<td>PART</td>
</tr>
<tr>
<td>Unit Price</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Record name: PART</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Part</td>
<td>No.</td>
<td>Primary</td>
<td></td>
</tr>
<tr>
<td>Part</td>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Part Unit</td>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Part Description</td>
<td>text</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Fig. 12. Complete-database schema.
(b) The skeleton-schema report. This report details only the record type names, their key field(s) and the 'owner(s)' and 'member(s)' of each record type. Figure 13 shows the skeleton-database schema report for the example. Note the similarity with the skeleton-DSD in Fig. 5(b).

In addition to the schema reports, ADDS creates a data dictionary. It contains all the details that are stored in the system concerning the ISD. For example, for the OTs it includes their descriptions, synonyms, OTNs and the relationships in which they are involved.

In addition to the data dictionary report, ADDS can answer on-line queries. When the user indicates an OT or relationship name, it retrieves the appropriate 'page' in its data-dictionary. On the first screen it shows the various details of the OT, and on the following screens it shows (upon request) the relationships in which that OT is involved. Figure 14 shows an example.
4. Summary and further developments

ADDS accepts a description of a conceptual schema in the form of an ISD and it converts it into a database schema which consists of normalized record types and hierarchical links between them. The resulting schema can be used to create a DBMS implementation. If the system is implemented using a relational based DBMS, then the normalized relations defined by ADDS can be utilized with no changes. Obviously, the links between ‘owner’ and ‘member’ relations are not utilized here, but these are still of help since they indicate to the users of the database which relations it is possible to join. If the system is implemented using a CODASYL/Network DBMS, then it is possible to utilize both the record types and the set links between ‘owners’ and ‘members’. In this case the database administrator may decide to eliminate (some of) the foreign keys, and, of course, he has to add definitions concerning location modes and set membership characteristics, according to the specific DDL. These, as other aspects of implementation, are beyond the current scope of ADDS.
Further developments of ADDS are sought. One of them is adding a graphical dimension. It is desirable that the system have the capability to accept the conceptual schema description in its diagrammatic form (instead of textual) and to produce the database schema in the form of a DSD (instead of reports). A system with these capabilities is already under development [7].

Another possible enhancement of ADDS is to add an explanation capability so that it will be able to explain to the user (i.e. the designer) what it is doing during the design process and how it came to its results. For example: what is the reason for creating a certain record type?, why is a certain field defined as a key?, why is a record type an 'owner' or 'member' of another record type? etc. With this ability ADDS will not only design, but it will also 'teach' how to design. A part of this ability already exists in the system in the module which advises how to convert 1:1 dependencies (see Section 3.2 above).

ADDS may be extended to do parts of the database creation stage, i.e. to create a database schema definition. This extension necessitates the ability to create specific DMBS schemata for specific DMBS models with their DDLs. For the case of the relational DMBS this task may seem to be straightforward, but for the case of a CODASYL/Network DBMS it is not so because of the many additional definitions required in the schema DDL (as indicated earlier).

Finally, a considerable step in database design may be the development of an expert system that will do the stage of information structure analysis. The current system begins its task with a given conceptual schema which is the result of an analysis done by a human expert. We can think of an expert system that will do at least part of this job. For example, it will accept from the user (analyst) a description of the information structure in the form of 'simplified' natural language sentences (we do not define here what 'simplified' means), and it will analyze them and extract the elementary/binary sentences. Obviously, such a system will have to interact with the user and prompt him to enter semantic information regarding, for example, the types of dependencies between objects in sentences, and it will suggest to the user possible ISDs for the sentences entered by him. However, a more detailed discussion of this issue deserves a separate report.

References