A NEW METHOD FOR DESIGNING A TOOL DATABASE
SYSTEM OF AUTOMATED MANUFACTURING SYSTEMS

Hong-Feng Lai

ABSTRACT

Since software development is a process of model transformations, to view various models in an integrated manner is a necessity. The purpose of this paper is to propose a new design method incorporating the required organization model (ROM) and IDEF1X data model. For function analysis, requirement specifications are first clustered into object domains using ROM. For data analysis, the IDEF1X is then employed to describe the static structure, which is transformed into a relation schema. Finally, the database system is implemented using an object-oriented (OO) database program. The correspondence and transformation rules of individual models are highlighted. The procedure applying ROM/IDEF1X methodology is demonstrated by a tool database system.

Key Words: object domain, relation schema, data model.

I. INTRODUCTION

Today's production environment is moving toward customization and wide variety, which has caused a growth in FMS or CIM tooling (Torvinen et al., 1991). The number of tools in a medium size manufacturing system can easily exceed one thousand. However, a system with a large number of tools is harder to handle than a small system (Kusiak and Finke, 1988). One solution to these problems is through a tool database system for alleviating the burden of tooling, selecting appropriate tools, and reducing the time for preparing tools.

However, how to design a tool database system is still a challenging issue (Shayan and Liu, 1995; Backes et al., 1998). Most of the work in developing a tool database system presents the simulation processes and implementation results (Torvinen et al., 1991; Chan, 1992). From an optimization viewpoint, Akturk and Onen (2002) point out that tool management and lot sizing cannot be viewed in isolation. They propose algorithms to determine lot sizing, tool allocation and machining conditions simultaneously.

Few papers have focused on the design method except that Shayan and Liu (1995) exploited structured analysis and design technique (using IDEF0) to implement tool database systems in ORACLE. Backes et al. (1998) employed an artificial neural network to provide the planning capability for bending tools.

OO technique (Wirfs-Brock et al., 1990; Rumbaugh et al., 1991; Booch, 1994) has been admittedly useful in the analysis and design of automated manufacturing systems. Tönshoff and Dittmer (1990) pointed out the benefits of the OO data model in the tool database design area. However, how to apply OO techniques in the design of a tool database has not been well worked out.

The transformation from requirement specifications into OO programs is regarded as the major challenge in OO applications. Wirfs-Brock et al. (1990) and Rumbaugh et al. (1991) identified the candidate classes and objects from the text of the problem domain, while Booch (1994) began the analysis tasks in the scenario. The other trend is a mixture approach. Hubbers and Hofstede (1997) combined the use case, interface description, and problem domain description to construct the requirement model. These methods offered a valuable support for analysis and guidance of an OO system. However, they did not offer unambiguous development procedures for the requirement analysis since they were heavily dependent on the textual analysis. Also, how to integrate the
requirement model and static structural model was not considered.

In computer integrated manufacturing systems, a tooling system can be applied to the following tasks including (1) select tools, (2) tool inventory management, (3) adjust tools, (4) monitor machining tool status, (5) transport tools, and (6) integrate with other systems (Bard and Feo, 1989). In our approach we aim at (1) select tools, (2) tool inventory management, and (6) integrate with other systems. That involves functional and static requirement specifications. Therefore, we consider the function and data model, modeled by ROM and IDEF1X respectively. If a tooling system is equipped with the capabilities of (3) adjust tools, (4) monitor machining tool status, and (5) transport tools, behavioral and temporal relationships should be addressed. However, that is outside the scope of this paper.

This paper addresses the problem of incorporating the requirement organization model (ROM) and IDEF1X in the design of a tool database system. We employ a tool database system as an example to demonstrate the proposed ROM/IDEF1X method.

This paper is organized as follows. Section II presents the ROM/IDEF1X method. Section III describes how to apply ROM/IDEF1X to analyze a tool database system. In Section IV, we continue the ROM/IDEF1X steps. Section V presents the implementation result. The final section draws conclusions and makes suggestions.

II. ROM/IDEF1X

In this section, the constituents of the ROM/IDEF1X method are introduced.

1. ROM

Requirement engineering is one of the crucial steps in the creation of an information system. There are a set of tools and notations to describe the intended behavior, and to ensure the consistency and feasibility of a proposed system (Hsia et al., 1993). Traditional requirement analysis is done using structured analysis, such as text or case analysis. However, these approaches depend heavily upon the preference of software developers.

ROM (Chen et al., 1992) is a systematic method based on analyzing the requirement specifications. The ROM contains two models: a requirement collection model (RCM) and a requirement reorganization model (RRM).

RCM is applied to collect the requirement statement database (RDB) and software requirement specification operations. For instance, a proposed system may contain the following requirements and operations such as adding, deleting, and editing.

\[ RDB = \{ Rq_i \} \]

\[ ADD (Rq_i) = RDB \cup Rq_i; \]

\[ DEL (Rq_i) = RDB - Rq_i; \]

\[ EDT (Rq_i) = \text{edit the } i\text{th requirement statement} \]

\[ RDB, \text{ where a requirement } Rq_i \text{ is usually a statement.} \]

RRM is used to represent the following relationships such as a target system and its object domain, an abstract data model with a requirement statement, and basic requirements. These relationships can be described as follows.

Object domains constitute a system. This relationship can be denoted by the following statement:

\[ O = \{ O_i \}, O_i = \text{object domain}, i \in N \} \]

The abstract data model and the requirement statement can be clustered into an object domain. This relationship is expressed as:

\[ O_i = \{(ADM_k, Rq_j) | ADM_k = \text{abstract data model}, Rq_j = \text{requirement statement}; k, j \in N \} \]

The basic requirements are components of a requirement statement. This relationship is written as:

\[ Rq_j = \{ G_i, g_i | G_i, k = \text{basic requirements}, j, k \in N \} \]

The ROM procedure is described as follows. A requirement statement is first decomposed into a set of basic requirements. An abstract data model is then created according to their attributes and variables. The object domain is constructed according to its abstract data model and requirement statement. The object domain is finally mapped into the programming structure.

2. IDEF1X Method

The Entity Relationship Diagram (ERD) was proposed by Chen (1976) for describing the information structure of entities and relationships. Several types of extended ERD have been proposed to reach some specific objectives. Teorey et al. (1986) and IDEF1X (FIPS 184, 1993) are some of them, where IDEF1X has the following useful characteristics: strong expressive capability and robust semantics (Zhang et al., 1996).

There are three types of relationships, identifying, non-identifying, and categorization, defined in IDEF1X. In Fig. 1(a), a solid line with a dot on the child end indicates an identifying relationship between the parent and child entities, where ‘P’ represents a one to (one or many) relationship. A dashed line with a dot on the child end indicates a non-iden-
identifying relationship between the parent and child entities. In Fig. 1(b), a categorization relationship is indicated as a line extending from the generic entity to an underlined circle and lines extending from the circle to the various types of category entities. In Fig. 1(c), a solid line with dots on the two entity ends denotes a non-specific relationship, which is applied at the beginning of the development stage and can be transformed further into a specific relationship.

The construction of an IDEF1X model may contain the following five phases: Phase 0 defines the goal of a system and identifies its limitations. Phase 1 identifies the entities that will be applied in the target system. Phase 2 specifies the relationships between the entities. Phase 3 refines non-specific relationships into specific ones and defines the key attributes, where primary keys are moved to form foreign keys in child entities. Phase 4 defines the attribute pool, attribute ownership, non-key attribute, and data structure (Kusiak et al., 1997).

3. Development Procedure of ROM/IDEF1X

The construction of an OO database system using ROM/IDEF1X is depicted in Fig. 2. The requirement organization model is constructed by ROM. The abstract data model is established by IDEF1X. The object domain is obtained through the integration of ROM and IDEF1X. It is then mapped into a programming structure.

The ROM/IDEF1X development procedure is divided into the following seven steps:
Step 1: Establish the plan for the collection of requirement specifications according to the user requirements. The outputs of this step are a set of requirement statements and a set of basic requirements.
Step 2: Find the related physical objects that are involved in the requirement statements.
Step 3: Identify the related entities using an entity relationship matrix. Depict the non-specific and specific relationships using IDEF1X.
Step 4: Identify the attributes and primary keys. Determine the minimal functional dependency using the minimal cover theorem.

Step 5: Depict the abstract data model according to the results of Step 2, Step 3, and Step 4. Transform the IDEF1X data model into relation schema. The transformation rules will be introduced in Section II.4.
Step 6: Cluster the basic requirements into their related objects. The output of this step is a set of object domains, where an object domain consists of a set of abstract data models and a set of basic requirements.
Step 7: Map each object domain into OO programming modules. The transformation rules will be described in Section II.5.

4. Transformation between IDEF1X Data Model and Relation Schema

In this section, the IDEF1X data model will be transformed into relation schema, a set of mathematical notations for describing data and relationships. Unlike Zhang et al. (1996), our work is an extension of Kusiak et al. (1997), and can formally model the requirement specifications since the transformation rules are based on mathematical notations. These transformation rules are inspired by the work of Markowitz and Makowsky (1990). They explored the correspondence between EER and relation schema. The basic concepts of a relational data model are reviewed below. Details can be found in the textbook by Ullman (1982).

A relation schema \((R, F, I)\) consists of a set \(R\) of relational schemes \(R_i\), a set of functional dependencies \(F\), and a set of inclusion dependencies \(I\).
A relational scheme \( R(X_i) \) contains a relation name \( R_i \), \( X_i \) refers to the associated set of attributes \((A_1, A_2, \ldots, A_n)\). A relation value \( r(R_i) \) is an assignment set of \( R_i \). Each attribute is assigned a domain. A relation value is a subset of the Cartesian product \( \Pi \in (\text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_n)) \).

A functional dependency \( R_i: X \rightarrow Y \) defines the key \( X \) in a relation. Let \( F \) be a set of functional dependencies. It can be stated that \( X \) logically implies \( Y \), denoted as \( f(X) = Y \).

An inclusion dependency \((ID)\) is a statement \( R_i(A) \subseteq R_i(B) \in I \) to express the referential integrity constraints, where \( R_i(X_i), R_j(X_j) \) are two relational schemes; \( A, B \) are two subsets of \( X_i, X_j \) respectively; and \( I \) is a set of inclusion dependencies. \( R_i(A) \subseteq R_i(B) \in I \) is satisfied by relations \( r_i, r_j \), if and only if \( \Pi_{A}(r_i) \subseteq \Pi_{B}(r_j) \), where \( \Pi_{A}(r_i) \) denotes the projection of \( r_i \) onto the components \( A \).

The transformation rules of IDEF1X/(R, F \& I) are expressed as follows.

Rule-(1): Map the value-set in IDEF1X into the attribute domain.
Rule-(2): Map the independent entity \( E_i \) in IDEF1X into the relational scheme \( R_i(X_i) \). If \( Z_i \) is the primary key of \( E_i \), then \( R_i: Z_i \rightarrow X_i \) is a set of functional dependencies in \( F \).
Rule-(3): For a composite child entity (relationships with many parent entities) \( E_i \), if it has parent entities \( E_{i_1}, 1 \leq j \leq m \), then map child entity \( E_i \) in IDEF1X into the relational scheme \( R_i(X_i) \), where \( X_i = X_i' \cup X_i'' \), \( X_i' \) is defined as \( R_i: Z_i \rightarrow X_i' \). \( Z_i \) is the primary key of \( E_i \), \( X_i'' = \bigcup_{j=1}^{m} FK_{ij} \), if all parent entities and child entities are identifying relationships as indicated in Fig. 3, where \( FK_{ij} \) is a correspondence of primary key \( K_{ij} \). \((Z_i \cup X_i'') \rightarrow X_i \) is a set of functional dependencies in \( F \). \( R_i[FK_{ij}] \subseteq R_i[K_{ij}] \) is a set of inclusion dependencies in \( I \).

If some parent entities \( E_{i_k} \) and a child entity \( E_i \) form non-identifying relationships, as indicated in Fig. 1(a), \( (X_i' \rightarrow FK_{ik}) \cup Z_i \) is the key of relational scheme \( R_i \), where \( FK_{ik} \) is a correspondence of primary key \( K_{ik} \). \( R_i[(X_i' \rightarrow FK_{ik}) \cup Z_i] \rightarrow X_i \) is a set of functional dependencies in \( F \).

Rule-(4): If \( E_i \) is a category entity, \( E_i \) is a generic entity, as indicated in Fig. 1(b), the key of \( E_i \) inherits from the key of \( E_j \). The corresponding relational scheme of category entity \( E_i \) is: \( R_i(X_i), X_i = X_i' \cup X_i'' \), where \( X_i'' \) is the non-key of \( E_i \) and \( X_i' \) is the key inherited from \( E_j \). \( R_i: X_i' \rightarrow X_i \) is a functional dependence of \( F \). \( R_i[FK_{ij}] \subseteq R_i[K_{ij}] \) is an inclusion dependence of \( I \).

5. Mapping Object Domain to OO Program

The final step of ROM/IDEF1X is the implementation of a target system in the OO program. There are still some gaps between the relational data model and the OO program (Jorgensen, 1998). In this section, the transformation rules of this study will be explored.

Rule-(5): Map the object domain into the database system.
Rule-(6): Map the relation schema \( R \) into the total database files.
Rule-(7): Map every relational scheme \( R \) into a database file (a table).
Rule-(8): Map every attribute name into a field name of a table.
Rule-(9): Map every functional dependence into the key and non-key relationships.
Rule-(10): Map every inclusion dependence into the referential integrity constraint.
Rule-(11): Map every requirement statement into a procedure in the OO program.

The correspondences between ROM/IDEF1X and an OO program are summarized in Table 1.

From the viewpoint of transformation, the design tasks of a proposed system are a series of model mappings. Thus, the ROM/IDEF1X transformation rules support the necessary information for model translation and system implementation.

III. APPLY ROM/IDEF1X TO DEVELOP A TOOL DATABASE SYSTEM

The ROM/IDEF1X development procedure for a tool database system is presented in the following two sections.

1. Decompose Requirement Statements into Basic Requirements

The functional requirements of a tool database system are extracted from the interactions of parts, orders, and tools. In this example, it is assumed that the functional requirement includes (1) record the
relevant tool information, (2) prepare necessary tools per order, (3) prepare necessary tools per part, (4) invoke a CAD system, (5) invoke a CAM system, and (6) enter and exit the tool database system.

After decomposing the requirement statements into basic requirements, a $RCM = \{Rq_1, Rq_2, Rq_3, Rq_4, Rq_5, Rq_6, Rq_7, Rq_8, Rq_9, Rq_{10}, Rq_{11}\}$ can be obtained.

- $Rq_1 = \{G_{1.1}, G_{1.2}, G_{1.3}\}$: $G_{1.1}=$add a part into the system. $G_{1.2}=$delete a part from the system. $G_{1.3}=$edit a part in the system.
- $Rq_2 = \{G_{2.1}, G_{2.2}, G_{2.3}\}$: $G_{2.1}=$add a tool into the system. $G_{2.2}=$delete a tool from the system. $G_{2.3}=$edit the tool data in the system.
- $Rq_3 = \{G_{3.1}, G_{3.2}, G_{3.3}\}$: $G_{3.1}=$add an order into the system. $G_{3.2}=$delete an order from the system. $G_{3.3}=$edit the order data in the system.
- $Rq_4 = \{G_{4.1}\}$: $G_{4.1}=$print the tool list according to a specific part.
- $Rq_5 = \{G_{5.1}\}$: $G_{5.1}=$print the tool list according to a typical order.
- $Rq_6 = \{G_{6.1}\}$: $G_{6.1}=$find tools according to the querying conditions.
- $Rq_7 = \{G_{7.1}, G_{7.2}\}$: $G_{7.1}=$invoke the CAD system. $G_{7.2}=$exit the CAD system.
- $Rq_8 = \{G_{8.1}, G_{8.2}\}$: $G_{8.1}=$invoke the CAM system. $G_{8.2}=$exit the CAM system.
- $Rq_9 = \{G_{9.1}, G_{9.2}\}$: $G_{9.1}=$enter the order information system. $G_{9.2}=$exit the order information system.
- $Rq_{10} = \{G_{10.1}, G_{10.2}\}$: $G_{10.1}=$enter the part information system. $G_{10.2}=$exit the part information system.
- $Rq_{11} = \{G_{11.1}, G_{11.2}\}$: $G_{11.1}=$enter the tool information system. $G_{11.2}=$exit the tool information system.

2. Define the Physical Objects

The information of interest in the tool database system domain can be classified into three types (Chakravarty et al., 1997). The first type relates to the information of a part or product. The second type concerns physical facilities, including machines, cells, robots, and tool information. The third type relates to information of operations and sequences, e.g. order information.

<table>
<thead>
<tr>
<th>Corresponding items:</th>
<th>ROM/IDEF1X</th>
<th>OO database program</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall correspondence</td>
<td>object domain</td>
<td>database system</td>
</tr>
<tr>
<td>data model</td>
<td>relation schema</td>
<td>set of database files</td>
</tr>
<tr>
<td>object</td>
<td>relational scheme</td>
<td>database file</td>
</tr>
<tr>
<td>attribute</td>
<td>attribute name</td>
<td>field name of a table</td>
</tr>
<tr>
<td>relation</td>
<td>functional dependency</td>
<td>key of a table</td>
</tr>
<tr>
<td>relation</td>
<td>inclusion dependency</td>
<td>referential integrity constraint</td>
</tr>
<tr>
<td>function</td>
<td>requirement statement</td>
<td>procedure</td>
</tr>
</tbody>
</table>

Table 1 The mapping between ROM/IDEF1X and an OO database program

3. Identify the Relationships and Selecting Rules

An order may contain many parts to be produced, while a part may be ordered by many orders. Thus, the relationship between an order and a part is non-specific. Similarly, a part and a tool also have a non-specific relationship. A non-specific relationship can be resolved into two specific relationships and a new associative entity (Kusiak et al., 1997) as shown in Fig. 4.

In Fig. 4, the relationship between a part and a tool is determined by a set of factors, e.g. machining features, diameter of a hole, length of a hole, tolerance on part diameter, and its operation properties. For example, center cuts are fit for a blind hole, while twist drills are employed for a through hole. A boring tool is needed if the diameter of the inner hole is...
greater than the diameter of a drill. A reamer is chosen if a machining of accurate size is required.

Applying the above mentioned selection rules, for instance, we can implement a selection rule using database program language as the following form: “select toolnum, tooltype, toolfun from tooldb where tooldia <= holedia and toolfun in featureset”, which indicates that a tool with field names toolnum, tooltype, and toolfun is selected from a database file “tooldb” if the diameter of a tool is smaller than the diameter of a hole and the machining function of a tool is in the set of features.

IV. OBJECT DOMAIN OF TOOL DATABASE SYSTEM

In this section, we continue the development steps of the ROM/IDEF1X. These steps include identification of the attributes and their primary key, determination of the abstract data model and transformation of the IDEF1X data model into relation schema, grouping of the basic requirements into their related object domain, and mapping each object domain into the OO programming modules.

1. Identify the Attributes and Their Corresponding Primary Key

Data identification defines the attributes and their functional dependence involved in a tooling system. To effectively maintain the inventory of all tools, a tool is identified typically by its type, number, and location (Souza, 1997). In our work, a tool entity is defined by its tool diameter, tool material, tool location, tool offset, and tool function. Its relational scheme is:

Tool(tool-num, tool-dia, tool-mat, tool-loc, tool-offset, tool-fun).

An underline refers to a primary key. A tool-list entity represents the required tools according to an order. Part number, tool-number, and required quantity can describe this entity. Its relational scheme is:

Tool-list(part-num, tool-num, tool-amt).

A part entity can be expressed by its management information such as part number, part material, and part price. Based on the shape information, a part is described by its part length and part angle. Moreover, according to the manufacturing significance, a part can be depicted by its features such as slot, step, blind hole, through hole, chamfer, and thread.

Altogether, we use part number, part material, part angle, part length, part feature, and part price to represent a part. The relational scheme of a part is:


An order entity describes the task of scheduling. This entity is defined by its order number, customer number, start time, and stop time. Its relational scheme is:

Order(order-num, cust-num, start-time, stop-time).

An order-detail entity is an associative entity denoting the part list of a specific order. Its order number, part number, quantity, price, and amount describe this entity. Its relational scheme is:

Order-detail(order-num, part-num, qty, price, amt).

2. Depict the Abstract Data Model and Transform it into Relation Schema

The purpose of developing the IDEF1X data model is to facilitate the implementation of a relational database. After having defined the entities, attributes, and relationships, an IDEF1X data model of a tool database system can be depicted in Fig. 5, where a tool entity is discriminated by a set of category entities: BoringCat, DrillerCat, ReamerCat, and TapCat.

According to the transformation rules of
Table 2 Relation schema of IDEF1X data model (see Fig. 5)

<table>
<thead>
<tr>
<th>Object</th>
<th>Relation schema</th>
<th>Attribute definition</th>
<th>Inclusion dependency(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>$R_1(A_{11}, A_{12}, A_{13})$</td>
<td>$A_{11}$: order-num, $A_{12}$: cust-num, $A_{13}$: start-time, $A_{14}$: stop-time.</td>
<td>$R_2[A_{21}] \subseteq R_1[A_{11}]$, $R_2[A_{22}] \subseteq R_1[A_{31}]$</td>
</tr>
<tr>
<td>Order-detail</td>
<td>$R_2(A_{21}, A_{22}, A_{23}, A_{24})$</td>
<td>$A_{23}$: qty, $A_{24}$: price, $A_{25}$: amt.</td>
<td>$R_2[A_{21}] \subseteq R_1[A_{11}]$, $R_2[A_{22}] \subseteq R_1[A_{31}]$</td>
</tr>
<tr>
<td>Part</td>
<td>$R_3(A_{31}, A_{32}, A_{33}, A_{34}, A_{35}, A_{36}, A_{37})$</td>
<td>$A_{31}$: part-num, $A_{32}$: part-dia, $A_{33}$: part-mat, $A_{34}$: part-ang, $A_{35}$: part-len, $A_{36}$: part-fea, $A_{37}$: part-price.</td>
<td>$R_3[A_{31}] \subseteq R_2[A_{11}]$, $R_3[A_{32}] \subseteq R_2[A_{11}]$</td>
</tr>
<tr>
<td>Tool</td>
<td>$R_4(A_{41}, A_{42}, A_{43}, A_{44}, A_{45}, A_{46})$</td>
<td>$A_{41}$: tool-num, $A_{42}$: tool-dia, $A_{43}$: tool-mat, $A_{44}$: tool-ang, $A_{45}$: tool-offset, $A_{46}$: tool-fun.</td>
<td>$R_3[A_{31}] \subseteq R_2[A_{11}]$, $R_3[A_{32}] \subseteq R_2[A_{11}]$</td>
</tr>
<tr>
<td>Tool-list</td>
<td>$R_5(A_{51}, A_{52}, A_{53})$</td>
<td>$A_{53}$: tool-amt</td>
<td>$R_5[A_{51}] \subseteq R_3[A_{11}]$, $R_5[A_{52}] \subseteq R_3[A_{41}]$</td>
</tr>
<tr>
<td>Boringcat</td>
<td>$R_6(A_{61})$</td>
<td></td>
<td>$R_6[A_{61}] \subseteq R_4[A_{41}]$</td>
</tr>
<tr>
<td>Drillcat</td>
<td>$R_7(A_{71})$</td>
<td></td>
<td>$R_7[A_{71}] \subseteq R_4[A_{41}]$</td>
</tr>
<tr>
<td>Reamercat</td>
<td>$R_8(A_{81})$</td>
<td></td>
<td>$R_8[A_{81}] \subseteq R_4[A_{41}]$</td>
</tr>
<tr>
<td>Tapecat</td>
<td>$R_9(A_{91})$</td>
<td></td>
<td>$R_9[A_{91}] \subseteq R_4[A_{41}]$</td>
</tr>
</tbody>
</table>

Table 3 Correspondence between object domain and relation schema

<table>
<thead>
<tr>
<th>Object domain</th>
<th>Group of basic requirements</th>
<th>Relational scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>OD-1</td>
<td>$D_1$</td>
<td>$R_1(A_{11}, A_{12}, A_{13})$</td>
</tr>
<tr>
<td>OD-2</td>
<td>$D_2$</td>
<td>$R_3(A_{21}, A_{22}, A_{23}, A_{24})$</td>
</tr>
<tr>
<td>OD-3</td>
<td>$D_3$</td>
<td>$R_3(A_{31}, A_{32}, A_{33}, A_{34}, A_{35}, A_{36}, A_{37})$</td>
</tr>
<tr>
<td>OD-4</td>
<td>$D_4$</td>
<td>$R_4(A_{41}, A_{42}, A_{43}, A_{44}, A_{45}, A_{46})$, $R_6(A_{61})$, $R_7(A_{71})$, $R_8(A_{81})$, $R_9(A_{91})$, $R_5(A_{51}, A_{52}, A_{53})$</td>
</tr>
<tr>
<td>OD-5</td>
<td>$D_5$</td>
<td>Fe-CAD</td>
</tr>
<tr>
<td>OD-6</td>
<td>$D_6$</td>
<td>Fe-CAD</td>
</tr>
<tr>
<td>OD-7</td>
<td>$D_7$</td>
<td>Fe-CAD</td>
</tr>
</tbody>
</table>

IDEFIX and relation schema, the schema is indicated in Table 2. For instance, the object Order identified by the order number ($A_{11}$) represents a functional dependence, while this attribute ($A_{11}$), also utilized by the object Order-detail, defines an inclusion dependence $R_2[A_{21}] \subseteq R_1[A_{11}]$.

3. Cluster Basic Requirements into Object Domains

In this step, basic requirements are clustered into groups corresponding to their involved entities. These groups will be incorporated with relation schema for constructing an object domain. The groups of basic requirements $D_i$ are defined as follows. $D_1=\{G_{3,1}, G_{3,2}, G_{3,3}, G_{9,1}, G_{9,2}\}$; $D_2=\{G_{2,1}, G_{2,2}, G_{3,3}\}$; $D_3=\{G_{1,1}, G_{1,2}, G_{1,3}, G_{6,1}, G_{10,1}, G_{10,2}\}$; $D_4=\{G_{2,1}, G_{2,2}, G_{2,3}, G_{6,1}, G_{11,1}, G_{11,2}\}$; $D_5=\{G_{4,1}, G_{5,1}\}$; $D_6=\{G_{7,1}, G_{7,2}\}$; $D_7=\{G_{8,1}, G_{8,2}\}$.

The correspondence between the object domain and relation schema is indicated in Table 3. A functional entity is utilized as the mode of communication with other entities in the environment (Beaumariage et al., 1995). For instance, an interface entity should be constructed for invoking the CAD or CAM system. This type of entity does not correspond to a relation schema, but is named Fe-*. The integration of relation schema (IDEFIX) and groups of basic requirements (ROM) construct an object domain. The structure of an object domain is shown in Fig. 6, where $D_i$ denotes a functional requirement, a relation scheme denotes a data model, and a rectangle of dashed line refers to a functional entity.
4. Map Each Object Domain into OO Programming Modules

Relational databases are the most common commercially available database systems. Based on the mapping rules presented in Section II.5, the correspondence between the relational scheme and relational database program are explored in Table 4, where a relational scheme is mapped into a database file name, and the groups of basic requirements are transformed into procedure names. The set of procedures corresponds to the groups of basic requirements which include: {metasys, orderpas, partpas, partedit, selecttool, toolpas, tooledit, reportt}.

Having distinct definitions of modules, a procedure corresponding to group $D_i$ is easy to implement in an OO database program language, e.g. Delphi4, which consists of a set of linking or interface elements for designing a system.

A portion of the procedure ‘toolpas’ corresponding to $D_2$ is indicated in Fig. 7. These programs are written in Object PASCAL. Also, these programs meet the functional requirements of group $D_2$.

5. IMPLEMENTATION AND DISCUSSIONS

In this section, we will describe the results and express the characteristics of the ROM/IDEF1X method. The implementation of the overall system is guided by the structure of the object domain (see Fig. 6 and Fig. 7).
Fig. 8. Details are also presented in Lai (1998).

1. System Structure and Operating Process

The overall system consists of modules of an order information system, a part information system, and a tool information system. The order information system (OIS) consists of object domains OD-1 and OD-2. The part information system (PIS) includes object domains OD-3, OD-6, and OD-7. The tool information system (TIS) includes object domains OD-4 and OD-5.

The OIS contain a set of database files \{order, orderdet\}, and a set of procedures \{orderpas, metasys\}. Its main functions are to receive and edit order data, e.g. order number, part number, due date, and amount.

The PIS contains a database file \{part\} and a set of procedures \{partpas, selecttool, partedit, metasys\}. It possesses four functions: editing part data, invoking CAD system, invoking CAM system, and selecting tools.

The TIS contain a set of database files \{tool, toollist\} and a set of procedures \{toolpas, tooledit, reportt, metasys\}. It holds two functions: editing tool data and printing the tool report.

A simple example is employed to demonstrate the processes and functions of this system. First, an order “d0001” is entered in OIS as indicated in Fig. 9.

Next, the process plan of part ‘p0001’, shown in Fig. 10, includes center cutting, drilling, boring, and reaming.

Finally, the tools satisfying the process requirements are listed in the reporting file including four tools: a center-cut, a twist drill, a boring tool, and a reamer as indicated in Fig. 11. These results are consistent with relational data base scheme using
referential constraints.

2. Characteristics and Advantages of ROM/IDEF1X

ROM/IDEF1X adopts several models according to various development stages. Similar to the work of Dodani (1996), the development process is taken as a chain of applications of OO methodologies. Among these applications, three topics deserve our attention. (1) Determine the necessary models to represent a problem domain. (2) Select the development strategy: top-down or bottom-up. (3) Determine the kernels of models: function-centered or structure-centered.

ROM/IDEF1X utilizes functional and structural models to specify a proposed system. Behavioral specifications are implicit and lie in a functional model. ROM/IDEF1X adopts a bottom-up strategy to develop a target system. An object domain derived from the basic requirements is the counterpart of a module in the bottom-up strategy having a look-ahead procedure to reuse software components. In the opposite approach direction, e.g. Shayan and Liu (1995) propose a top-down strategy to develop a system. Since each development step of that strategy is primarily influenced by the previous stage, the reusability will be reduced (Mili et al., 1995).

Moreover, ROM/IDEF1X is function-centered since it is based on functional requirements originally. Cho and Lee (1999) apply IDEF0 to model the functional specifications and model the captured data and their semantic relationships using the IDEF1X data modelling method. By our approach, the functional requirements are clustered into an object domain and the structural data (IDEF1X) is derived from the functional data (ROM).

The significant characteristics and contributions of this research are summarized below.

(1) It provides designers systematic and comprehensive development procedures for a tool database system.

(2) It specifies the functional and structural requirements by ROM and IDEF1X.

(3) It establishes the relation schema for an IDEF1X data model that results in an algebraic manipulation of the IDEF1X data model.

VI. CONCLUSIONS AND SUGGESTIONS

This paper proposes a systematic development method for a tool database system. This method first organizes the basic requirement statements into object domains. Then, the structural relationships are represented in an IDEF1X data model according to the entities involved in ROM. This data model is transformed into relational schema thereafter. The tooling example presented in this paper indicates that this method is straightforward and easy to develop. ROM/IDEF1X is suitable in representing structural and spatial relationships, but it is inappropriate to describe behavioral and temporal relationships due to the constraints on ROM/IDEF1X. Thus, the characteristics of problem domains dominate the application area of this method. However, for modeling behavior specifications, state transition diagrams may be considered to integrate into the proposed approach.

Some of the following topics may deserve future research. The first one is to extend the capability of ROM/IDEF1X. For instance, as requirements change, ROM/IDEF1X must have the capability to accommodate these changes. The second thing is to apply ROM/IDEF1X to other database domains such as work-in-process, spare parts (Akcali et al., 1997), dies, fixtures (Senthil et al., 2001), and robot tools data base management systems since the ROM/IDEF1X approach is suitable for manufacturing systems that consist of large scale components. The last issue is to construct the mathematical foundation of ROM/IDEF1X.

ACKNOWLEDGMENTS

The authors would like to thank the Editor of the Information Eng. Section and the paper reviewers who, with their comments, enhanced the legibility of the manuscript. This work is supported partially by National Science Council, Taiwan, Grant No. NSC91-2213-E-239-007.
REFERENCES


FIPS 184, 1993, “Integration Definition for Information Modeling (IDEF1X),” National Institute of Standards and Technology, USA.


\[ R_i:Z_i \rightarrow X_i \] a set of functional dependencies

\[ R_j(A) \subseteq R_j(B) \implies R_j \] an inclusion dependency (ID)

\[ R_{d_j} \] requirements statement \( j \)

\[ R_{d_j} \{ G_{j,k} | G_{j,k} \text{= basic requirements, } j, k \in N \} \]

\[ S \{ O_i | O_i \text{= object domain, } i \in N \} \]

\[ \Pi_i(r_i) \] the projection of \( r_i \) onto the components \( A \)

\[ \subseteq \] subset of the Cartesian product of the attribute domain

Manuscript Received: Jun. 08, 2001
Revision Received: Apr. 15, 2003
and Accepted: Jun. 20, 2003