INFORMATION RESOURCES MANAGEMENT IN HETEROGENEOUS, DISTRIBUTED ENVIRONMENTS: A METADATABASE APPROACH

Cheng Hsu[1], M'hamed Bouziane[2], Laurie Rattner[3], Lester Yee[1]

April 1990

Revised February 1991

DSES Technical Report #37-90-230
Rensselaer Polytechnic Institute
IEEE Transactions on Software Engineering (Forthcoming)

Note: A preliminary and shorter version of this paper was presented at the 2nd International Conference on Computer Integrated Manufacturing, Rensselaer Polytechnic Institute, Troy, NY 12180, May 1990 [17].

[1]Decision Sciences and Engineering Systems Department
[2]Computer Science Department
[3]Center for Manufacturing Productivity and Technology Transfer
RENSSELAER POLYTECHNIC INSTITUTE
Troy, NY 12180-3590

ABSTRACT

Repository systems have emerged recently as a cornerstone to information management. Their envisaged purposes include (1) applications development in a CASE (Computer-Aided Software Engineering) environment, (2) enterprise information resources management (as a passive kernel for information integration), and (3) global information processing and management (as an active facilitator/integrator). Accordingly, their contents comprise metadata that model functional (sub-)systems and metadata that formulate systems interactions. In other words, the scope of metadata encompasses both data resources and knowledge, and hence requires new methods and techniques to manage these information resources in a unified way. Most previous metadata systems are limited to containing only data resources models.

A metadatabase system is being developed at Rensselaer for information integration in heterogeneous and distributed environments. This paper presents its core structure, the GIRD (Global Information Resources Dictionary) model for unified metadata representation and management (both data and knowledge). Current implementation, some illustrative examples, and a comparison with the IRDS model are also included.

1. INTRODUCTION

The notion of computerized enterprises is perhaps one of the most exciting and far-reaching visions that has emerged over the past decade for the role of information technology in modern organizations. In essence, we may argue, this notion envisions information as a fourth factor of production (along with land, labor and capital) and thereby shifts upward the entire production function of an organization; i.e., it fosters a quantum jump in its productivity. Examples range from specific approaches, such as Computer Integrated Manufacturing and Concurrent Engineering, to general corporate strategies for the employment and deployment of information systems technology.

1.1 The Role of Metadata

To fully implement this vision, however, new results for systems integration are necessary, especially in the areas of information integration and data and knowledge resources management. A key element in these areas is what we shall call enterprise *metadata*. Representative (previous) metadata systems include traditional data dictionary systems, the Information Resources Dictionary System (IRDS) of the National Institute of Standards and Technology (formerly NBS) [3; 10] and the latest repository systems developed by IBM and others. While most of the previous systems focus only on data resources management (including software development using CASE), some of the new repository systems (e.g., the one by IBM) are also designed for enterprise data integration.

However, to effect systems integration for computerized enterprises, the scope of metadata must be extended from simply representing data systems to synergistically including knowledge resources as well. These knowledge resources not only comprise the various knowledge-based systems pertaining to a computerized enterprise, but more importantly represent the functional contexts in which individual information (sub-)systems contribute to enterprise-wide synergy, or information integration.

In particular, a metadata model should contain at least three classes of knowledge: (1) operating knowledge (business rules) concerning individual data models or functional (sub-)systems (e.g., triggers, processes, and integrity constraints); (2) control knowledge for sequential systems interactions, including data transfer rules and global equivalence knowledge for all data items pertaining to the same logical object but implemented differently; and (3) decision knowledge for parallel

systems interactions, such as global decision processes and their implementation into information flows, localized decision rules, and control procedures. These aspects are not covered in most metadata systems; the majority of which are traditional data dictionaries.

1.2 Traditional Data Dictionaries

The role of data dictionary systems ranges from a conventional productivity tool supporting the life-cycle of database systems [1] to a tool for information resources management; an example of the latter is IRDS [7].

The contents of the data dictionary itself (i.e., metadata) generally include the schema information of the database(s) it describes and their respective data elements to facilitate software development and the control of data sharing among different programs and systems [22]. As such, most data dictionary systems are tightly coupled with the application databases and the metadata reside along side with the raw data. However, as the advent of modern information technology made information a major corporate resource, a higher level of management and control of information models (metadata) is needed [23; 24].

The IRDS effort was developed to address some of these issues by formulating a standard of data dictionary functionality required to qualify for use in a government information system [9]. As the new data dictionary standard, the IRDS incorporates a major intersection of features found in existing data dictionary systems and affords some expected benefits as a result of the standardization [10]. Accrued benefits of an IRDS type of system also include support of software engineering principles such as self-descriptiveness, cataloging computer programs, and on-line documentation.

Though comprehensive in its specification for a standardized data dictionary system, IRDS is not poised to handle aspects that are considered vital to enterprise information integration, namely knowledge models.

1.3 The Notion of Metadatabase

As the vision of information integration evolves, the focus has shifted from simply managing information systems to enhancing functional synergy of business and/or production systems. Accordingly, the emerging concept of repository systems calls explicitly for the inclusion of such knowledge resources as process models and business rules, as well as data models. Since these new systems were conceived and designed to operate independently for corporate

information resources management, as opposed to being tightly coupled with application databases, we refer to this class of metadata systems as a meta*database*.

More formally, a metadatabase in its own right is a (new type of) database integrated with a knowledge base. It aims to satisfy a diverse set of users across the enterprise with an unified architecture. First, there are enterprise users such as managers and systems developers who either need to maintain or want to reap benefits from enterprise information resources. The metadatabase supports these functions. In addition, the metadatabase is a repository of intelligence into which all user groups in different functional (sub-)systems or databases can tap for decision support or global query formation and processing. Last, but not least, in an active mode whereby all systems interact with each other, the metadatabase implements the above three classes of knowledge and is shared among (parallel) processes and automated systems to facilitate information integration. The metadatabase also has other characteristics of application databases, including metadata integrity and independence.

Three basic elements are necessary for the metadatabase environment: (1) a logical model to define and represent the metadata, (2) a physical storage structure derived from this representation, and (3) a metadatabase management system providing facilities for effective implementation and processing of metadata. All these elements involve data and knowledge integration.

It is worth noting that the contextual knowledge is an integral part of the information requirements model that specifies the structure, flow, and function of an organization. This knowledge should naturally be implemented in the repository of enterprise metadata to achieve global synergy, especially when local systems function in a heterogeneous environment (e.g., federated or composite systems [12; 25]; see also [20] for a sampler of systems). The importance of managing heterogeneous knowledge resources has only recently been explicitly formulated [14; 18]. Further, since organizational information requirements are often characterized by multi-stage decision processes, therefore the representation of the individual systems, combining their roles in the context of the enterprise with their implemented data and knowledge models is fundamentally important to system analysis and evolution.

1.4 A Unified Metadata Model

A global metadata model should, therefore, provide a consolidated representation of the various data and knowledge models at the enterprise level [14]. It should not only contain a common catalog encompassing local database schemata, but more significantly, it should also specify the intended contexts (i.e., the operating knowledge, control knowledge and decision knowledge mentioned above) within which individual systems operate and interact with each other. Some results to be gained from this model include a more cohesive information structure for organizations, information accessibility for decision making, and greater efficiencies in operations on data.

Toward this end, the Global Information Resources Dictionary (GIRD) model is developed for the metadatabase approach. Constructs of the GIRD model constitute a unified representation of major classes of metadata, with a design suitable for creating a metadatabase for distributed environments. The major classes cover system analysis-level functional models combined with logical design-level structural models and knowledge representation coupled with data resources. The support of heterogeneous databases by the GIRD includes directory (location) information, global equivalence of data items, and view integration.

The model itself is developed using the Two Stage Entity Relationship (TSER) model [11], which avails the needed modeling capabilities for the proposed metadatabase. Furthermore, the GIRD constructs are also represented by using TSER in a recursive manner, thereby simplifying the structure as well as facilitating its management through integrity constraints provided by the typing capabilities of the approach. The GIRD model can be implemented on a relational platform with some enhancements (e.g., constraints and file management), or it can be developed with a designated design in most environments. Compared to previous efforts, the GIRD model features (1) the inclusion of contextual knowledge, (2) a hierarchy of (modeling) constructs, and (3) self-descriptiveness.

1.5 Organization of the Paper

This paper focuses on the development of the GIRD model for information resources management through the metadatabase approach. Current implementation of the model first results in a stand-alone, passive metadatabase which, when properly connected with local systems, will simultaneously serve as

an active facilitator for enterprise information integration. The architecture for this active mode of metadatabase is beyond the scope of this paper.

Section 2 describes the GIRD representation method. It begins with a review of the TSER approach and includes a manufacturing example to illustrate the concepts. The conceptual development of GIRD and its structure are detailed in Section 3. The implementation of GIRD into a metadatabase is described in Section 4. To illustrate some major properties of the metadatabase model, and show how the GIRD elements fit together to deliver these properties, manufacturing information management examples are given in Section 5. Section 6 contains feature comparisons of the GIRD with the IRDS standard. Comments and conclusions, along with continuing research and development are covered in Section 7. Appendix 1 provides the schema for all meta-relations defined, and Appendix 2 is a glossary of attributes used in the schema.

2. THE REPRESENTATION METHOD: TSER

There are several requirements for an ideal metadata representation method that can be derived from the above discussion. The requirements include (1) the ability to represent the three major classes of knowledge (i.e., operating, control, and decision), (2) the inclusion of all pertinent models from a multi-stage analysis and design life cycle, (3) the neutral accommodation of heterogeneous data views (especially relations and object hierarchies), and (4) the unified representation of the full content of these metadata.

Concerning the representation of knowledge, a major element of these classes of knowledge is "flows" or dynamics that can be characterized with digraphs, such as data flows and control flows in the model. These global flows cannot be sufficiently represented through any implicit methods that embed knowledge into an encapsulation around data types. Such encapsulation methods are amenable mainly to localized triggers, contingency definitions, and other similar application logic. A combination of encapsulation and digraph techniques using certain generic primitives would suffice the need.

An objective for combining multi-stage models is isomorphism. That is, the mapping from a model at one stage of the life cycle to a model at the next stage should be complete and accurate. A corollary of this isomorphism is a reversible modeling process where, after the initial development of systems has long been completed, the information models that resulted from the initial analysis and design can be recovered from the metadatabase.

On the issue of data representation, a clear need is to encompass heterogeneity as exemplified by the relational approach and the object-oriented paradigm, two recent alternative models. It is evident that each has revealed certain unique but fundamental elements of the science of data modeling that have to be considered by any general model. In particular, the relational approach has, among other things, established the dependency theory for data integration (i.e., removal of redundancy) and the principle of separating applications and user views from common data structures [5; 6]; while the object-oriented paradigm asserts (or reasserts) the data abstraction hierarchy [19] and integration of certain classes of transactions with data modeling [21].

There is probably no ideal method existing presently that satisfies all of the above requirements. However, the GIRD scope and structure aim directly at *subjects* of databases and their *contexts* for information integration, leading to a particular representation method which is discussed below.

2.1 The Two Stage Entity Relationship (TSER) Approach

The TSER approach to information modeling is designed to represent complex and multiple systems [11; 13; 15]. This approach provides a representation method (modeling constructs and algorithms) and a modeling methodology, including both the contextual knowledge of an organizational information system and its data subjects. The TSER constructs are comprised of a functional (semantic) level model and a structural level model which are linked through dependency theory-based algorithms. These algorithms map the functional model to the structural model and ensure that the resulting structures are in at least third normal form. The integrity constraints implied by the typing of the TSER constructs are used to facilitate the management and control of the metadatabase itself.

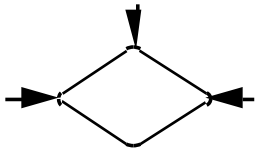
Underlying these constructs are two types of primitives: data items and predicate logic. Therefore, the basic structure of TSER metadata is characterized by (1) data representation as relations, (2) knowledge representation in the form of production rules, and (3) the two representations being tied via data items.

The TSER constructs are presented in Tables 1a and 1b. From the perspective of an information modeling process, the constructs may be described as follows. The constructs for functional modeling are called SUBJECTs and CONTEXTs and are used to represent data semantics and knowledge as viewed from an application level or a systems analysis perspective. The structural modeling con-

structs consist of one type of entity and three special types of associations that are used to refine the representation of data and production rules captured in the functional model for logic design. The metadata is represented by connecting the first two sets of constructs and structuring them into TSER meta-models; the result, along with other classes of metadata, is a metadatabase. This process, which corresponds to the common life cycle of systems analysis and design, is depicted in Figure 1, where a typical manufacturing enterprise is used for illustration purposes.

Table 1a: TSER Functional Model (SER) Constructs

User-oriented semantic-level constructs for object-hierarchy and process representation.

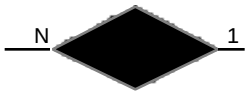
CONSTRUCT	PRIMITIVES	DESCRIPTION
SUBJECT (SE) 	**Contains data items (attributes), functional dependencies (among data items), *intra*-SUBJECT rules (characterized by data items belonging to a single SUBJECT), and class hierarchy (generalizes and aggregates SUBJECTs).**	**Represents functional units of information such as user views and application systems, and is analogous to frame or object. Triggers and dynamic definition of items are examples of *intra*-SUBJECT rules.**
CONTEXT (SR) 	**Contains *inter*-SUBJECT rules (characterized by data items belonging to different SUBJECTs), typically includes directions of flows for logic (decision and control) and data (communication, etc.)**	**Represents interactions among SUBJECT and control knowledge such as business rules and operating procedures and is analogous to process logic.**

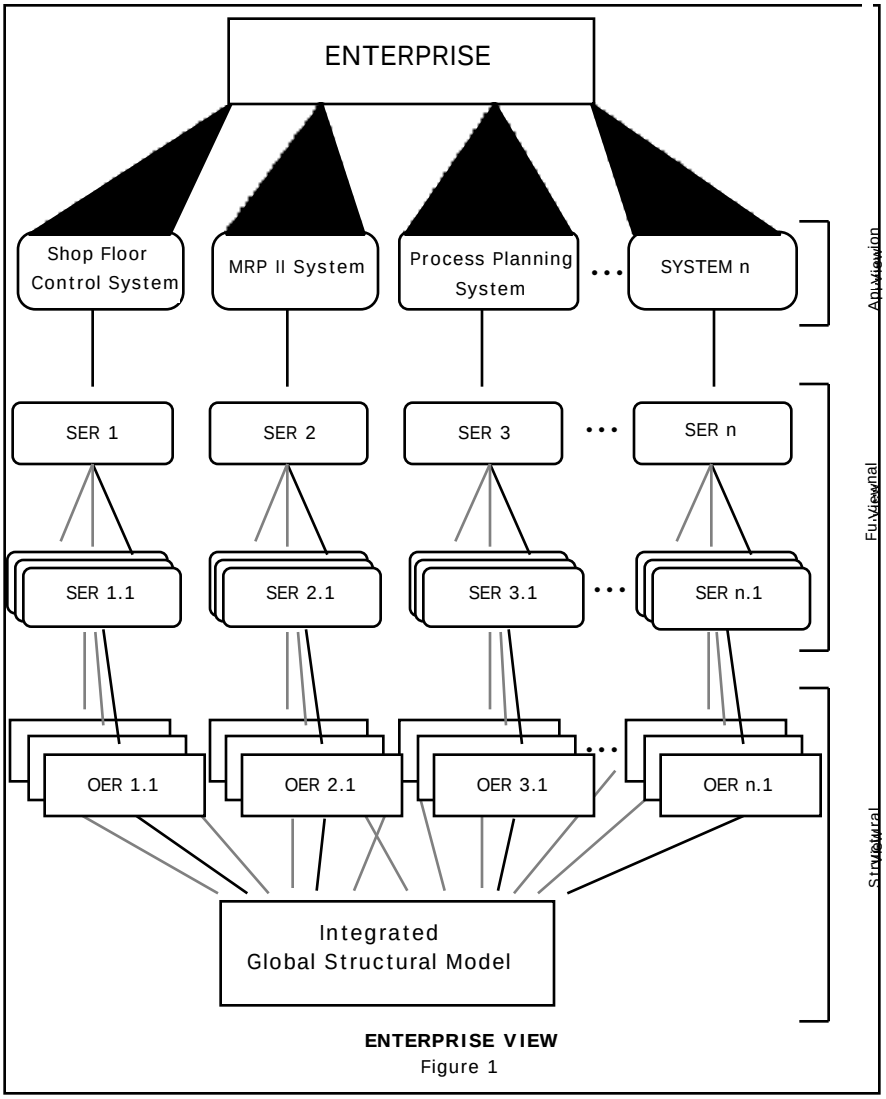
Note: (1) The full contents (as applicable) must be specified for all SUBJECTS at the leaf level of the SUBJECT hierarchy. The class hierarchy implies integrity rules for applications, but its presence is not required.

(2) Rules are constructed in the form of (a subset of) predicate logic where all clauses must only consist of the logical operators and the data items that have been declared or defined in the SUBJECTs, excepting certain key words such as *do* and *execute*. A data item may be defined to represent an executable routine, algorithm, or mathematical expression.

Table 1b: TSER Structural Model (OER) Constructs

Dependency-theoretic, neutral representation of data semantics and production rules for logical design.

CONSTRUCT	DEFINITION AND DESCRIPTION
OPERATIONAL ENTITY (OE) 	**Objects identified by a singular primary key, (optional) alternative keys, and non-prime attributes.**
PLURAL RELATIONSHIP (PR) 	**Association of entities characterized by a composite primary key and signifying a many-to-many and independent association.**
FUNCTIONAL RELATIONSHIP (FR) 	**A many-to-one association that signifies inheritance relationships. FRs represent the referential integrity constraint implied by the existence of foreign keys.** **The arrow side is called the *determined* and points to either an OE or a PR, while the other side is called the *determinant* and is also linked to either an OE or a PR. The primary key of the *determined* side is included as a non-prime attribute (i.e., a foreign key) of the *determinant* side.**
MANDATORY RELATIONSHIP (MR) 	**A one-to-many *fixed* association of OEs. MRs represent the existence-dependency constraint.** **The "1" side is linked to the *owner* OE while the arrow side points to the *owned* OE.**
Note: (1) In both top-down design and reverse engineering, the OER model is typically derived automatically from the SER model by using the TSER normalization algorithm. **(2) While there usually are multiple SER models representing different views or application systems of an enterprise model, there always exists only one integrated OER model for the global system.**	



ENTERPRISE VIEW
Figure 1

In the usual case where models of more than one system are developed, a three-step process for basic global information modeling is used. First, an SER for each application system is created; second, each of these SERs is mapped to its corresponding OER model; and third, the several OER models are consolidated into a single global OER model using dependency-theoretical principles (e.g., data normalization). When systems integration is actively formulated on top of this basic model, the second step would be expanded to provide a single SER model. That is, the several SERs are integrated into a global SER model by creating and populating *inter*-application CONTEXTs with the control knowledge and operating rules which define the interactions among application systems.

2.2 Shop Floor Control Case

To illustrate the TSER methodology, we show below a modeling example using a simple shop floor control system which is part of a computer integrated manufacturing (CIM) laboratory at Rensselaer Polytechnic Institute. There are five main subsystems comprising the CIM system: computer-aided product design, process planning, production planning manufacturing resources planning (MRP II), shop floor control, and work in progress tracking. Of particular interest for this example is the shop floor control system, which is itself comprised of three levels of hierarchical control: plant management, cell control, and workstation control. In this implementation, specialized workstation control logic has been developed for the material mover system — and thus this workstation is distinct in many respects from all other workstations.

Figures 2a-c show the SER hierarchy representing this CIM system. Figure 2a is the highest-level functional model of the CIM system. Note that each subsystem has been modeled as a SUBJECT with their respective interactions shown as CONTEXTs. When the SUBJECT "SHOP FLOOR CONTROL SYSTEM" from Figure 2a is decomposed, the result is a model of the three-tiered control hierarchy shown in Figure 2b. In this figure, the sub-systems are modeled as SUBJECTs and their interactions as CONTEXTs. Finally, the SUBJECT "WORKSTATION CONTROLLER" from Figure 2b is decomposed to expose the two workstation logic modules and their interactions; the result is Figure 2c.

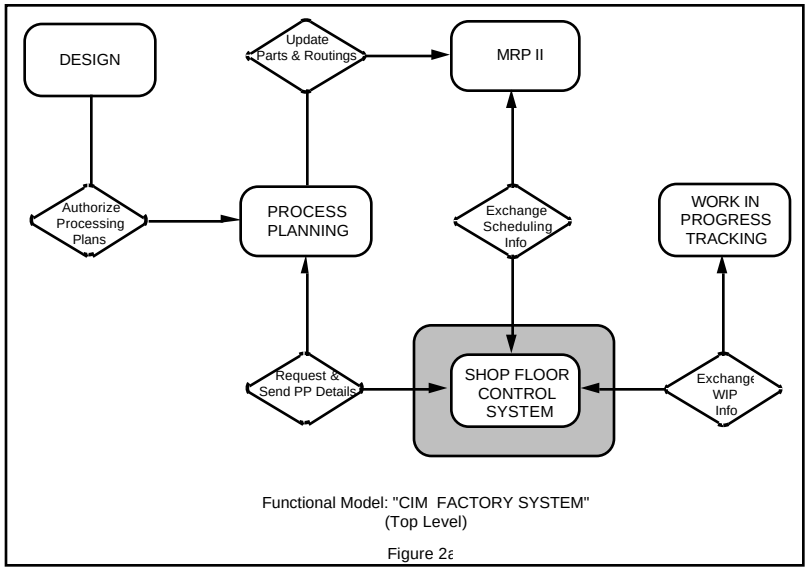
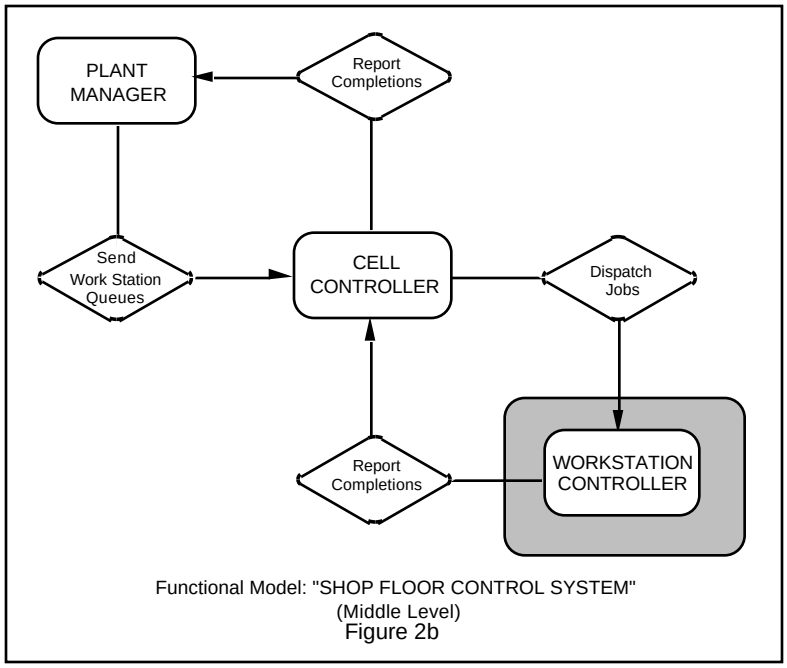
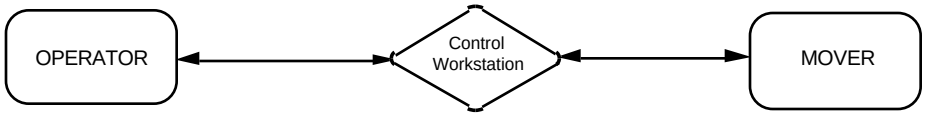


Figure 2a



Functional Model: "SHOP FLOOR CONTROL SYSTEM"
(Middle Level)
Figure 2b



FUNCTIONAL DEPENDENCIES:
WOID,SEQID --> WSID
PARTID --> RTGID, PARTNAME, PARTDESC
WSID --> WSSTATUS, WSNAME
WOID --> PARTID

FACTS:
Start_of_shift is-a context_variable
Start_of_shift = "08:00"
OPERATOR.WSStatus domain ("Inactive" "Busy" "Down")
OPERATOR.WOStatus domain ("Inprocess" "Complete" "Notstarted")
Do-compute-PP-filename isa procedure

RULES:
Rule #0001:
If System_time = Start_of_shift
Then Process(Next(OPERATOR.WOID))

Rule #0002:
If OPERATOR.WOStatus = "Complete"
Then DoNext(OPERATOR.WOID)

FACTS:
Do-wait-one-cycle is-a procedure
Do-call-mover is-a procedure
WSName domain ("Mover") +

RULES:
Rule #0012:
If OPERATOR.WOStatus = "Complete"
and OPERATOR.WSID = "X"
and MOVER.WSSTATUS = "Busy"
Then Do-wait-one-cycle(OPERATOR.WSID)

Rule #0013:
If OPERATOR.WOStatus = "Complete"
and MOVER.WSStatus = "Inactive"
Then Do-call-mover(OPERATOR.WSID)

FUNCTIONAL DEPENDENCIES:
FROMWS, TOWS --> MOVEDETAILS
PARTID --> PARTDESC
WSID --> WSNAME

FACTS:
MOVER.WSNAME = "Mover"

RULES:

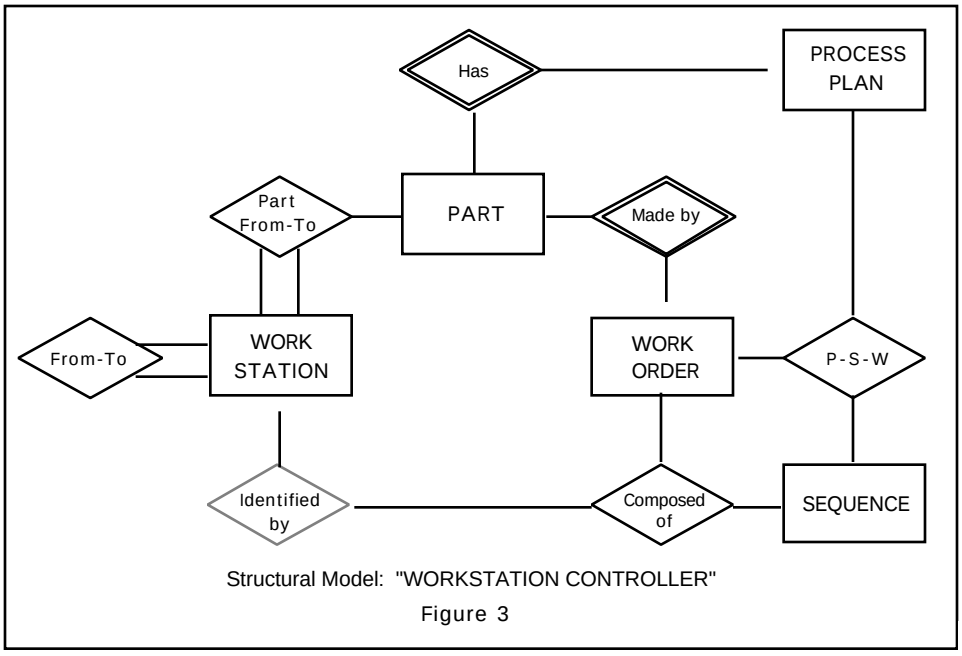
Functional Model: "WORKSTATION CONTROLLER"
(Bottom Level)

Figure 2c

Each of the CONTEXTs in these figures contains production rules and operating knowledge that define the dynamics between the SUBJECTs. Each SUBJECT, on the other hand, is characterized by an encapsulation of data semantics describing the statics of information. For simplicity, only the contents

of the constructs in Figure 2c are shown, however, as shown in the figure, all SUBJECTs contain data items, functional dependencies, and *intra*-SUBJECT facts and rules while CONTEXTs contain *inter*-SUBJECT knowledge.

Applying TSER normalization algorithms to the data items and their functional dependencies in the "MOVER" and "OPERATOR" SUBJECTs shown in Figure 2c (i.e., the "WORKSTATION CONTROLLER") gives rise to the OER submodel in Figure 3. Finally, Figure 4 shows the global integrated OER model which resulted from the consolidation of this "WORKSTATION CONTROLLER" OER submodel and the OER submodel resulting from the decomposition of "CELL CONTROLLER" (not shown).

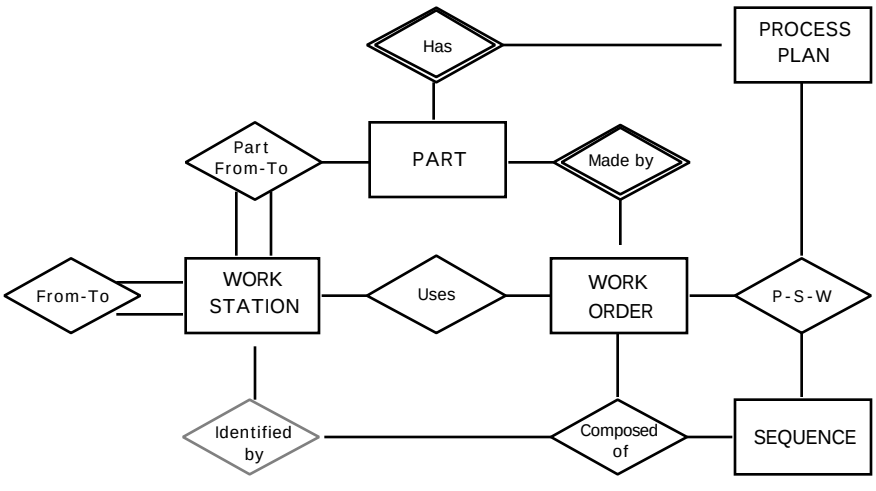


Structural Model: "WORKSTATION CONTROLLER"

Figure 3

Consolidation begins by identifying the identical constructs (i.e., those having the same primary key) within and across the OER submodels. Each such set of identical constructs is merged into a single construct. The second step in consolidation is representing the notion of foreign keys (referential integrity constraint) arising between constructs in different submodels (those arising within submodels have already been specified) by creating FRs. The individual and consolidated OER models are normalized structural models and are amenable to input to the (automated) mapping algorithms for schema generation. The same process would be applied to other subsystems to obtain an overall information model. All OER models for these subsystems would be consolidated to yield a global integrated model.

All of the above models give rise to metadata which, in their own right, are complex and need to be modeled or represented. The structuring of metadata is discussed in the next section.



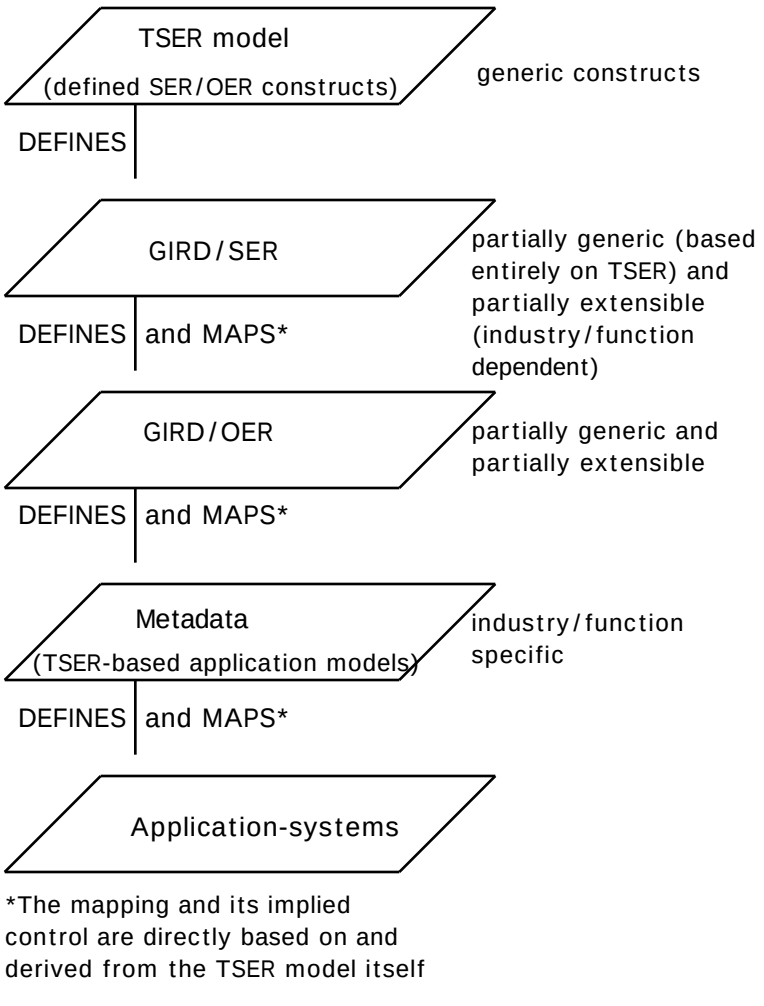
Consolidated Structural Model: "SHOP FLOOR CONTROL SYSTEM"
Figure 4

3. THE GLOBAL INFORMATION RESOURCES DICTIONARY (GIRD)

The GIRD model is constructed as the logical design for a stand-alone (passive mode) metadatabase, which, in turn, is the first phase in the development of an active metadatabase that implements the knowledge (see Section 1) to directly facilitate the integration of local systems. Therefore, the GIRD model is developed not only to support passive metadata applications, but also to effect functionalities needed in global query processing or system interactions.

3.1 Overview of the GIRD Model

The GIRD model is represented according to the TSER approach described in Section 2 above. As such, TSER is used in two planes. First, it is used to describe each application system within the enterprise, and therefore, each such system has an SER and an OER model (e.g., the example in Section 2.2). These models constitute the major contents of the metadatabase. Secondly, however, TSER is used to abstract the metadatabase contents into a generic metadata model: the GIRD model. Thus there are both an SER and an OER model for the metadatabase itself; together they constitute the GIRD model. By using these two planes, a definitive structure consisting of five layers is determined, thereby achieving a desirable form of self-descriptiveness.



The Logical Layers of the GIRD Concept
Figure 5

As a preview, Figure 5 shows the five layers of the GIRD modeling concept. The TSER constructs are used recursively at each layer to achieve both typing and self-description for metadata management. This structure is essentially generic and deterministic, since it is derived directly from the TSER model itself. Nonetheless, a part of the GIRD structure (i.e., "RESOURCES-VIEW" of the GIRD/SER and its corresponding HARDWARE RESOURCE and SOFTWARE RESOURCE of the GIRD/OER — see Section 3.2 below) is extensible in the sense

that it may be changed to accommodate other views or requirements of particular industries or functions.

Any such changes would result in a slightly different GIRD structure, but nevertheless, a structure that should be able to retain all of the design properties so long as the same process is used as described in this section. For clarity, we shall refer to the TSER constructs that are used to represent metadata in the GIRD layers as *meta*-constructs; e.g., meta-subject, -context, -entity, -relationship, and -attribute, respectively.

3.2 The GIRD/SER Layer

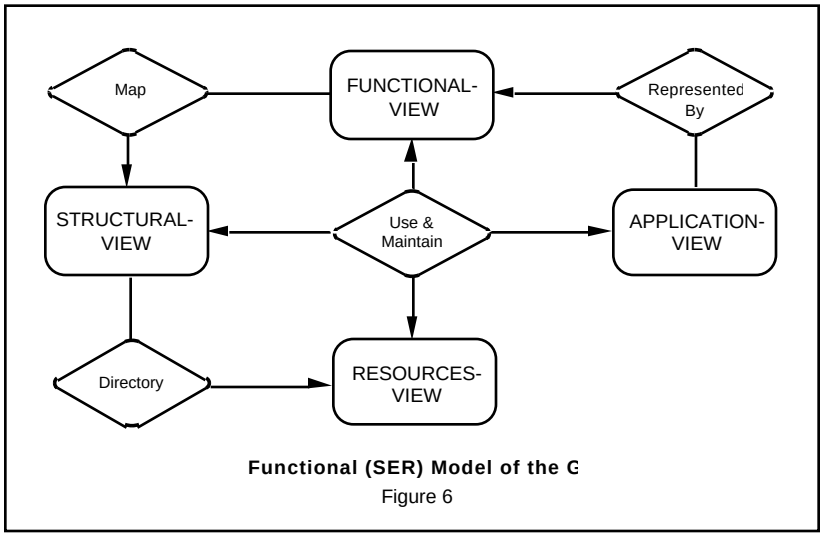
The contents of this layer are depicted in Figure 6. These contents are derived directly from the general modeling process shown in Figure 1.

As depicted in this figure, an enterprise can be viewed as a number of application systems (view 1). Since each system is represented by an SER model, another view of the enterprise is the set of SER models (view 2). Similarly, since each such SER is mapped to a corresponding OER data-structure model, the enterprise may be viewed as a set of OER models that are integrated into one global OER (view 3). Each of these three views is essentially a category of metadata determined entirely from the model constructs themselves, which in turn represent the general stages in the systems analysis and design process [13]. These categories, along with a fourth one representing hardware and software resources in the enterprise, constitute an SER-level schema for the metadata. Each category is modeled as a meta-subject in Figure 6. The only meta-subject that is not rooted in the model itself is "RESOURCES-VIEW", which, however, is arguably always implied in an information model.

In sum, the information resources are represented at the SER level using four meta-subjects ("APPLICATION-VIEW", "FUNCTIONAL-VIEW", "STRUCTURAL-VIEW", and "RESOURCES-VIEW") and five meta-contexts specifying the interactions among the meta-subjects. The contents of these meta-subjects comprise information resources determined from three sources: the literature (e.g., IRDS schema), target modeling paradigms (i.e., relational, object-oriented, and TSER), and empirical studies (viz., manufacturing).

The functional modeling logic and rules are contained in the meta-context, "Represented By"; meta-context "Map"stores the mapping algorithms (i.e., dependency theory and heuristic rules) that produce data structure (OER) models from SERs. The contents of both these meta-contexts are stored in files using routines

and programs; in the future, their knowledge will be modeled explicitly. The meta-context "Directory"contains the knowledge about where OER files are stored. The remaining "Use & Maintain"meta-context specifies security and maintenance knowledge.



Functional (SER) Model of the G

Figure 6

3.3 The GIRD/OER Layer

The TSER methodology is then applied to further develop the GIRD model. Each meta-subject is first mapped to its OER submodel, and then the submodels are consolidated into an OER model — this model is the integrated representation for the global metadatabase. This representation, in turn, is mapped to a schema design in a target software environment for implementation as an enhanced database by itself. Because the TSER methodology is employed, the submodel data structures (and therefore those in the consolidated GIRD OER) are in at least third normal form.

To illustrate the entire structure, the submodels corresponding to each of the four meta-subjects (views) are described individually, and then the consolidated model is discussed. The schema for each meta-entity and meta-relationship is listed in Appendix 1 with all meta-attributes defined in Appendix 2.

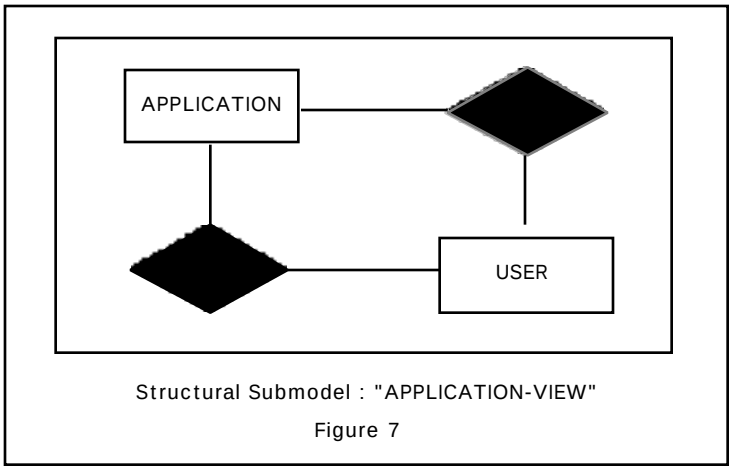
3.3.1 "APPLICATION-VIEW" OER Submodel

The "APPLICATION-VIEW" meta-subject permits a high-level view of the enterprise. Objects of interest in this view are the application systems comprising the enterprise and their associated end users. This view is reflected by the submodel mapped from the "APPLICATION-VIEW" meta-subject; the submodel

contains the meta-entities APPLICATION and USER. USER describes classes of users and groups in the organization. As shown in Figure 7 and described in Appendix 1, there are three general classes of users for whom metadata is maintained: (1) information administrators modeled with the ApplMaintain meta-Functional Relationship (FR), (2) end users (see the ApplUsers meta-Plural Relationship (PR)), and (3) information modelers (see meta-attribute addedby in all other meta-relations).

The meta-PR ApplUsers identifies the various end-users of each application with their respective access codes and passwords, while the meta-FR ApplMaintain indicates the "information administrator" type of user responsible for maintaining the applications. (Note that neither meta-FRs nor meta-Mandatory Relationships (MRs) are implemented as base relations; only their implied integrity constraints are implemented.)

In the modeling example, five applications were depicted in Figure 1a: Design, Process Planning, MRP II, Shop Floor Control, and Work in Process (WIP) Tracking. These are instances of the meta-entity "APPLICATION-VIEW" in the GIRD.



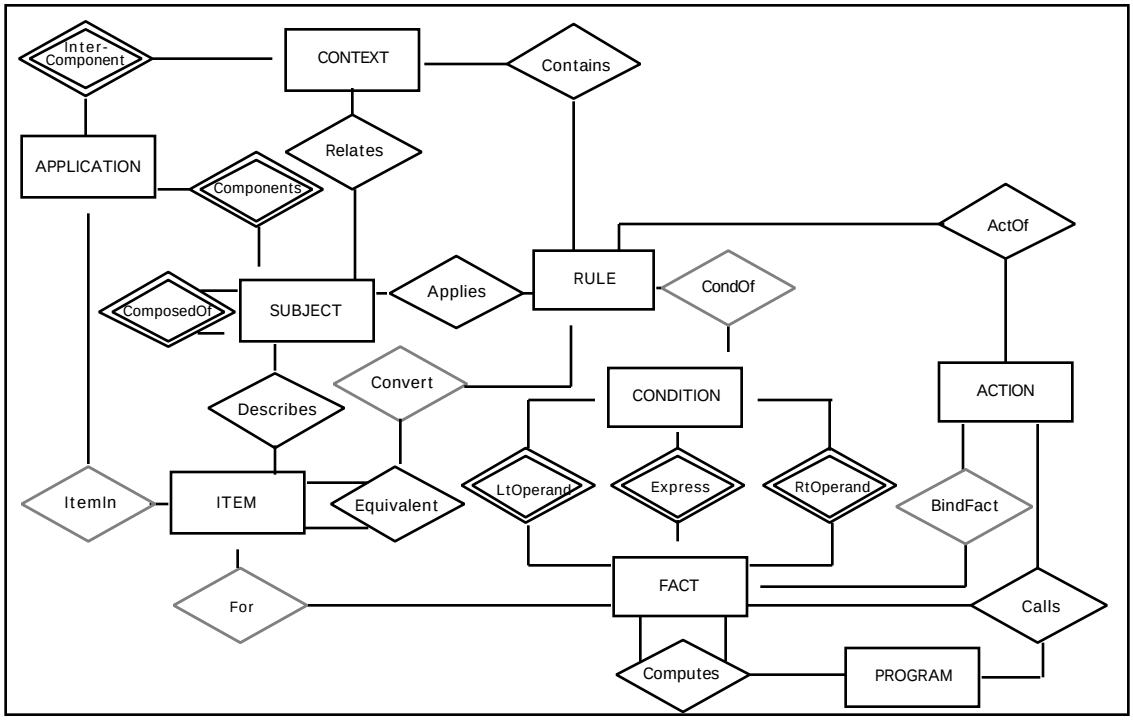
Structural Submodel : "APPLICATION-VIEW"

Figure 7

3.3.2 "FUNCTIONAL-VIEW" OER Submodel

The meta-subject, "FUNCTIONAL-VIEW" in Figure 6 captures the semantic or functional level modeling (SER) constructs used in the enterprise. Therefore, its structural design (i.e., the OER submodel shown in Figure 8) is based on the primitives defining these constructs (see Table 1a). In the figure, the meta-entities SUBJECT and ITEM, together with the meta-PR Describes — which

specifies the data items that are aggregated into each SUBJECT — represents the data contents of a functional model.



Structural Submodel: "FUNCTIONAL-VIEW"
Figure 8

The meta-entity CONTEXT represents the knowledge among SUBJECTs, while the meta-PR Relates shows which CONTEXTs contain the rules governing the interactions among sets of SUBJECTs. Thus, each CONTEXT represents dynamic knowledge and contains a set of production rules which, themselves, are instances of the meta-entity RULE. This association between CONTEXTs and rules is represented by the meta-PR Contains. Similarly, each SUBJECT also Applies rules that pertain only to itself (i.e., these are *intra*-subject rules in contrast to the *inter*-subject rules contained in the CONTEXT).

The basic building block for this rule representation is the meta-entity FACT. This meta-entity represents dynamic (or, run-time) interpretations of data items in specific problem domains and is the basis for an inference engine to trigger and fire the rules. A fact is either a simple value (the value of a persistent database item or a constant) or a computed value (the result of a function call or an expression evaluation). Accordingly, in addition to a system generated identifier, a fact is characterized with the meta-attributes factname and facttype. A factname is

an Itemcode if the fact corresponds to a constant or a database item, or an expression if the fact is the result of an expression evaluation. The factvalue meta-attribute represents the value of the fact and facttype indicates these types (see Appendix 2).

Each rule has the basic form IF <Condition> THEN <Action> where <Condition> is an expression whose value is either TRUE or FALSE. The expression is a logical or arithmetic operation between two facts (the left and right operands of the operation). Thus, the meta-entity CONDITION consists of meta-attributes Condid, leftfact, operator, and rightfact. Condid is an identifier of the condition, leftfact is a Factid representing the value of the left operand, operator is an arithmetic or logical operator, and rightfact is another Factid representing the value of the right operand.

Meta-MRs LtOperand and RtOperand represent the two foreign keys leftfact and rightfact (due to Factid) in CONDITION, respectively. The meta-FR For links facts of FACT with items of ITEM and represents a unifying link between rules and data. This also represents the first type of facts (simple value facts). The second type of facts, which takes as its value the result of an expression evaluation, is represented by the meta-MR Express between CONDITION and FACT. This case also includes those facts whose values are computed by executing functions or routines. The trinary meta-PR Computes represents this class of facts evaluation and consists of Factid, functid, parid, and parorder meta-attributes. Functid identifies the function (represented in PROGRAM) to be executed for that Factid. Parid (defined on the same domain as Factid) and parorder correspond respectively to the function parameters and their relative order during the call.

Note that the three meta-MRs LtOperand, RtOperand, and Express collectively constitute a recursive definition of expressions (i.e., while containing sub-expressions for condition, each in its own right also represents expressions that are defined in the same way).

The conclusion of a rule consists of one or more actions. Each action (represented by the meta-entity ACTION) binds a fact (assigns a value to a fact; either a simple value or more complex expression value) or calls a procedure. These two types of actions are represented by the meta-FR BindFact and the meta-PR Calls, respectively. An acttype meta-attribute, which identifies the type of the action, is included in ACTION meta-entity. The meta-FR BindFact also provides

an explicit representation of the chaining mechanism for rules to be fired and thus can result in an efficient rule processing facility.

The actions are linked to the proper rules by means of the meta-PR ActOf, while conditions are linked through the meta-FR CondOf. The difference in such linking is that a rule can have many actions, each of which can be involved in one or more rules; thereby establishing a many-to-many relationship. The conditions of a rule, on the other hand, must be related together through logical operators "AND" and "OR" and thus form one condition which can be involved in many rules, and hence the many-to-one relationship.

This design in its own right provides a basis for implementing the concept of rulebases (see [14] for a discussion of this concept). This concept, succinctly put, addresses the consolidation and management of some basic business rules or system knowledge shared either by multiple knowledge-based systems or by different application programs.

The static knowledge is comprised of structural and equivalence information. The hierarchy with which SUBJECTs are often modeled (applying the usual notions of aggregation and generalization) is captured with two meta-MRs: ComposedOf and Components. This hierarchy may be characterized as a tree whose root node corresponds to the meta-entity APPLICATION and whose stems and leaves correspond to meta-entity SUBJECT. The meta-MR Components identifies the SUBJECTs that are aggregated into each application, while ComposedOf represents the hierarchy among SUBJECTs. The meta-MR InterComponent represents the link between an application and each of its CONTEXTs. Synonyms among data items are represented with the meta-PR Equivalent which not only provides information about which data items are logically equivalent, but also the necessary conversion rules if these items have different formats. This conversion is represented by the meta-FR Convert. By using normalized data structures, the GIRD maintains naming consistency and transparency to users across the organization.

It is worth noting that, in support of local autonomy in determining meaningful names, this representation does *not* require unique names for data items. Instead, items are uniquely identified by a metadatabase-system generated Itemcode.

In the modeling example, the Shop Floor Control components are "PLANT MANAGER", "CELL CONTROLLER" and "WORKSTATION CONTROLLER" (refer to Figure 2b). The "WORKSTATION CONTROLLER" is ComposedOf

"MOVER" and "OPERATOR" (Figure 2c). "MOVER" represents the material mover workstation, and "OPERATOR" represents all other workstations.

Examples of the use of knowledge constructs are shown in the rules depicted in Figure 2c. The CONTEXT "Control WS" contains Rules #0012 and #0013. "OPERATOR.WO-STATUS = 'complete' " is a condition of Rule #0012, and this same condition has a FACT which is For ITEM "OPERATOR.WO-STATUS".

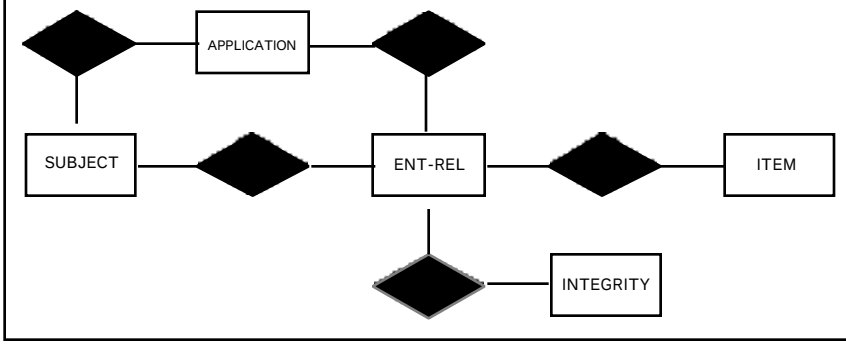
3.3.3 "STRUCTURAL-VIEW" OER Submodel

The meta-subject "STRUCTURAL-VIEW" represents the structural model (i.e., the OER model) underlying each application and the global enterprise information model that results from their integration. Basically, the four types of OER model constructs (see Table 1b) can be grouped into two major classes according to their respective functionality. Shown in Figure 9, the first class, ENT-REL, contains all constructs of the type OE or PR and reflects the normalized (in at least Boyce-Codd normal form) structure for data pertaining to an application. The second, INTEGRITY, includes sets of integrity constraints specified by the OER constructs and user-defined constraints; i.e., it contains all constructs of the type FR and MR. The meta-FR ERExist enforces the constraint that for an integrity constraint to be declared between two ENT-RELs, the ENT-RELs themselves must exist.

As described in Section 2, an OER model is derived from its corresponding SER model by mapping algorithms. Specifically, each SUBJECT in the SER model is mapped into a set of OER constructs (i.e., OE, PR, MR, and FR), then the data items of the SUBJECT are assigned to OER constructs according to dependency and normalization theory. These two steps are represented by the meta-PRs MappedTo and BelongTo, respectively. The meta-PR MappedTo between SUBJECT and ENT-REL links each SUBJECT with its corresponding set of OER constructs. Similarly,BelongTo, between ITEM and ENT-REL, represents the assignment of items to entities and relationships.

It should be noted that during the mapping from SUBJECTs to OER constructs, any entities (or relationships) with identical primary keys are merged. This is the basic mechanism by which the integration of views, submodels and models is ultimately achieved. However, since the entity (or relationship) could have different names in each local application system, the meta-PR NamedAs is developed between ENT-REL and APPLICATION. Thus, NamedAs provides the mechanism to store a single logical object while maintain-

ing its various names in local applications. This capability enables the metadatabase to facilitate view/model integration.



Structural Submodel: "STRUCTURAL-VIEW"
Figure 9

3.3.4 "RESOURCE-VIEW" OER Submodel

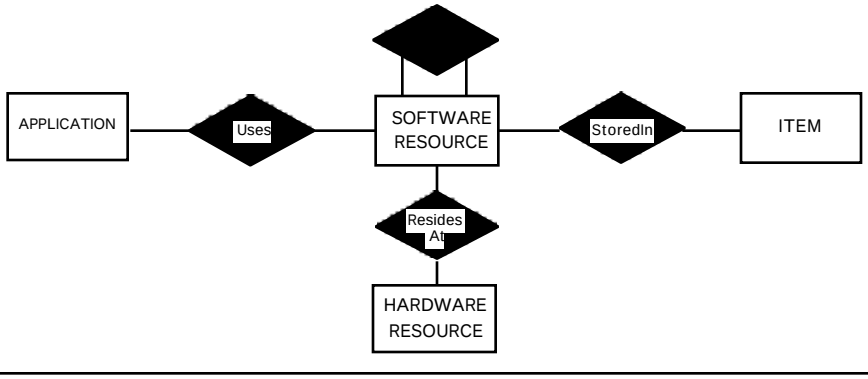
The fourth, and last view of metadata in the GIRD model represents the resources used in an enterprise to operationalize its various applications. Two types of resources are identified and represented in Figure 10: HARDWARE RESOURCE, and SOFTWARE RESOURCE. HARDWARE RESOURCE describes the physical components of the organization's information resources, while SOFTWARE RESOURCE represents the many types of software resources such as files, documents, and programs. Included in the definition of software are the files that store data, and this notion is represented by the meta-PR StoredIn between ITEM and SOFTWARE RESOURCE. The meta-PR ModuleOf is a recursive relationship representing the fact that a software resource may use or call other software resources to perform some of its activities. The meta-PR Uses depicts the many-to-many relationship between applications and their respective software resources. Finally, the meta-PR Resides-at indicates that a particular software resource may be resident on different hardware resources, and, at the same time, each location may house multiple software resources.

We might emphasize that the local schemata are represented in and through the metadata in this submodel, especially ITEM, StoredIn, SOFTWARE RESOURCE, and ModuleOf, along with the structural submodel discussed above In other words, each logical data object represented in the metadatabase (e.g., in ITEM) is also linked to its local implementation (e.g., as a base relation in Oracle or a file under MVS) through the GIRD model. Thus, no mappings or

conversions between any schemata (local or global) are needed in the metadatabase approach. This way, the heterogeneity of local data models is dealt with directly without requiring an "integrated (global) schema," which not only is difficult, but is also restrictive or even contradicting with the notion of heterogeneous, distributed environments.

3.3.5 Consolidation of the Four Submodels

All the submodels are consolidated using the TSER rules (see Section 2.2 for an example). Figure 11 shows the consolidated model, with the complete description of all meta-entities and -relationships given in Appendix 1. Meta-attributes provide documentation and audit trail information for the meta-entities and -relationships; they are defined in Appendix 2.



Structural Submodel: "RESOURCES-VIEW"
Figure 10

In consolidating the GIRD submodels, four meta-OEs have been merged: APPLICATION (occurs in all four submodels); SUBJECT (occurs in "FUNCTIONAL-VIEW" and "STRUCTURAL-VIEW" submodels); ITEM (occurs in "FUNCTIONAL-VIEW," "STRUCTURAL-VIEW," and "RESOURCES-VIEW" submodels); and finally, PROGRAM (in the "FUNCTIONAL-VIEW" submodel) was merged with SOFTWARE RESOURCE (in the "RESOURCES-VIEW" submodel). Further, after consolidating identical constructs, meta-FRs are created to represent referential integrity constraints across submodels. In this case, it is the meta-FR HardMaintain between USER and HARDWARE-RESOURCES.

3.4 Summary

Based on the above discussions, several salient characteristics and design properties of the GIRD model are summarized below (see also Figure 11).

(1) A hierarchy of models (as represented by the four meta-subjects) corresponding to the usual life cycle of systems analysis and design are covered and stored in a structured way.

(2) Three basic dimensions of data representations found in representative data models are included; namely, dependency theory-based relations, integrity constraints, and data abstraction hierarchy.

(3) By virtue of intra-subject rules (encapsulation), inter-subject rules grouped into contexts, and digraphs (flows), the knowledge representation encompasses both static and dynamic dimensions of the major classes of knowledge required for information integration (see Sections 1 and 2).

(4) The classes of metadata are unified to the extent of the GIRD/OER model that connects all its elements through two interrelated primitives: data items and predicate logic (in the form of production rules).

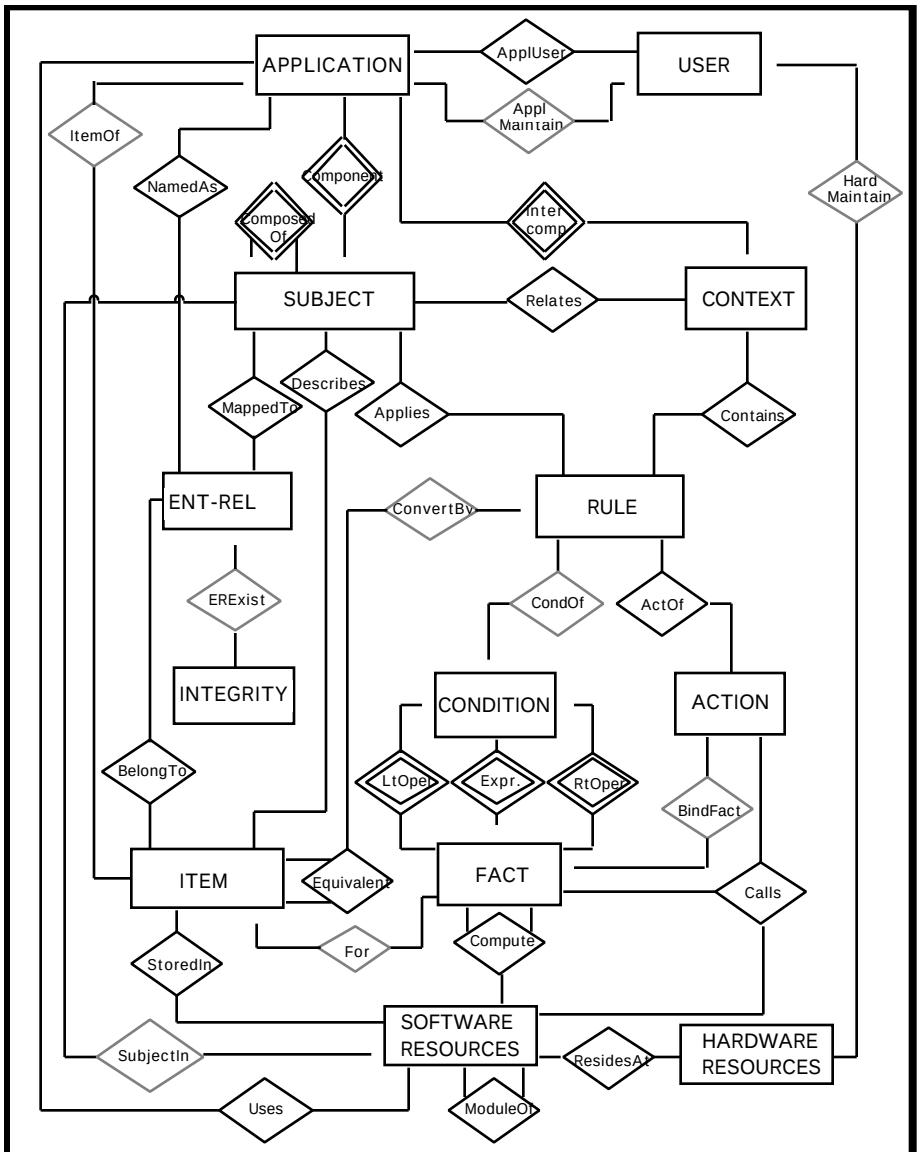
(5) The GIRD model achieves a fundamental level of capacity for both representing some heterogeneity and facilitating information integration (through e.g., equivalence knowledge).

(6) Integration of heterogeneous data models is achieved purely at a logical level (i.e., via the enterprise information model) and implemented by virtue of links represented in the GIRD model, thus it does not require physical integration of local schemata.

(7) The basic structure of GIRD is rooted in TSER and derived directly from the modeling logic; therefore, it is self-descriptive (in the sense of recursive definition as IRDS advocates) up to the TSER definition itself.

(8) When TSER is employed for information modeling, and therefore provides metadata for populating the GIRD structure, the modeling process is reversible; i.e., the original functional and structural models can be reproduced directly and completely from the metadatabase.

We might stress that, although the GIRD model is developed using TSER, it can be implemented without having to use TSER as the modeling methodology. The structure, definition, and conventions of GIRD become independent after its development is completed. The next section provides details on how this model can be implemented in a typical environment.



Structural Model: GLOBAL INFORMATION RESOURCES DICTIONARY (GIRD)

FIGURE 11

4. GIRD IMPLEMENTATION: A METADATABASE

Basically, the GIRD model (see Figure 11) calls for four classes of computing capabilities to operate fully: (1) meta-relation processing to implement all meta-entities and -relationships (e.g., basic relational technology), (2) rule processing to fire the rules represented in the model (e.g., the rule-based techniques), (3) integrity control to enforce the links among meta-relations (e.g., data management shells), and (4) routine management to execute the programs and function calls (e.g., system programming modules). These capabilities require common techniques that are readily available from the software industry.

Further, it should be pointed out that rule processing and routine management are not required for a passive metadatabase where rules are simply stored for querying and are not fired. Storage and display of rules can be done in the same manner as meta-relations.

Therefore, there can be a number of practical strategies for metadatabase implementation in a typical enterprise; the choice depends mainly on the usual criteria of cost-benefits and efficiency vs effectiveness. A simple approach is to employ as the engine a database management system that provides extended relational capabilities, runs under an operating systems with sufficient file management functions, and interfaces with rule-based languages or their equivalent. Although custom-designed software shells would still be needed to put these components together and to provide a user interface, the effort will, at most, amount to a moderate software engineering endeavor.

A somewhat designated approach using custom architecture and software was adopted to develop the metadatabase prototype at Rensselaer. This strategy takes advantage of TSER and the particular computing environments at hand. This implementation is detailed below.

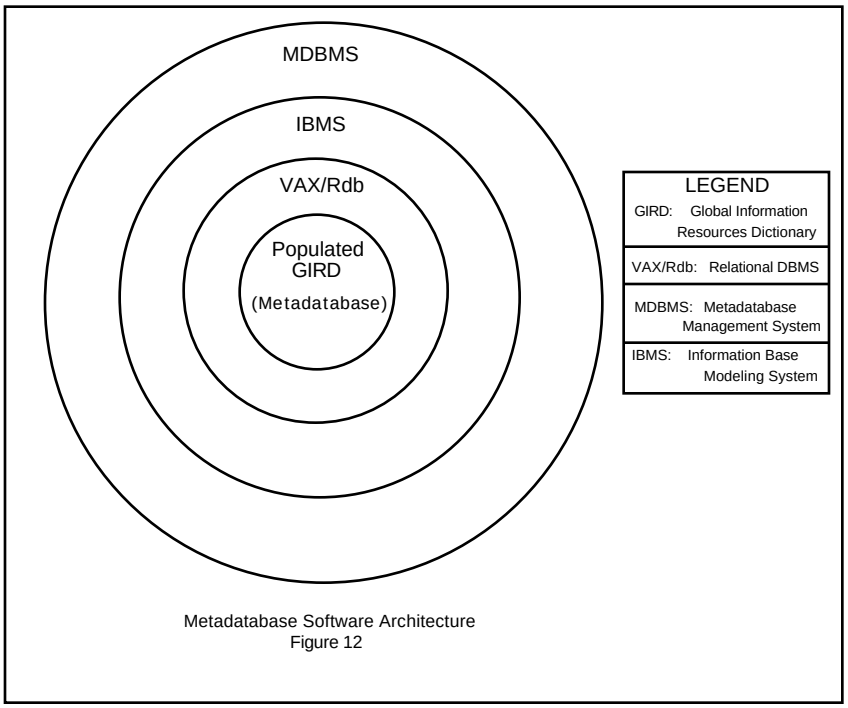
4.1 A Custom Design Using a Relational Platform

The metadatabase system is presently implemented using Rdb (an extended relational database management system) coupled with a LISP environment on a Digital Equipment Corporation MicroVAX computer. Figure 12 depicts its current architecture.

At the core is the populated GIRD model; i.e., the passive metadatabase. The entire metadatabase schema and most of the metadata are managed using Rdb according to the GIRD model, while routines and other software resources are managed through the VAX operating system. The metadatabase schema includes both data structures and integrity rules and is expressed in terms of Rdb DDL statements (which include some integrity constraints).

The schema design is derived entirely from the GIRD/OER model (see Figure 11) according to TSER definitions. To facilitate this effort, however, a custom tool was used for the Rensselaer prototype. This tool, Information Base Modeling System (IBMS), is a CASE implementation for TSER [16] and is coded in LISP and Pascal. A schema generator (Rdb/DDL and SQL/DDL) that maps the GIRD model into an implementation system is included in IBMS. The metadatabase prototype

was populated with metadata from the information model created with IBMS. The same logic, however, can be performed without using IBMS.



Metadatabase Software Architecture
Figure 12

Access to the metadatabase is through its management shell (Metadatabase Management System - MDBMS) built on Rdb plus LISP and C. For metadatabase users, MDBMS would be the only interface needed; IBMS would be invoked mainly for tasks that involve modeling (e.g., creating or evolving the enterprise information system). In addition, this management system is designed to provide rulebase processing capabilities as well as to support metadata applications. A menu-driven query system is currently in place for this purpose.

The components of the software architecture are discussed below.

4.2 Rdb Schema Design for the GIRD Model

The GIRD/OER structure (Figure 11) can be interpreted into two classes of relational constructs: (1) relations; i.e., the data structures pertaining to the meta-entities and -relationships, and (2) integrity constraints.

The meta-entities and -plural relationships (PRs) in the model are mapped one-to-one onto base relations in the relational schema. Specifically, for each of them a base relation having exactly the same name, primary key(s), and attributes (as those in Appendix 1) is defined in the schema. Meta-functional relationships (FRs) result in the implementation of referential integrity

constraints, while the meta-mandatory relationships (MRs) result in existence dependency constraints. As a sample, Table 2 shows how the meta-entities RULE and CONDITION and the meta-FR CondOf are translated into Rdb schema. The meta-attributes included in each relation have been previously defined using Rdb "DEFINE FIELD" statements.

```
DEFINE RELATION RULE.                    DEFINE RELATION CONDITION
    RNAME.                                   CONDID.
    RTYPE.                                   LEFTFACT.
    DESCRIPT.                                OPERATOR.
    CONDID.                                  RIGHTFACT.
    ADDEDBY.                                 ADDEDBY.
    DATEADDED.                               DATEADDED.
END RULE RELATION.                       END CONDITION RELATION.

DEFINE INDEX RULE_KEY.                   DEFINE INDEX CONDITION_KEY.
    FOR RULE                                 FOR CONDITION
    DUPLICATES ARE NOT ALLOWED.              DUPLICATES ARE NOT ALLOWED.
    RNAME.                                   COND-ID.
END RULE_KEY INDEX.                      END CONDITION_KEY INDEX.

DEFINE CONSTRAINT CONDOF
    FOR X IN RULE
      REQUIRE ANY Y IN CONDITION
         WITH X.CONDID = Y.CONDID
CHECK ON COMMIT.
```

SAMPLE RDB DDL STATEMENTS
TABLE 2

The four classes of meta-relations imply four classes of integrity rules governing the processing of the metadata instances. The rules are listed below; the first three are analogous to the usual relational integrity rules [5], but they are more exact because of the meaning of the OER model itself.

Entity Integrity: No meta-entity base relation may accept null values for its primary key nor can the primary key accept duplicate values. For example,

```
DEFINE CONSTRAINT RULE_KEY_INT
    FOR X IN RULE
    REQUIRE NOT X.RNAME MISSING
CHECK ON COMMIT.
```

specifies the entity integrity for the relation RULE, non-duplication is handled by the index definition (see Table 2).

Associative Integrity: In a meta-PR base relation, the value of each attribute in the primary key must exist as a value in at least one tuple of its associated meta-entity relations.

```
DEFINE CONSTRAINT ACTOF_ASSOC
  FOR X IN ACTOF
  REQUIRE ANY Y IN ACTION WITH
    X.ACTID =Y.ACTID
  AND ANY Z IN RULE WITH
    X.RNAME = Z.RNAME
  CHECK ON COMMIT.
```

Referential Integrity: If a meta-entity/-PR is a *determinant* of some meta-FR, then every value of the foreign key in its base relation must either be null or match some value of the corresponding primary key of the *determined* meta-entity/-PR base relation.

```
DEFINE CONSTRAINT CONDOF
  FOR X IN RULE WITH X.CONDID NE ""
  REQUIRE ANY  Z IN CONDITION WITH
    X.CONDID = Z.CONDID
  CHECK ON COMMIT.
```

It should be noted that, albeit supported by Rdb, the construct "CONSTRAINT" in the above algorithms is not commonly found in many other relational systems. More programming effort would be needed to implement a comparable capability in its place.

Existence Dependency: Every meta-MR base relation must obey the associative integrity rule plus one additional requirement. Each instance in a meta-entity relation that corresponds to the *owned* component of a meta-MR base relation must contain only values which match a value in at least one instance of the associated meta-MR, unless the *owned* component is further involved with a meta-PR and follows the associative constraint implied by that meta-PR. The existence dependency requirement cannot be implemented directly in Rdb, and is implemented in the management system shell.

4.3 Populating the Metadatabase

Once the appropriate base relations have been created, view and functional model information (from the SER) and application schema information (from the OER) are entered into the system using Rdb "STORE" statements.

The mapping algorithms for this process have also been automated. After the creation of an SER model and corresponding OER model for a given application system, a set of "STORE" statements are generated to input these classes of metadata into the corresponding GIRD relations. Table 3 shows a sample of the "STORE" statements that enters the metadata of the shop floor controller (from the modeling example in Section 2.2) into the metadatabase.

```
INVOKE DATABASE FILENAME  'GIRD'
START_TRANSACTION READ_WRITE

STORE X IN ITEM USING
      X.ITEMCODE = "I120";
      X.ITEMNAME = "PART_ID";
      X.DESCRIPT = "PART IDENTIFIER"
      X.FORMAT = "CHARACTER";
      X.LENGTH = 20;
      X.ADDEDBY = "LR";
      X.DATEADDED = "3/27/1989";
      X.APPLNAME = "MRP_II";
END_STORE

STORE X IN EQUIVALENT USING
  X.ITEMCODE = "I120";
  X.ITEMCODE = "I230";
  X.RNAME= C_I_CONVERT;
  X.ADDEDBY = "MB";
  X.DATEADDED = "4/09/1990";
END_STORE

STORE S IN SUBJECT USING
  X.SNAME = "JOBQUEUE";
  X.DESCRIPT = "QUEUE FOR ALL
                WORKSTATIONS";
  X.XCOORD = 40;
  X.YCOORD = 10;
  X.ADDEDBY = "LR";
  X.DATEADDED = "3/27/1989";
  X.SUPERSUBJECT = "CELLCONT";
  X.APPLNAME = "SHOP FLOOR";
 END_STORE
```

SAMPLE RDB DML STATEMENTS TO POPULATE THE METADATABASE
TABLE 3

4.4 Metadatabase Management System and Status of the Prototype

The metadatabase management system (MDBMS) shell has been developed on top of the platform to provide the required capabilities that go beyond VAX/Rdb features and to serve as a user interface. The MDBMS is menu driven with a high-level language to be added in the near future.

Table 4 below shows the MDBMS main menu that summarizes its major classes of use. Function 1, “Manage the Metadatabase”, provides the four basic management operations (insert, delete, update, and retrieve) for metadata and helps users to formulate queries that require navigation through the GIRD model. Function 2 offers a set of predetermined classes of queries that are frequently needed in the target application environment. Other, general queries

can also be formulated in a syntax-free manner with assistance from the metadatabase. Function 3 provides a gateway to a metadata-assisted global query system for formulating and processing queries on the raw data in local systems. Function 4 provides a gateway to the information base modeling system (IBMS) that allows users to model new applications or contemplated changes to existing application systems. Finally, Function 5 is invoked to manage the knowledge-based integrator; i.e., the modules which transform the system into an active metadatabase.

MDBMS MAIN MENU
1. Manage the Metadatabase
2. Query the Metadatabase
3. Access Application (Local) Systems
4. Model Application (Local) Systems
5. Manage the System Integrator
6. Exit MDBMS

TABLE 4

Among these functions, number 1 is currently limited to what can be done through Rdb and number 5 is the least complete. The remaining three, however, are all in place although improvements and enhancements are continually underway. The functionalities of the current implemented system will be illustrated with some detailed examples in the next section.

5. THE FUNCTIONALITY OF THE GIRD MODEL: SOME EXAMPLES

The metadatabase prototype using the GIRD model has been tested in a Computer-Integrated Manufacturing (CIM) environment to illustrate the metadatabase approach to information integration. The CIM environment is described in Section 2.2. In this section, some representative examples are illustrated and a high-level algorithm is given for each example to show how the GIRD model works. There are four classes of queries considered central to information management using a passive metadatabase:

1. **Global dictionary/directory**
2. **Synthesized views/subjects**
3. **Event-triggered queries**

4. "What-if" simulations

To answer most query types in the scenario only requires information from a stand-alone metadatabase; i.e., the metadatabase does not need to be connected to the local systems in order to satisfy the query. The remaining queries assume a semi-active mode whereby the metadatabase interacts with local database management systems (DBMS) to (1) query and (2) receive results from them.

Scenario: Suppose that enterprise users are concerned with a project to evaluate the manufacturability of a given part design.

5.1 Global Data Dictionary/Directory Queries

One type of directory query seeks information concerning the modeling and implementation hierarchy. That is, for a given application/concept (e.g., "Workstation Controller") the user is interested in finding the functional model, structural model, local implementation model, and the raw data. To answer such a query using the GIRD structure, results are given from the SUBJECT (hierarchy) meta-entity, the ENT-REL and INTEGRITY meta-entities, the SOFTWARE RESOURCES meta-entity, and the raw data, respectively. An example of such a query is given in Example 1a below.

Another and more classic example of this query type is, "Where do we have information on <parts>?" Unlike queries to traditional DBMSs or dictionary systems which retrieve information from one database at a time, this type of query retrieves global dictionary/directory information (names, formats, and other pertinent metadata) from the many different DBMSs at once. Moreover, this approach not only returns dictionary/directory information for the variable named, but also for all its variants in different systems (Example 1b).

This capability requires a global equivalence knowledge for all data items pertaining to the same logical objects. Such knowledge is beyond the scope of traditional dictionary systems, but is maintained with the metadatabase. Note that the GIRD structure also supports the search for relations with composite identifiers even when only a piece of the identifier is used in the query. Therefore, using the metadatabase approach, a single query yields enterprise-wide results.

Note that there are model hierarchies corresponding to MRP II System, and Process Planning System, and WIP Tracking System, and Product Design System. These hierarchies converge from separate conceptual models into a

single logical model (see Figure 1) and then diverge to separate physical models and raw data.

Finally, once it is known where basic types of information are stored, it may also be desirable to know what kind of information (attributes) about parts is available in the systems identified. This type of query builds on results of the initial query and returns field names if the source of information is a DBMS or returns a description if the source of information is not under the control of a DBMS (i.e., under the control of a file manager).

Note that this type of query can be used to find detailed metadata on a class of data structures (i.e., part) or, in a semi-active mode, to return detailed metadata on a particular instance of a class (i.e., "part#xyz") in any local system. Example 1c below illustrates the first case. The logic needed to produce these results from the GIRD model (in conjunction with local systems) is illustrated in Part A of Table 5. Queries shown in A1, A2, and A3 of that table correspond to example 1a below and those in A4 and A5 correspond to examples 1b and 1c, respectively).

5.2 Synthesized View/Subjects

This type of query allows users to define new and persistent views from the existing data and rules stored in the metadatabase. Clearly, the new view must be defined before any queries using that view may be processed. Synthesized views are defined in terms of existing data items (bottom-up design) then stored for reuse. This class of use clearly indicates an integral role for information modeling capabilities in metadata management.

Consider the manufacturability example. Since manufacturability is a widely used concept which spans many implementation systems, and does not have a consensus definition, it is desirable to capture and store the organization's view of the term "manufacturability." This view is, by definition, a SUBJECT with data items, intra-subject rules, functional dependencies characteristic of all SUBJECTs.

This new view is then stored, and available for satisfying future queries, including queries as to the definition of "manufacturability" itself. It is transparent to subsequent queries that the SUBJECT "manufacturability" was created in this manner.

This type of functionality, whereby new views may be created by combining elements from multiple and perhaps, heterogeneous systems, is unique to the metadatabase approach. The structure of GIRD enables the same integrity

constraints to be enforced for these synthesized SUBJECTs. (Similar synthesis may be used to create new CONTEXTs.) Example 2 shows the creation of a synthesized SUBJECT including assignment of data items and intra-subject rules, and uses the logic presented in Part B of Table 5.

Example 1a:

KEY: < x > Instance of variables.

Find the modeling/implementation hierarchy for <Workstation Controller>:

Functional Model: There are 3 levels in the SUBJECT hierarchy for <Workstation Controller>.

SUBJECT level 0: SHOP FLOOR CONTROL SYSTEM
sub-subject level1: WORKSTATION CONTROLLER
sub-subject level2: MOVER
sub-subject level2: OPERATOR

Structural Model: There are <9> ENT-REL structures representing <Workstation Controller>.
There are <3> INTEGRITY structures representing <Workstation Controller>.

ENT-REL:
Entity (OE) Part, Workorder, Sequence, ProcessPlans, Workstation
Relationship (PR) Composed-of, P-S-W, From-To, Part-From-To
INTEGRITY:
Existence dependency (MR) Has, Made-by
Referential (FR) Identified-by

Local Implementation Model: There are <14> files storing data used in <Workstation Controller>.

***Entity* (OE): Part**

File name	*Type*	*Sizeunit*	*Sizevalue*	*Location*	*System*
part.db	Rdb data	bytes	1000	128.113.4.121	SFC
part.dbf	dBase3	records	500	128.113.7.156	Process Plan
partmaster.dat	Btrieve	pages	10	128.113.7.221	MRP II

Example 1b:

KEY: **{ n }** Calculated results. Required algorithms are defined and stored in the metadatabase.

<part> information is maintained in {3} databases/ systems

<Part> is an identifier for the information stored in the following locations

#	Relation	Type	Identifier	System	Location (node id)
1.	part	3D model	part#	CATIA	128.117.5.153
2.	part	NC code file	part#	CATIA	128.117.5.153
3.	part	Textfile	part#	CATIA	128.117.5.153
4.	BOM	Textfile	part#	CATIA	128.117.5.153
5.	Partmaster	Btrieve (db) file	PID	MRPII	128.113.4.127
6.	BOM	Btrieve (db) file	PID	MRPII	128.113.4.127
7.	part	Rdb (db) file	partid	SFC	128.113.4.121
8.	product	Rdb (db) file	partid	SFC	128.113.4.121

Example 1c

What type of information is available on <part>?

#	System	Description or fields
1.	CATIA	3D wireframe model, require 8514A terminal to display
2.	CATIA	functional specifications, mat'l specification, nominal dimensions/tolerances, BOM, NC code,authorization-status, date, ...
3.	Proc. Plan	part classification, GT code, operation and detail descriptions, routings, BOM, workstation/machine, tooling, fixtures, quality control/inspection details, ...
4.	MRPII	costs, leadtime, routing, inventory, selling price, source, scheduling code, order policy, cycle count class, ABC code, planned orders, firm planned orders, open work orders/ purchase orders, scrap, defects, ...
5.	SFC	location, WIP quantity, defects, scraps, open WOs, ...

Example 2:

Define new <SUBJECT>. New SUBJECT name? <manufacturability>

Attributes of <SUBJECT> <manufacturability> include:

Created by: LR
Date created: 01/16/91

#	Relation	Attribute	System
1.	part	x-dimension	CATIA
2.	part	y-dimension	CATIA
3.	part	z-dimension	CATIA
4.	part	tolerances	CATIA
5.	part	weight	CATIA
6.	part	prod. volume	sales/marketing
7.	workstation	size limits	Proc. Plan
8.	workstation	tolerances	Proc. Plan
9.	workstation	payload	Proc. Plan
10.	workstation	capacity	MRPII
11.	workstation	avail. capacity	MRPII
12.	workstation	reliability	SFC/MRPII

Intra-subject rules in the <SUBJECT> <manufacturability> include:

Created by: LR
Date created: 01/16/91

```
1.  Rule111.condition   IF Part(x).GTcode = Part(x').GTcode
            .action       THEN  Like(Part(x'), Part(x))

2.  Rule112.condition   IF Part(x).status EQUALS <production>
            .action       THEN
                            DEFINE  Part(x).<Monthly defect rate> :=
                            Part(x).#defects_per_month /
                            Part(x).total production/month

3.  Rule113.condition   IF PART(x).status EQUALS <pre-production>
            .action       THEN
                            SELECT any PART(x')
                            WHERE Like(Part(x'), Part(x))
                          AND
                            DEFINE  Part(x).<Monthly defect rate> :=
                            Part(x').<Monthly defect rate>
```

5.3 Event-triggered Queries

This type of query results in (1) some manipulation of (raw) data that has been retrieved from (multiple) systems, and/or (2) the triggering of some rules or procedures based on the occurrence of an event (or set of events). These queries require that raw data be accessible by the MDB in order to be manipulated; i.e., it requires a semi-active metadatabase.

Continuing on the manufacturability example to illustrate the first type of query, we might want to know the defect and rework rates for a given part. If the part is already in production, then the part's metadata concerning rework and defects would be used to answer the query. If, however, the part is still in a pre-production stage, then the query would be satisfied by metadata from similar parts (Example 3). (Part similarity would be determined by, for example, group technology (GT) codes stored in a process planning system.) As shown in Example 2, these rules are stored in the metadatabase. This type of query requires that not only current status, but also historical information be maintained. Its processing needs full capabilities of an inference engine to fire the rules and trigger the desired events requires a complex set of functions.

Example 3

```
IF <defect rate> GREATER THAN <2%> on <PART>
AND <defect rate> LESS THAN <7%> on <PART>

THEN SEND <warning#176> TO <process plan manager>
AND SEND <warning#1719> TO <QC manager>
```

5.4 "What if" Simulation

The metadatabase structure can help a user determine the global impact of proposed changes to local databases before making any changes. That is, by storing metadata globally, elemental relationships among sub-systems can be captured and studied, in a largely unobtrusive manner.

For example, what if the manufacturing managers wants to redesign an outdated shop floor control database? It would be vitally important to know just how each of the attributes of the current system is related to other systems -- such that the impact of changes will be understood before they are undertaken. Discovering where the elements are used throughout the enterprise facilitates systematic analysis and design.

Example 4 demonstrates this process when a user is considering deleting an attribute from a relation in the shop floor control system. The response indicates that the attribute is a significant participant in constructs that affect other systems, and the algorithm in Part C of Table 5 indicates the logic of traversing the GIRD model to assess the impact of this change.

Example 4:

What if ATTRIBUTE <SFC.WO_status> is DELETED?

ATTRIBUTE <SFC.WO_status> is used:	
Type	**Description**
Rule.condition	**If SFC.WO_status = complete ...**
.action	**Then UPDATE MRPII.workorderstatus**
.action	**And UPDATE MRPII.multilocation**
Rule.condition	**If SFC.WO_status = scrap ...**
.action	**Then Do-Rework-Rule**
.action	**And UPDATE MRPII.workorderstatus**
.action	**And UPDATE MRPII.multilocation**
EQUIVALENCE	**SFC.WO_status Equivalent MRPII.workorderstatus**
FDs	**SFC.WO#, SFC.WO_status --> SFC.nextWO#**

A. Retrieval of Global metadata

A1. Functional Model

```
SELECT   SSNAME FROM SUBJECT
  WHERE  SNAME = 'WORKSTATION'
SELECT   SNAME FROM SUBJECT
  WHERE  SSNAME = 'WORKSTATION'
SELECT   CNAME FROM CONTEXT, RELATES
  WHERE  RELATES.SNAME = subjname
 (applied recursively to all subjects found above)
```

A2. Structural Model

```
SELECT   ERNAME, ERTYPE
  FROM   MAPPEDTO, ENTREL
  WHERE  MAPPEDTO.SNAME = 'WKSTATION'
  AND    MAPPEDTO.ERNAME = ENTREL.ERNAME
SELECT   INTNAME, INTTYPE
  FROM   MAPPEDTO, INTEGRITY
  WHERE  MAPPEDTO.SNAME = 'WORKSTATION'
  AND    ( ERNAME = INTEGRITY.MASTER
           OR ERNAME = INTEGRITY.SLAVE )
```

A3. Implementation Model

```
SELECT   RESNAME, RESTYPE, SIZEUNIT,
  SIZEVALUE, LOCATION, APPLNAME
  FROM   SOFTWARE, USES, HARDWARE,
         STOREDIN, BELONGTO
  WHERE  BELONGTO.ERNAME = 'PART'
  AND    BELONGTO.INPKEY = 1
  AND    BELONGTO.ITEMCODE =
                  STOREDIN.ITEMCODE
  AND    STOREDIN.RESID = SOFTWARE.RESID
  AND    USES.RESID = SOFTWARE.RESID
  AND    HARDWARE.SERIALNO =
                  RESIDESAT.SERIALNO
```

A4. where is information about 'part' maintained

```
SELECT   RESNAME, RESTYPE, ITEMNAME,
         APPLNAME, LOCATION
  FROM   SOFTWARE, USES, HARDWARE,
         STOREDIN, BELONGTO
  WHERE  BELONGTO.ERNAME = 'PART'
  AND    BELONGTO.INPKEY = 1
  AND    BELONGTO.ITEMCODE =
                  STOREDIN.ITEMCODE
  AND    BELONGTO.ITEMCODE =
                  ITEM.ITEMCODE
  AND    STOREDIN.RESID = SOFTWARE.RESID
  AND    USES.RESID = SOFTWARE.RESID
  AND    HARDWARE.SERIALNO =
            RESIDESAT.SERIALNO
```

A5. Information available on part

```
SELECT   SUBJECT.APPLNAME,
         ENTREL.DESCRIPT, ITEMNAME
  FROM   SUBJECT, DESCRIBES, ENTREL,
         MAPPEDTO, BELONGTO, ITEM
WHERE    ENTREL.ERNAME = 'PART'
  AND    BELONGTO.ERNAME = PART'
  AND    BELONGTO.ITEMCODE =
                  ITEM.ITEMCODE
  AND    DESCRIBES.ERNAME = 'PART'
  AND    DESCRIBES.SNAME =
                  SUBJECT.SNAME
```

B. View Synthesis/Modeling

- Find pertinent information (similar to A1-A5)
- Invoke the modeling system (optional)
- Create and Insert new views using:

```
INSERT   INTO SUBJECT (SNAME, DESCRIPT,
           ADDEDBY, DATEADDED)
  VALUES  ('MANUFACT', '.....', 'LR005', ' ')
INSERT   INTO DESCRIBES(SNAME, ITEMCODE)
  VALUES ('MANUFACT', 'ITEM-100')
INSERT   INTO APPLIES(SNAME, RNAME)
  VALUES ('MANUFACT', 'RULE-112')
```

C. For Event/Trigger queries Find:

- Factid for item from ITEM, FOR, and FACT
- Conditions on Factid from LTOPERAND, RTOPERAND, and CONDITION
- All Rules to be fired from CONDOF and RULE
- Actions from ACTOF,ACTION,CALLS
- Invoke Rule Processor to execute the actions

D. Simulation queries

- Find the itemcode from ITEM
- Find the equivalent items

```
SELECT  ITEMNAME , APPLNAME
FROM    ITEM, EQUIVALENT WHERE
     EQUIVALENT.ITEMCODE  = ITEM.ITEMCODE
AND     EQUIVALENT.ITEMCODE  ~= Itcode
AND     STOREDIN.EQITEMCODE =
(SELECT EQITEMCODE  FROM EQUIVALENT
          WHERE ITEMCODE  = Itcode
                OR EQITEMCODE  = Itcode )
```

- For these Items Find:
  - SUBJECTs from DESCRIBES
  - Entities/relationships from BELONGTO
  - Files from SOFTWARE and STOREDIN
  - Rules from ITEM, APPLIES, and RULE

ALGORITHMS FOR THE EXAMPLES USING GIRD MODEL
TABLE 5

6. COMPARISON OF GIRD WITH IRDS STANDARD

The GIRD and IRDS models differ at a fundamental level; i.e., the scope and use of metadata as discussed in Section 1. In sum, GIRD (1) aims to facilitate information integration for functional synergy of the enterprise as opposed to managing software resources per se, (2) accommodates heterogeneous systems, and (3) also includes knowledge.

Some technical points are discussed herein. The Global Information Resources Dictionary model extends the concept of metadata to encompass subjects and contexts including events and the rules that trigger such events in both the enterprise and the metadatabase itself. IRDS, in comparison, represents a static view of the enterprise through the Entity-Relationship model due to P. Chen [4]; i.e., its constructs do not capture the dynamic interrelationships between systems in the enterprise. Consider the shop floor control example in Section 2.2. In this example, inter-subject operating rules govern the interactions between the mover and operator workstations. IRDS can represent the data structures of the mover and operator, but does not represent this inter-subject knowledge which dictates how, when, and why the information is passed between these systems.

An analytical benefit of the metadatabase using GIRD model is that as a *system*, the metadatabase is modeled and the model is stored within itself. The modeling of the metadatabase produced SER and OER models which were stored as instances of a system within the implemented metadatabase system (see Section 4). In fact, as depicted in Figure 5, the GIRD model entails five layers in its logic — layers that are independent of industry or application functions. Each layer is definitively structured for its own contents and linked with other layers through mapping algorithms that are derived directly from the modeling constructs. Controls are also provided by virtue of the types and definitions of these TSER constructs. This is a class of strong self-descriptiveness unique to the GIRD model. As a result, the self-description permits comprehensive yet concise control of metadata and also allows a user to query about the underlying GIRD model itself. In contrast, the four layer structure of IRDS (see, e.g., [8]) anchors less in its underlying model or logic than in applications. Therefore, it does not allow for stable and generic self-description [7].

While IRDS extends the traditional concept of DDS, practitioners and researchers in this field have highlighted several issues concerning IRDS as a

standard [2], some of which the present GIRD work addresses. Specifically, IRDS was considered as lacking the following: (1) the ability to represent hierarchical structures of applications or semantics, such as object-oriented models; (2) representation of relationships of degree greater than binary; (3) metadata representing complex structures, definitional knowledge, and process models; and (4) thesaurus capability.

The TSER methodology (functional modeling, structural modeling, and metadata modeling) provides the GIRD model the necessary structures to fully represent modeling hierarchies. The three levels of modeling (views, structural models, and physical implementation models) are represented through the meta-entities SUBJECT and CONTEXT, ENT-REL , and SOFTWARE RESOURCES, respectively. In addition, since the GIRD model employs the TSER approach for modeling application systems (i.e., the metadata layer in Figure 5), the meta-entities SUBJECT and CONTEXT actually contain a hierarchy of semantic models for the enterprise information system.

Although the majority of associations of data in implemented systems can be considered to be binary relationships, data modeling capabilities for advanced and future systems require them to be more flexible in handling complex data. Increasing use of complex data-structures such as geometric data, however, requires n-ary relationships be supported in a global data dictionary. N-ary relationships are supported within the GIRD model by the ENT-REL meta-entity.

A unique property of the GIRD model is the inclusion of knowledge comprising contextual rules for application models, integrity constraints applied to the modeling structure, and classification knowledge over data. For complex models or ones with control information, the GIRD model meta-entities CONTEXT, RULE, CONDITION, and ACTION provide for such knowledge representation. Combining these meta-entities with SUBJECT, ENT-REL, and the capability of n-ary structures creates appropriate provisions for representing object-oriented models. For instance, objects would be represented by SUBJECTs with their transaction knowledge corresponding to intra-subject rules. CONTEXTs would be created when inter-subject rules are present. Inheritance constraints would be interpreted into OER integrity rules. These results are all supported in the above structure.

To address the need for thesaurus capabilities, especially in a multi-information systems environment, the GIRD model provides for global equivalence among data items that represent the same logical objects -- but that

happen to be implemented differently. This property, which is important for heterogeneous database systems, was developed in the GIRD model via a meta-PR called Equivalent. Instances of this meta-PR have two attributes in their composite primary key and a non-key attribute pointing to a rule for translation. The key appears as Itemcode and EqItemcode corresponding to a pair of system generated ITEM identifiers. The Rname attribute references the data translation rules, if any, for data interchange. See Appendix 1 for details.

Finally, the addition of a control shell and an inference engine on top of the GIRD gives it the capability of becoming an intelligent, active metadatabase. These added features serve to manage data and knowledge within complex enterprises with greater efficiency and effectiveness.

7. SUMMARY AND CONCLUSIONS

Metadata has always played a central role in software development and information resources management. The development of IRDS and repository systems attests to the fact that metadata management is becoming acutely important as enterprises strive to achieve functional synergies through information integration. This new role necessitates a new metadata method that encompasses both database and knowledge-based systems and models in an integrative way (see Section 1). To bring the problem into prominence, a *metadatabase* environment has been formulated to characterize the role of metadata as an on-line kernel in information integration and information resources management.

This effort investigates the metadatabase problem and develops methods for metadata representation and processing. Specifically, the Global Information Resources Dictionary model has been developed using the TSER constructs for representing heterogeneous, distributed information systems (see Sections 2 and 3). The model contributes major extensions with respect to previous efforts and addresses some aspects of the above problems (see Section 3.4; see also Sections 5 and 6 for details). The GIRD model is implemented to create a prototype metadatabase for computer integrated manufacturing at Rensselaer Polytechnic Institute (see Section 4).

We might mention that the GIRD model is generic and can be implemented in contemporary environments (say, a relational system) with moderate system development effort; although it has been developed using TSER and is currently implemented for a manufacturing environment. These concerns for the generality of the model are addressed respectively in Sections 3 and 4.

There are, however, some other issues that have not been clearly resolved. One complex issue concerns transactions modeling. To what extent this class of metadata should be considered as part of the enterprise information resources and can be incorporated into the metadatabase is not easy to answer. The inclusion of transactions at a significantly detailed level would increase the size of the rulebase considerably. As a result, new languages might be needed to optimize the processing of these metadata and their execution (firing) in applications. Another issue that requires empirical studies to resolve is whether the metadatabase should be distributed. Current investigation on the prototype indicates the feasibility of a distributed metadatabase system. At least, we believe that multiple copies of the metadatabase can be created at different sites without requiring as complex control as the conventional distributed databases do. The main reason for this simplified control is that metadata tend to be more stable over relatively long periods of time.

Further research and development of the metadatabase system is ongoing. In the initial implementation of the GIRD model, data and knowledge components are represented with the same set of constructs, but are processed separately (see Section 4). A major objective for ongoing enhancements and new development is to integrate these two parts of processing. Based on the GIRD work, an active metadatabase that interacts directly with local systems for global integration is being developed. These endeavors indicate opportunities for developing new metadata languages that are based on the GIRD model and combine data processing with rules processing.

Additional future work includes the migration of the metadatabase model to a more generic environment for non-manufacturing enterprises. Moreover, since IRDS is now the government standard, mapping algorithms will be developed to derive IRDS schemata from corresponding GIRD schemata. It is expected that either some classes of information will be excluded in the translation, or that new IRDS constructs and modules will be created via IRDS extensibility feature to handle the extended metadata available from the GIRD model.

REFERENCES

[1] F. Allen, M. Loomis, and M. Mannino, "The Integrated Dictionary/ Directory System," *Computing Surveys*, Vol. 14, No. 2, June 1982, pp. 245-286.

[2] S. Anderson, "The Information Resource Dictionary System (IRDS) Standard Proposal," *Database Newsletter*, Vol. 16, No. 5, Sept/Oct 1988.

[3] American National Standards Institute. (Draft proposed) American National Standard Information Resource Dictionary System: Part 1-Core Standard. ANSI X3H4, American National Standards Institute, New York. 1985.

[4] P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions Database Systems*, Vol. 1, No. 1, March 1976, pp. 9-36.

[5] E. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, Vol. 13, No. 6, 1970, pp. 377-387.

[6] C. Date, *An Introduction to Database Systems*, Vol.1, 4th Edition, Addison Wesley, 1986.

[7] D. Dolk and R. Kirsch, II. "A Relational Information Resource Dictionary System," *Communications of the ACM*, Vol. 30, No. 1, January 1987, pp. 48-61.

[8] A. Goldfine, "The Information Resource Dictionary System," *Proceedings of the 4th International Entity-Relationship Conference*, IEEE Press, New York, 1985, pp. 114-122.

[9] _____ and P. Konig, *A Technical Overview of the Information Resource Dictionary System*, NBSIR 85-3164, National Bureau of Standards, Gaithersburg, MD, April 1985.

[10] _____ and _____ , *A Technical Overview of the Information Resource Dictionary System (Second Edition)*, NBS Special Publication NBSIR 88-3700, National Bureau of Standards, Gaithersburg, MD, January 1988.

[11] C. Hsu, "Structured Database Systems Analysis and Design Through Entity-Relationship Approach," *Proceedings 4th International Conference on Entity-Relationship Approach*, IEEE Computer Society, 1985, pp. 56-63.

[12] _____ and C. Skevington, "Integration of Data and Knowledge in Manufacturing Enterprises: A Conceptual Framework," *Manufacturing Systems*, Vol. 6, No. 4, 1987 pp. 277-285.

[13] _____ , A. Perry, M. Bouziane and W. Cheung, "TSER: A Data Modeling System Using the Two-Stage Entity-Relationship Approach," *Proceedings of the 6th Entity Relationship Approach Conference*, New York, November 1987, pp. 461-478.

[14] _____ and A. Perry, "TSER-Based Knowledge Model for Shop Floor Control and MRP II," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1988, pp. 330-335.

[15] _____ and L. Rattner, "Information Modeling for Computerized Manufacturing," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 4, 1990.

[16] _____ , M. Bouziane, W. Cheung, J. Nogues, L. Rattner, L. Yee, "A Metadata System for Information Modeling and Integration," *Proceedings of the International Conference Systems Integration*, IEEE Computer Society, April, 1990.

[17] _____ , M. Bouziane, L. Rattner, L. Yee, "A Metadatabase Architecture for Heterogeneous, Distributed Environments: Global Information Resources Dictionary (GIRD), *Proceedings of 2nd International Conference on Computer Integrated Manufacturing*, IEEE Computer Society, May, 1990.

[18] R. Jansen and P. Compton, "The Knowledge Dictionary: Storing Different Knowledge Representations," *Proceedings of the 3rd European Knowledge Acquisition Workshop*, Paris, July 1989, pp. 448-461.

[19] W. Kim, "Object-Oriented Databases: Definition and Research Directions," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 3, 1990, pp. 327-341.

[20] _____ , (ed.), *Data Engineering*, IEEE Computer Society, Vol. 13, No. 2., (Special Issue on Database Connectivity), 1990.

[21] T. Korson and J. McGregor, "Understanding Object-Oriented: A Unifying Paradigm," *Communications of the ACM*, Vol. 33, No. 9, 1990, pp. 40-60.

[22] L. Mark and N. Roussopolous, "Metadata Management," *IEEE Computer*, December 1986, pp. 26-36.

[23] S. Navathe and L. Kerschberg, "Role of Data Dictionaries in Information Resource Management," *Information and Management*, Vol 10, 1986, pp. 21-46.

[24] _____ , R. Elmasri, and J. Larson, "Integrating User Views in Database Design," *Computer*, IEEE Computer Society, Vol. 19, No. 1, 1986, pp. 50-62.

[25] R. Wang and S. Madnick, "Facilitating Connectivity in Composite Information Systems," *ACM Data Base*, Vol. 20, No. 3, Fall 1989, pp. 38-46.

Appendix 1: GIRD Constructs

Meta-Entities

KEY: CONSTRUCT NAME(Primary_key, attribute[1],...attribute[n])

1. **ACTION (Actid, acttype, factid, addedby, dateadded, modifby, lastmod, nummods)**
2. **APPLICATION (Applname, descript, addedby, dateadded, modifby, lastmod, nummods, userid)**
3. **CONDITION (Condid, leftfact, operator, rightfact, addedby, dateadded, modifby, lastmod, nummods)**
4. **CONTEXT (Cname, applname, descript, xcoord, ycoord, addedby, dateadded, modifby, lastmod, nummods)**
5. **ENT-REL (ERname, ertype, descript, akey, addedby, dateadded, modifby, lastmod, nummods)**
6. **FACT(Factid, factname, facttype, factvalue)**
7. **HARDWARE RESOURCE (Serialno, hname, htype, descript, location, nodename, nodeaddr, manufacturer, purchby, datepurch, addedby, dateadded, modifby, lastmod, nummods, userid)**
8. **INTEGRITY (Intname, inttype, descript, master, slave, addedby, dateadded, modifby, lastmod, nummods)**
9. **ITEM (Itemcode, itemname, itemtype, descript, format, length, domain, defvalue, addedby, dateadded, modifby, lastmod, nummods, applname)**
10. **PROGRAM(Functid, ...)** ***Note: This meta-entity is consolidated in SOFTWARE RESOURCES.***
11. **RULE (Rname, rtype, descript, condid, addedby, dateadded, modifby, lastmod, nummods)**
12. **SOFTWARE RESOURCE (Resid, resname, extension, restype, descript, sizevalue, sizeunit, coding, developedby, addedby, dateadded, modifby, lastmod, nummods)**
13. **SUBJECT (Sname, descript, xcoord, ycoord, addedby, dateadded, modifby, lastmod, nummods, supersname, applname, fileid)**
14. **USER (Userid, username, class, position, phone, office, address, addedby, dateadded, modifby, lastmod, nummods)**

Meta-Plural Relationships (PRs)

1. **ActOf (Actid, Rname, relorder, addedby, dateadded, modifby, lastmod, nummods)**
2. **Applies (Sname, Rname, relorder)**
3. **ApplUser (Applname, Userid, password, accesscode, addedby, dateadded, modifby, lastmod, nummods)**
4. **BelongTo (Itemcode,ERname, relpos, inpkey, posinpk)**
5. **Calls (Actid, Procid, Parid, parorder)**
6. **Computes (Factid, Functid,Parid, parorder)**
7. **Contains (Cname, Rname, relorder, addedby, dateadded, modifby, lastmod, nummods)**
8. **Describes (Itemcode, Sname, relpos)**
9. **Equivalent (Itemcode, EqItemcode, rname, addedby, dateadded)**
10. **MappedTo (Sname, ERname, addedby, dateadded, modifby, lastmod, nummods)**
11. **ModuleOf (Resid, Subresid, relationship)**
12. **NamedAs (ERname,Applname, localname)**
13. **Relates (Cname, Sname, direction)**
14. **ResidesAt (Resid, Serialno, path, invokecom, addedby, dateadded, modifby, lastmod, nummods)**
15. **StoredIn (Itemcode, Resid, relpos)**
16. **Uses (Applname, Resid, dataorg, addedby, dateadded, modifby, lastmod, nummods)**

Meta-Mandatory Relationships (MRs)

1. **Component (APPLICATION, SUBJECT)**
   Role APPLICATION = owner
   Role SUBJECT = owned
2. **ComposedOf (SUBJECT1, SUBJECT2)**
   Role SUBJECT1 = owner (i.e; the superclass)
   Role SUBJECT2 = owned (i.e; the subclass)
3. **Express (CONDITION, FACT)**
   Role CONDITION = owner
   Role FACT = owned
4. **InterComponent (APPLICATION, CONTEXT)**
   Role APPLICATION = owner
   Role CONTEXT = owned
5. **LtOperand (FACT, CONDITION)**
   Role FACT = owner
   Role CONDITION = owned
6. **RtOperand (FACT, CONDITION)**
   Role FACT = owner
   Role CONDITION = owned

Meta-Functional Relationships (FRs)

1. **ApplMaintain (APPLICATION, USER)**
   Role APPLICATION = determinant
   Role USER = determined
2. **BindFact (ACTION, FACT)**
   Role ACTION = determinant
   Role FACT = determined
3. **CondOf (RULE, CONDITION)**
   Role RULE = determinant
   Role CONDITION = determined
4. **Convert (Equivalent, RULE)**
   Role Equivalent = determinant
   Role RULE = determined
5. **ERExist (INTEGRITY, ENT-REL)**
   Role INTEGRITY = determinant
   Role ENT-REL = determined
6. **For (FACT, ITEM)**
   Role FACT = determinant
   Role ITEM = determined
7. **HardMaintain (HARDWARE RESOURCE, USER)**
   Role HARDWARE RESOURCE = determinant
   Role USER = determined
8. **ItemIn (ITEM, APPLICATION)**
   Role ITEM = determinant
   Role APPLICATION = determined
9. **SubjectIn (SUBJECT, SOFTWARE RESOURCE)**
   Role SUBJECT = determinant
   Role SOFTWARE RESOURCE = determined

Appendix 2
Definition of Meta-attributes

KEY (for meta-attribute name):

attribute - (lower case) non-key field in meta-relations
Attribute - (upper/lower case) key field in meta-relations
Attribute - (underlined upper/lower case) both key and non-key field in meta-relations

Meta-attribute Name	Description
accesscode	An attribute of the meta-PR ApplUser that identifies a user's authorized data access level; e.g., Read (R), Write (W), Execute (E), Delete (D).
Actid	Unique identifier (primary key) for meta-entity ACTION.
acttype	Class of consequences of the production rule. Ex. Takes on a value of 0 if result of rule is binding of a fact or a value of 1 for a procedure call.
addedby	Name/initials of a modeler or information administrator who entered the meta-entity or relationship into the GIRD. Provides for an audit trail.
address	Home address of a user. Attribute of meta-entity USER.
akey	Alternative primary-key(s) for an ENT-REL base relation.
Applname	Unique name (primary key) for an application.
class	Classification scheme for end-users; can serve to control priveledges and data access.
Cname	Unique name (primary key) for the meta-entity CONTEXT.
coding	The type of physical representation of a software sesource; e.g., Pascal or LISP for program code; or ASCII, VSAM, or ISAM for data files.
Condid	Unique identifier (primary key) for meta-entity CONDITION. Also an attribute of meta-entity RULE.
dataorg	Indicates how the data is organized in an application in meta-PR USES.
dateadded	Date that instance of meta-entity or meta-relationship was added to GIRD.
datepurch	Date on which a hardware resource was purchased/acquired.
defvalue	Default value, if any, for a meta-entity ITEM.
descript	Description of all defined meta-entities and meta-relationships.
developedby	The name of the firm or person who developed a software resource.
direction	Indiates how the link (data flows) between a CONTEXT and SUBJECT is directed graphically. (i.e.; 1 = toward SUBJECT; 2 = toward CONTEXT; 3 = bidirectional; nil = none) Attribute of meta-PR Relates.

Meta-attribute Name	Description
domain	The set of values that can be assigned to a data item (meta-entity ITEM).
EqItemcode	Synonym for Itemcode in meta-PR Equivalent.
ERname	Unique name (primary key) for meta-entity ENT-REL.
ertype	The type of ENT-REL; takes on a value of "OE" or "PR" corresponding to operational entity and plural relationship respectively.
extension	The file-name extension (if any) for a software resource.
Factid	Unique system-generated identifier (primary key) for meta-entity FACT. Also, an attribute of meta-entity ACTION.
Factname	Attribute of a fact that is either an Itemcode or an expression (Condid).
facttype	Attribute of a fact that indicates how the value of the fact is to be assigned: 0 if the fact value is to be retrieved from a local database, 1 if it is the result of an expression evaluation, and 2 if it is computed by a function call.
factvalue	The calculated or referenced value, or a constant, that binds a fact during the rule inferencing process.
fileid	Attribute of meta-entity SUBJECT. Synonym for Resid.
format	The data item representation type. Attribute of meta-entity ITEM. Examples: Character (C), Integer (I), Real (R), BCD (B), EPCDIC (E), etc.
Functid	Synonym of Resid; identifies the function to be called for binding a fact. Key field in meta-PR Computes.
hname	Model number or name of a hardware resource.
htype	The type of hardware. Attribute of meta-entity HARDWARE RESOURCE. Examples: line-printer, mainframe, mini-, micro-computer, harddisk, etc.
inpkey	A flag (boolean value) indicating whether or not a data item is part of the primary key of ENT-REL. Attribute of meta-PR BelongTo.
Intname	Unique name (primary key) for an integrity constraint.
inttype	The type of integrity constraint, either "FR" or "MR" corresponding to functional relationship or mandatory relationship respectively.
invokecom	The command to invoke a software resource on a hardware resource. Attribute of meta-PR ResidesAt.
Itemcode	Unique system-generated identifier (primary key) for a data element (meta-entity ITEM).

Meta-attribute Name	Description
Itemname	The name of a data item in meta-entity ITEM.
itemtype	An attribute of meta-entity ITEM to indicate whether the data item is "persistent" (exists in at least in one local DB) or is generated at runtime.
lastmod	Date of last modification of GIRD meta-entities and meta-relationships.
leftfact	Synonym of Factid and represents the left operand of an expression.
length	The length of a data item. May refer to length in character positions or bytes depending upon implementation.
localname	Attribute of meta-PR NamedAs.
location	Physical location for meta-entity HARDWARE RESOURCE.
manufacturer	The manufacturer of a hardware resource.
master	An attribute of meta-entity INTEGRITY which, in the case of an FRtype, plays the role of determinant; and in the case of an MRtype, plays the role of owner.
modifby	Identifier (name or initials) of an individual who last modified an instance of a given meta-relation.
nodeaddress	Network address for a hardware resource.
nodename	Network "node" name for a hardware resource.
nummods	Number of modifications to a meta-entity. This attribute is in all meta-entities and most meta-PRs.
office	Office location or address of meta-entity USER.
operator	The logical operator in antecedant of a production rule. This includes the set of arithmetic and set operators.
Parid	Synonym of Factid which represents a parameter of a function in meta-PR Calls.
parorder	The relative position of the parameter in a function/procedure parameter list .
path	Path to top level directory in which an software resurce resides on a hardware resource.
password	The password to an application in meta-PR ApplUser.
phone	Business telephone number of a user.
posinpkey	The relative position of a data item field in the primary key of ENT-REL.

Meta-attribute Name	Description
position	Organizational position of the user; e.g., president, DBA, data-entry clerk.
Procid	Synonym of Resid, it identifies the procedure to be called for a rule action.
purchby	Identifier of individual resposible for the purchase of the hardware resource.
relationship	The relationship among software resources; in meta-PR ModuleOf.
relorder	Relative order (sequence) of a rule within a SUBJECT or CONTEXT — or of a condition in a rule.
relpos	Relative position of a data item in meta-entity ENT-REL.
Resid	A unique identifier (primary key) for meta-entity SOFTWARE RESOURCE
resname	Title/name of a software resource.
restype	Software resource type; e.g., program, data file, network, document.
rightfact	Synonym of Factid and represents the right side operand of an expression.
Rname	Unique name (primary key) for meta-entity RULE.
rtype	The type of rule; e.g., Modeling (M), Operating (O), Production (P), etc.
Serialno	The unique identifier (primary key) for meta-entity HARDWARE RESOURCE.
Sizeunit	The unit of measure for describing storage of a software resource; e.g., KBytes, blocks, cylinders, pages, etc.
Sizevalue	Quantity of units of storage for a specified software resource (expressed in sizeunits).
slave	An attribute of meta-entity INTEGRITY which, in the case of an FRtype, plays the role of determined; and in the case of an MRtype, plays the role of owned.
Sname	Unique name (primary key) of meta-entity SUBJECT.
Subresid	A synonym for Resid. .Key field in meta-PR ModuleOf.
supersname	The upper-level (if any) subject name for meta-entity SUBJECT.
Userid	Unique identifer (primary key) for meta-entity USER.
username	Full name of a user in meta-entity USER.
xcoord	X-coordinate of the graphical representation of a SUBJECT or CONTEXT.
y coord	Y-coordinate of the graphical representation of a SUBJECT or CONTEXT.