Extending and formalizing the framework for information systems architecture

by J. F. Sowa
J. A. Zachman

John Zachman introduced a framework for information systems architecture (ISA) that has been widely adopted by systems analysts and database designers. It provides a taxonomy for relating the concepts that describe the real world to the concepts that describe an information system and its implementation. The ISA framework has a simple elegance that makes it easy to remember, yet it draws attention to fundamental distinctions that are often overlooked in systems design. This paper presents the framework and its recent extensions and shows how it can be formalized in the notation of conceptual graphs.

The world contains entities, processes, locations, people, times, and purposes. Computer systems are filled with bits, bytes, numbers, and the programs that manipulate them. If the computer is to do anything useful, the concrete things in the world must be related to the abstract bits in the computer. Zachman's framework for information systems architecture (ISA) makes that link.[1] It provides a systematic taxonomy of concepts for relating things in the world to the representations in the computer. It is not a replacement for other programming tools, techniques, or methodologies. Instead, it provides a way of viewing a system from many different perspectives and showing how they are all related.[2]

Most programming tools and techniques focus on one aspect or a few related aspects of a system. The details of the aspect they select are shown in utmost clarity, but other details may be obscured or forgotten. As examples, consider each of the following techniques:

- Flowcharts, which were introduced by John von Neumann in 1945, are the oldest and still most widely used programming aid. They focus on the operations performed by a computer and their temporal sequence. They are fine for showing algorithms, but the data structures processed by the algorithms are only mentioned incidentally as they are being operated upon.
- Entity-relationship diagrams are a popular graphic notation for showing entity types, their attributes, and the relations that connect them. They are fine for showing certain kinds of constraints, but they cannot show all constraints, and they ignore the operations performed by and on the entities.
- Relational databases emphasize tables and the operations for manipulating them to derive an-

swers to complex queries. They are excellent for representing highly repetitive business data that originate in forms and tables, but they are less well-suited to the irregular data structures that occur in graphics and engineering designs.
- Object-oriented systems emphasize objects and the operations that may be performed by and on them. They are excellent for representing constraints and operations that apply to each object and its parts, but current object-oriented databases are not as good as relational systems for handling complex queries.

Each of these techniques is specialized for a different purpose. By concentrating on one aspect, each technique loses sight of the overall information system and how it relates to the enterprise and its surrounding environment. The purpose of the ISA framework is to show how everything fits together. It is a taxonomy with 30 boxes or cells organized into six columns (labeled A through E) and five rows (numbered 1 through 5). Instead of replacing other techniques, it shows how they fit in the overall scheme. Flowcharts, for example, may be suitable for describing the cell in Column B, Row 5 (the process column, component row in the framework); and entity-relationship (E-R) diagrams may be acceptable for Column A, Row 3 (the data column, system model row). But the ISA framework shows how the cells in different columns and rows relate to one another.

For any one of the 30 cells in the ISA framework, it is possible to develop a special notation that is ideally suited to the subject matter described in that cell. But to relate all of the cells to one another, there should be a common language that can describe all of them and their interrelationships. A natural language such as English is capable of describing everything in every cell. Yet English, which may be good for discussing design decisions in the early stages of system development, is not as precise and implementable as the more specialized notations. Symbolic logic is precise and general enough to describe anything that can be implemented on a digital computer and even the computer itself. But the usual predicate calculus notation for logic tends to become unreadable, even for simple examples. Conceptual graphs are a readable graphic notation for logic that is designed for translations to and from natural languages.[3,4] They can describe anything in any cell of the ISA framework, they have a formally defined mapping to and from other forms of logic, and they can coexist with the more specialized notations, including flowcharts, E-R diagrams, and object-oriented hierarchies.

Because of the generality and readability of conceptual graphs, the American National Standards Institute (ANSI) X3H4.6 Task Group is using them as a basis for the normative language of the conceptual schema for Information Resource Dictionary Systems (IRDS).[5] They invited John Sowa, the designer of the conceptual graph system, to participate in the development of the IRDS standards. They also invited John Zachman, the author of the ISA framework, to present his framework and its recent extensions. In writing this paper, the authors have applied conceptual graphs to the description of the ISA framework and its extensions.

Within IBM, the ANSI IRDS standards and Zachman's ISA framework are especially important for AD/Cycle*.[6,7] The AD/Cycle perspectives of the enterprise model, the information model, and the technology model were influenced by the three middle rows of the ISA framework. The current information model is consistent with the 1988 ANSI IRDS standards, which are based on E-R diagrams. The ANSI IRDS committee must make future standards upward compatible with the current ones, but extensions beyond E-R diagrams are needed to accommodate new developments, especially in object-oriented systems. Conceptual graphs can provide those extensions in a form that is compatible with the ISA framework.

Overview of the framework

When applied to an information system, the word *architecture* is a metaphor that compares the construction of a computer system to the construction of a house. The ISA framework is an elaboration of that metaphor. It compares the perspectives in describing an information system to the perspectives produced by an architect in designing and constructing a building:

- Scope—The first architectural sketch is a "bubble chart," which depicts in gross terms the size, shape, spatial relationships, and basic purpose of the final structure. In the ISA framework, it corresponds to an executive summary for a planner or investor who wants an estimate of the scope of the system, what it would cost, and how it would perform.

Table 1 Characteristics of framework rows

Row	Perspective	Constraint	Model
1	Planner	Financial/external	Scope
2	Owner	Usage/policy	Enterprise model
3	Designer	Structure/operation	System model
4	Builder	Technology	Technology model
5	Subcontractor	Implementation	Out-of-context models

- Enterprise or business model—Next are the architect's drawings that depict the final building from the perspective of the owner, who will have to live with it in the daily routines of business. They correspond to the enterprise (business) model, which constitutes the design of the business and shows the business entities and processes and how they interact.
- System model—The architect's plans are the translation of the drawings into detailed specifications from the designer's perspective. They correspond to the system model designed by a systems analyst who must determine the data elements and functions that represent business entities and processes.[8]
- Technology model—The contractor must redraw the architect's plans to represent the builder's perspective, which must consider the constraints of tools, technology, and materials. The builder's plans correspond to the technology model, which must adapt the information system model to the details of the programming languages, I/O devices, or other technology.
- Components—Subcontractors work from shop plans that specify the details of parts or subsections. These correspond to the detailed specifications that are given to programmers who code individual modules without being concerned with the overall context or structure of the system.

These five descriptions correspond to the five rows of the ISA framework. Figure 1 shows the original version of the framework in which three columns describe the data, function, and network at each of the five levels. Inside the 15 cells are examples of notations used to describe the corresponding perspectives on an information system.

The three columns in Figure 1 represent the data, function, and network of an information system. For each of the five rows, Column A shows what entities are involved, Column B shows the functions performed, and Column C shows the locations and interconnections. For physical processes in architecture and engineering, Column A represents the material, Column B the function, and Column C the geometry. Each row represents a different role or perspective, a different set of constraints, and therefore different model structures. Table 1 shows the perspectives, constraints, and models for the first three columns that were described in Zachman's original framework.[1]

The constraints are additive; that is, the constraints of a lower row are added to the model of a higher row to produce a new model at a new perspective. Ideally, the new model should not be so dissimilar that the higher-row model cannot be inferred (or reverse engineered) from the new lower-row model. This is the challenge of quality management: to ensure that as the model transformations take place (that is, as each model is structurally changed through the successive application of additional constraints) the original purpose of the business is not so obscured that the business requirements are not recognizable in the end product.

In practice, a constraint in a lower row might be inconsistent with the model in the next higher row. Then the designers who are responsible for the two rows must initiate a dialog to determine what must be changed and to ensure that no gap in expectations exists between the different perspectives. No matter how desirable or aesthetically attractive a particular feature might be, if it violates the laws of physics, it cannot be implemented. Therefore, the designers must notify the owner and explicitly negotiate an alternative.

The columns of the framework represent different abstractions from or different ways to describe the real world. The reason for isolating one variable (abstraction) while suppressing all others is to contain the complexity of the design problem.

Figure 1 The original ISA framework

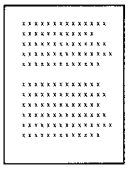
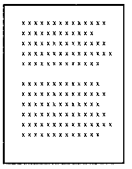
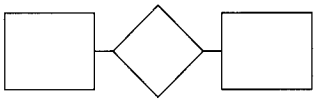
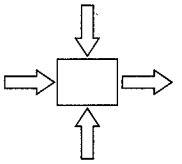
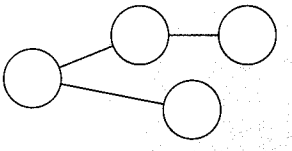
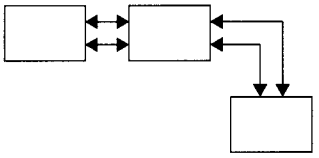
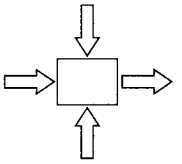
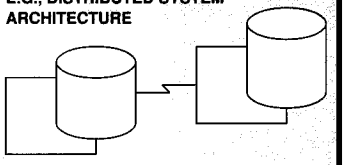
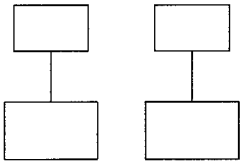
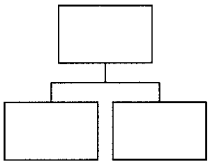
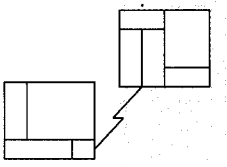
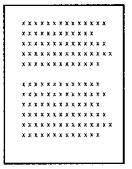
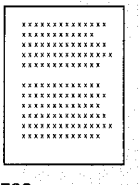
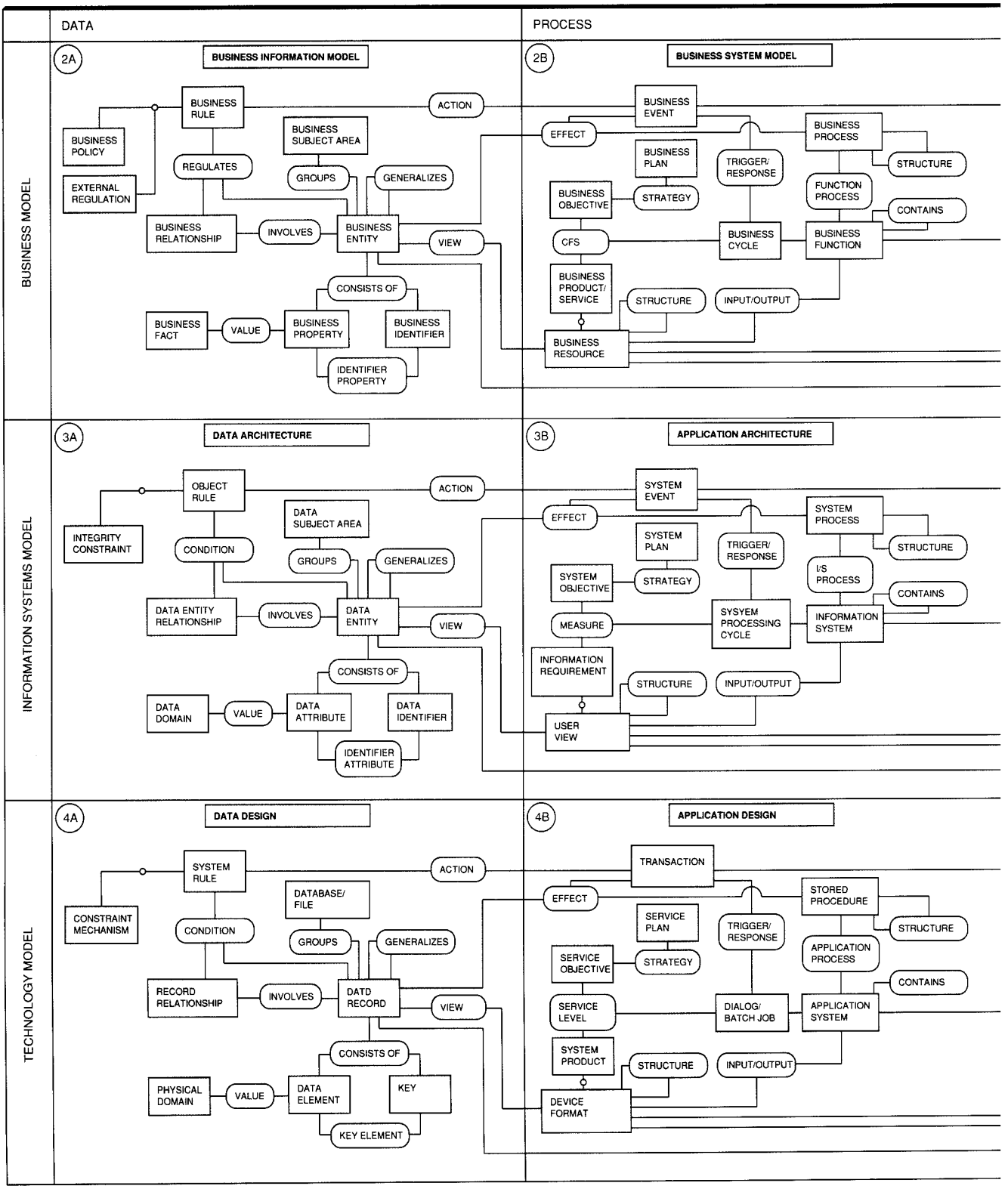
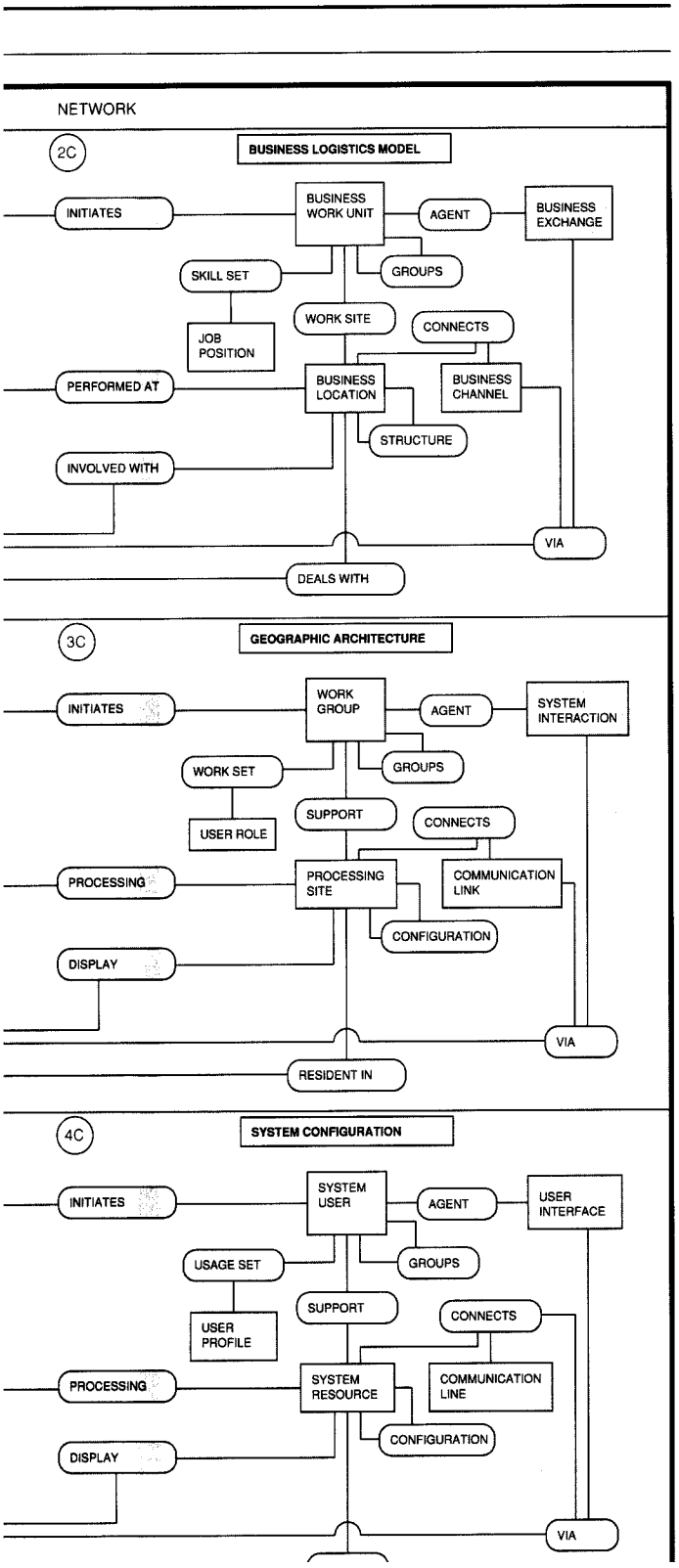
	DATA ENTITY RELN	FUNCTION FUNCTION ARG	NETWORK NODE LINK
SCOPE PLANNER	**LIST OF THINGS IMPORTANT TO THE BUSINESS**  ENTITY = CLASS OF BUSINESS THING	**LIST OF PROCESSES THE BUSINESS PERFORMS**  FUNCTION = CLASS OF BUSINESS PROCESS	**LIST OF LOCATIONS IN WHICH THE BUSINESS OPERATES**  NODE = MAJOR BUSINESS LOCATION
ENTERPRISE MODEL OWNER	**E.G., "ENT/REL DIAGRAM"**  ENTITY = BUSINESS ENTITY RELN = BUSINESS CONSTRAINT	**E.G., "PROCESS FLOW DIAGRAM"**  FUNCTION = BUSINESS PROCESS ARG = BUSINESS RESOURCES	**E.G., LOGISTICS NETWORK**  NODE = BUSINESS LOCATION LINK = BUSINESS LINKAGE
SYSTEM MODEL DESIGNER	**E.G., "DATA MODEL"**  ENT = DATA ENTITY RELN = DATA RELATIONSHIP	**E.G., "DATA FLOW DIAGRAM"**  FUNCTION = APPLICATION FUNCTION ARG = USER VIEW	**E.G., DISTRIBUTED SYSTEM ARCHITECTURE**  NODE = I/S FUNCTION (PROCESSOR, STORAGE, ETC) LINK = LINE CHARACTERISTICS
TECHNOLOGY MODEL BUILDER	**E.G., DATA DESIGN**  ENT = SEGMENT/ROW RELN = POINTER/KEY	**E.G., "STRUCTURE CHART"**  FUNCTION = COMPUTER FUNCTION ARG = SCREEN/DEVICE FORMAT	**E.G., SYSTEM ARCHITECTURE**  NODE = HARDWARE/SYSTEM SOFTWARE LINK = LINE SPECIFICATIONS
COMPONENTS SUB-CONTRACTOR	**E.G., DATA DEFINITION DESCRIPTION**  ENT = FIELD RELN = ADDRESS	**E.G., "PROGRAM"**  FUNCTION = LANGUAGE STMT ARG = CONTROL BLOCK	**E.G., NETWORK ARCHITECTURE**  NODE = ADDRESS LINK = PROTOCOL
FUNCTIONING SYSTEM	**E.G., DATA**	**E.G., FUNCTION**	**E.G., NETWORK**

Figure 2 Detailed cell metamodel



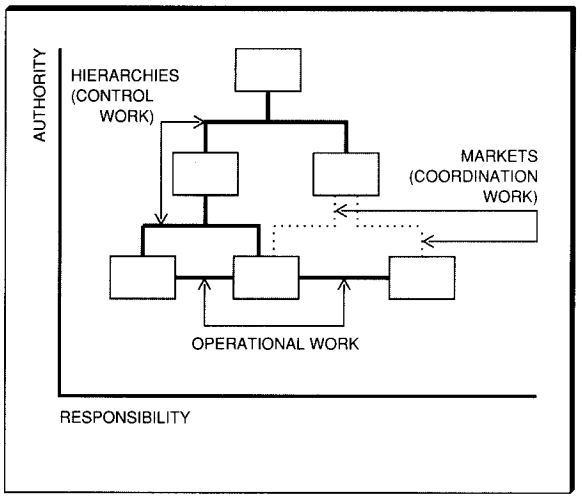


For example, it is complicated enough to design the process-to-process relationships of an enterprise without attempting to address the entity-to-entity and location-to-location design issues at the same time. However, since the processes, entities, and locations are all abstractions of the same enterprise, each is related to all of the others. Therefore, during the design of any one, the structural integrity of the others could undoubtedly be impacted. The challenge here is to design each while understanding the impact on the integrity of all others to avoid being surprised by undesirable side effects appearing long after it is possible to contain them. The evidence of this wisdom (or lack thereof) is seen in the majority of applications portfolios of today. For 50 years, we unknowingly optimized the function at the expense of the data, and only now are we beginning to discover the considerable, negative side effects.

Although the original framework suggested modeling constraints in each of the 15 cells, it did not include a formal specification language that would be suitable for detailed system design or for populating a repository for AD/Cycle or other computer-aided software engineering (CASE) tools. E-R diagrams are one specification language that has been used to describe the ISA framework and thereby constitute the basis for a repository model which could, in turn, store the enterprise models. As an example, Figure 2 represents an E-R style of definition for several of the framework cells.

When the framework was first defined,[1] it was fairly easy to find examples of all of the cells in the data and process columns. Creative work was even done in the network column when the concepts of distributed processing were first popularized. However, this work was not widely adopted by the practitioners in the data processing community. The advent of workstation technology, client-server concepts, and distributed systems is once again focusing attention on the network issues, and it is likely that formalisms will begin to proliferate as remote processing and storage become a reality. In any case, examples of many of the cells of the original three-column framework were available either in the practicing or theoretical communities to empirically validate the framework cells as they were originally described.

Figure 3 Sample organization graphic



The extended ISA framework

The three columns in Figure 1 correspond to the three English question words *what*, *how*, and *where*. For each of the five rows, Column A shows what entities are involved, Column B shows how they are processed, and Column C shows where they are located. Since the original ISA framework version was published in 1987, it has been extended by considering the other three question words in English: *who*, *when*, and *why*. Each of these words directs attention to a different focus on each of the five rows: who works with the system, when events occur, and why these activities are taking place. The six questions for each of the five rows lead to 30 different perspectives on an information system.

Few formalisms, at least in the traditional data processing community, are available for the who, when, and why column cells. Some formalisms exist in other disciplines, or even in some of the very specialized (but less generally understood) data processing circles. But their general unavailability causes the new cell descriptions to be more theoretical and less empirical. Therefore, it is important to understand and rigorously abide by the rules of the framework while hypothesizing the contents of the cells of the other three columns. The lack of commonly accepted formalisms for these columns also means that the terminology is not based on long-established traditions. As the state of the art advances, new insights will be gained, and the terminology may become more precise and standardized. Until then, the names of the cell contents used in this paper should be considered as suggestive, but not definitive.

The people (who) column. It is useful to abstract the concept of people out of the real-world enterprise because of the significance of designing the organizational infrastructure or the people-to-people relationships of the enterprise. The organization design challenge has to do with the allocation of work and the structure of authority and responsibility. Therefore, the basic columnar model is people-work-people, and the classic organization chart is a graphic depiction of the basic model. See Figure 3.

The vertical dimension of the graphic represents the delegation of authority, and the horizontal dimension represents the assignment of responsibility. Classic organization charts do not usually attempt to display the definition of the authority work product or the functional (or responsibility) work product. These work definitions, if they are defined at all, are presented in text as supplementary documents. However, if it is assumed that organizations and work design were perceived to be important enough to ascribe engineering discipline to them, it is not inconceivable that a graphic formalism in the order of Figure 3 could be developed in support of actual organization design.

With regard to the diagram in Figure 3, the organizational dynamics community defines two styles of work allocations: markets and hierarchies.[9] In brief, an enterprise will form into a free market structure if the nature of the transaction between two organization units is simple, well-defined, and universally understood. In this case, the organization (or person) with work to assign would survey all possible workers to find one who is acceptable in terms of availability and cost. This method is much like a stock buyer who scans the pool of stockbrokers to find one who will execute a buy within an agreeable time and for a reasonable fee.

In contrast, when the intraorganizational transaction is complex, not well-defined, and not universally understood, the enterprise will establish a hierarchy, that is, a regulatory organization that will arbitrarily define the work product, the

Table 2 Contents of cells in the people (who) column

Row	Perspective	Cell Example	Agent	Work
1	Planner	Organization list	Class of agent	
2	Owner	Organization chart	Organization unit	Work product
3	Designer	Human interface architecture	Role	Deliverable
4	Builder	Human/technology interface	User	Job
5	Subcontractor	Security architecture	Identity	Transaction

Table 3 Contents of cells in the time (when) column

Row	Perspective	Cell Example	Event	Cycle
1	Planner	List of events	Major event	Major cycle
2	Owner	Master schedule	Business event	Business cycle
3	Designer	Processing structure	System event	Processing cycle
4	Builder	Control architecture	Execute	Component cycle
5	Subcontractor	Timing definition	Interrupt	Machine cycle

schedule, and the cost that connects the subordinate organizations. This method is much like an oil company that arbitrarily defines the refining product slate, schedules, and transfer pricing of the product from the refinery as it is transferred to the wholesale or retail distribution organization.

There is no requirement that either the entity that allocates the work or the one to whom work is allocated be an organization or a person. It may well be a machine or even some software agent, as in artificial intelligence systems. Therefore, instead of specifying the columnar abstraction entity as an organization, person, or user, it seems more appropriate to select a generic name like agent that might apply to humans or nonhumans.

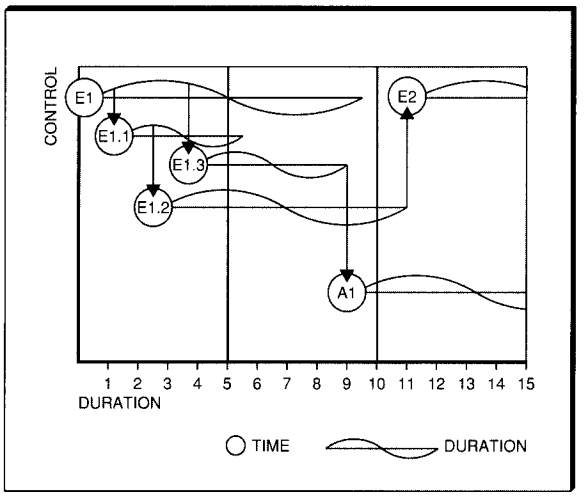
The connector entity, work, should probably be interpreted to mean work product to avoid confusing it with the concept of process or function that falls within the purview of the process column. Further, work product would be a kind of user view of materials or process inputs/outputs. To avoid any confusion between the concepts of work products and inputs/outputs, it would be useful to change the original name of the connector entity in the process column from input/output to argument, the mathematical term for input to a function that is the object of a transformation as in $f(x) = K$, where f is the transformation, x is the input argument, and K is the output.

Table 2 shows the kind of information that would be found in each of the rows of the people (who) column. The meaning of the entities in the column changes with the change of perspective from row to row, and their meaning is consistent with the other column cells in the same row. That is, the new cells are consistent with the overall framework, vertically and horizontally.

The time (when) column. Time is abstracted out of the real world to design the event-to-event relationships that establish the performance criteria and quantitative levels for enterprise resources. For example, from Event 1, product announcement at time t_0, until Event 2, first customer ship at time t_1, there is a duration $(t_1 - t_0)$. The length of the duration establishes the external commitments of the enterprise as well as the resource levels required to meet the commitments. In general, the shorter the duration, the more resources required to meet the commitments. The longer the duration, the less resources required to meet the commitments.

Figure 4 is an example of a graphic representation that might be appropriate for describing the time characteristics of an enterprise where the vertical axis is the control axis and the horizontal axis is the duration axis. The points in time are displayed as circles, and the durations are shown as cycles. Table 3 lists the kind of information that would be found in each of the cells of the when column.

Figure 4 Sample time-model graphic



The components of the basic columnar model are points in time (events) and lengths of time (durations), and the enterprise design challenge is to produce a schedule of events and states that maximizes the utilization of available resources while at the same time satisfying the external commitments. The cell models in the time column would once again maintain the vertical and horizontal consistency in conformance with the rules.

The motivation (why) column. The why column would be comprised of the descriptive representations that depict the motivation of the enterprise, and the basic columnar model would likely be ends-means-ends, where ends are objectives (or goals) and means are strategies (or methods). Figure 5 shows a hypothetical model, constructed around a video tape store case study that has been used in a number of publications.[10] This simplistic model illustrates the basic points; more detailed

Figure 5 Sample ends-means model

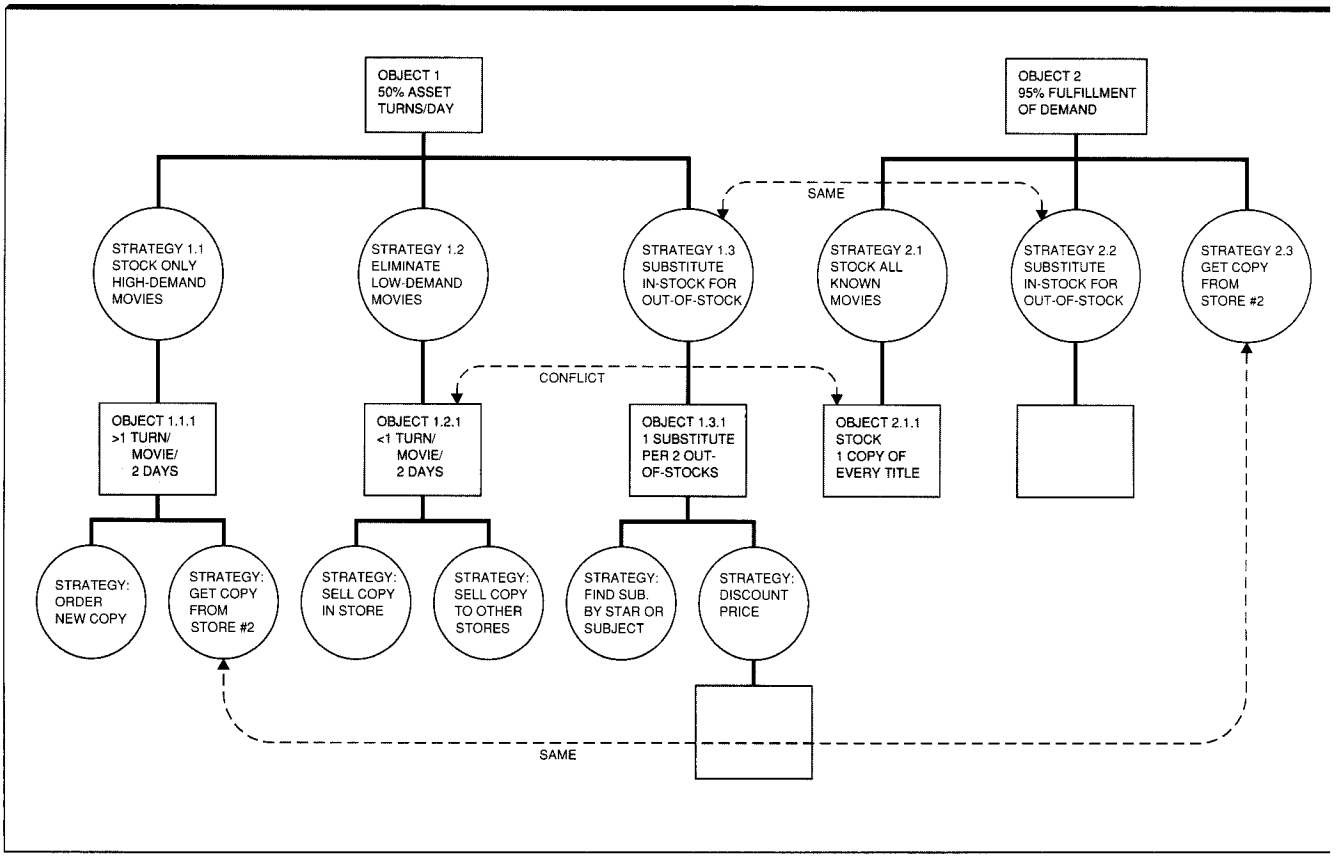


Table 4 Contents of cells in the motivation (why) column

Row	Perspective	Cell Example	Ends	Means
1	Planner	Objectives list	Major objectives	Major strategies
2	Owner	Business plan	Business objectives	Business strategy
3	Designer	Knowledge architecture	Criterion	Option
4	Builder	Knowledge design	Condition	Action
5	Subcontractor	Knowledge definition	Subcondition	Step

representations of goal-subgoal trees are used in game-playing programs and planning programs in artificial intelligence.

The cell contents in the motivation column as derived according to the framework rules represent the ends-means-ends motivation shown in Table 4.

The complete six-column framework is shown in Figure 6.

To illustrate the extended framework with all six columns, Figure 6 shows the kinds of descriptions that go into each cell. Figure 7 is a hypothetical case, showing English descriptions of some of the data, functions, network, organization, schedules, and strategies of the Oz Car Registration Authority (OCRA). These descriptions were derived from the specification of a car registration system that was used to compare several different methodologies for conceptual schemas.[11]

A natural language such as English is universal in the sense that it can describe anything that can be described. Yet natural languages have potential ambiguities. Those ambiguities are both a help and a hindrance. In the early design stages, they allow decisions to be deferred until further analysis has been done. But in the later stages, they prevent the results from being compiled automatically into executable code. The example in Figure 7 shows the use of English in an early design stage. E-R diagrams are a more formal notation that can represent some of the information expressed in English, but not all of it. Symbolic logic, in either the predicate calculus or the conceptual graph notation, is both formal and universal: it can describe everything that can be described, and it can be translated into executable code.

Rules of the framework

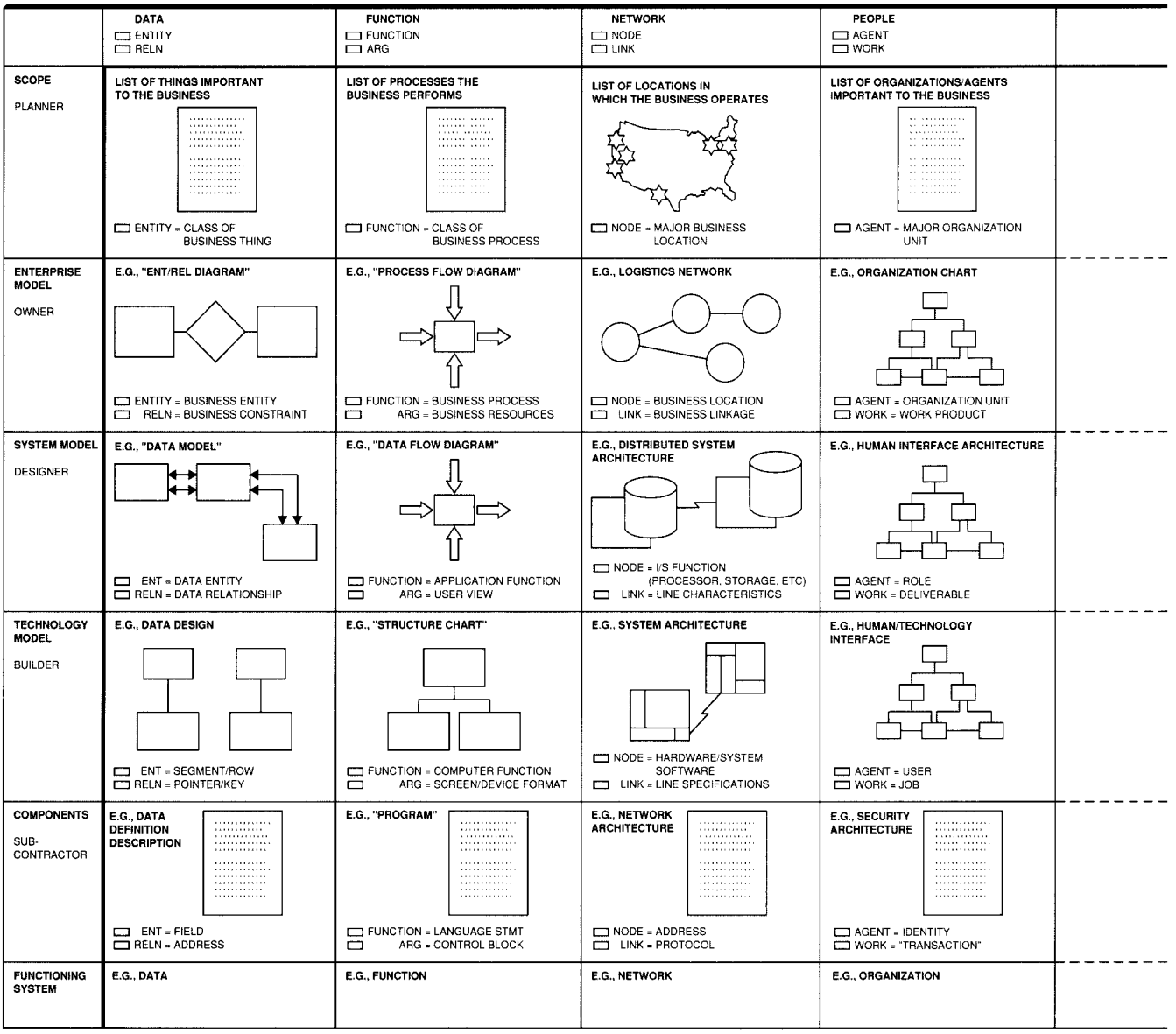
The previous two sections of this paper have shown examples of the ISA framework and the contents of each cell. This section presents the rules of the framework in a more abstract way. The reader may find it helpful to look back at the examples in order to get a clearer understanding of the rules.

Rule 1. *The columns have no order.* Order implies priorities. It creates a bias toward one aspect at the expense of others. Traditional programmers, for example, tend to have a bias toward function. They usually prefer to see the function column first in the framework. They start by designing algorithms that implement the function and leave the data as an afterthought. In fact, they may even claim that there is no need to expend resources on defining other models; that is, the functional or process models are adequate in themselves. As a result, all other aspects would be (inadvertently) ignored or suboptimized.

By the same token, programmers from the data community prefer that the data column be ordered first in the framework because history has proved that if the data are not designed first, they will invariably be compromised (suboptimized). By implication, process and network would be suboptimized in the interest of preserving the integrity of the data structure. There are also those who would claim that the network column should be first because the location of the data and processes really drives the design; they would cause the other aspects to be suboptimized. Equally, others might plausibly argue that the people, time, or motivation column should be first.

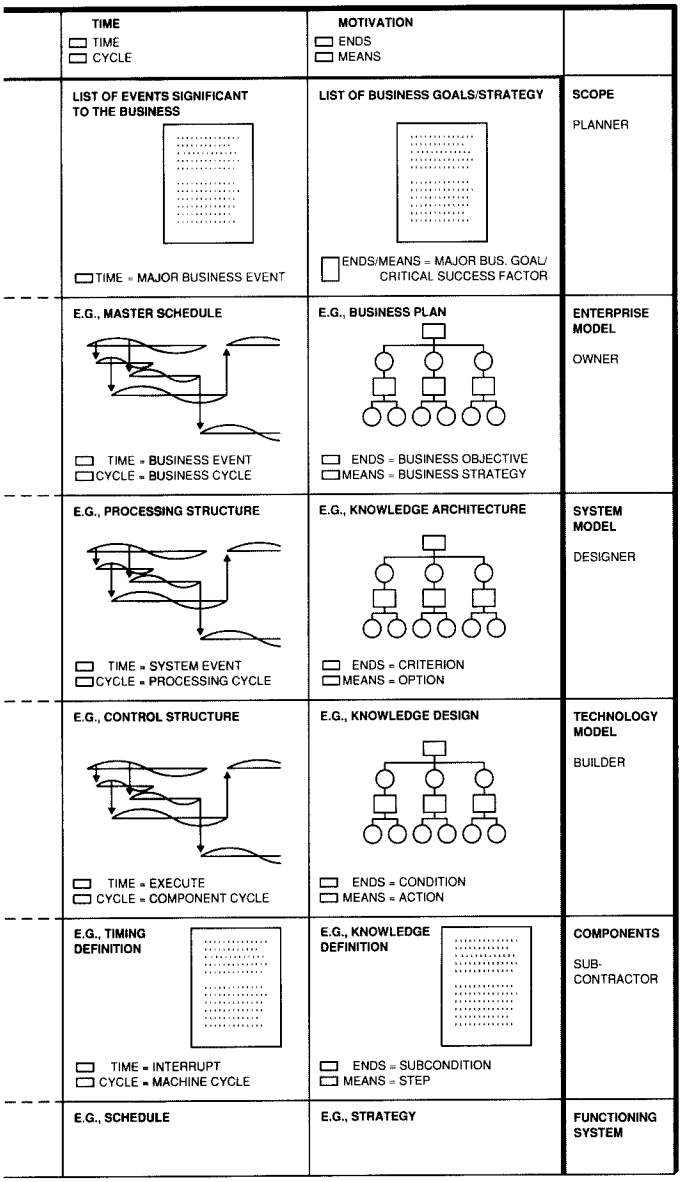
In any case, there is no natural order to the columns. Order or sequence implies method, which

Figure 6 Six-column ISA framework



embodies value judgments. All columns are equally important, for all are abstractions of the same enterprise. One or another may well be suboptimized at any given point in time, but it is important to understand why such a tradeoff is being made.

Rule 2. *Each column has a simple, basic model.* Each column represents an abstraction from the real-world enterprise for convenience of design. These abstractions correspond to a classification scheme suggested by the English interrogatives *what*, *how*, *where*, *who*, *when*, and *why*. The answers to these six questions are the basic entities or columnar variables: entities, functions, locations, people, times, and motivations. But in addition to these basic entities, the connections between them are also important for the design. The



data column, Column A, for example, has the simple basic model entity-relationship-entity. The columnar variable is entity, and the connector is relationship.

The basic model for each column is actually a generic metamodel. It is generic because it is the same for each cell in the column. It is "meta" because it is a model of the enterprise model. For example, the enterprise model in Cell A2 might be comprised of the sequence: employee related to organization related to customer related to product. The metamodel of this model is the abstraction entity-relationship-entity. In a similar fashion, each column has a simple, basic model that constitutes the generic metamodel for that column. Table 5 shows examples for Columns A, B, and C.

Rule 3. *The basic model of each column must be unique.* Uniqueness is essential for any useful classification scheme. Therefore, no entity or connector in the basic, columnar model is repeated, either in name or in concept. For example, entity and relationship are unique to Column A. Function and argument are unique to Column B. Entity is not equivalent to function, and relationship is not equivalent to argument. They may all be related to one another because they are all abstractions of the same real-world enterprise, but they are all separate and unique concepts.

The same logic applies to all of the basic, columnar models. That is, each basic model is unique.

Rule 4. *Each row represents a distinct, unique perspective.* This rule is most easily demonstrated in Rows 2, 3, and 4 which represent the owner's, designer's and builder's perspectives. Each perspective is different in that it is dealing with a different set of constraints relevant to that perspective. For example:

- Owner: Deals with usability constraints, both aesthetic and utilitarian in the conceptual view of the end product.
- Designer: Deals with the design constraints—the laws of physics or nature in the logical view of the end product.
- Builder: Deals with the construction constraints—the state of the art in methods and technologies in the physical view of the end product.

Because each perspective reflects a different set of constraints, the meaning (or definition) of the basic entity in a given column will change from row to row. For example, entity has one meaning for the owner, another one for the designer, and yet a different one for the builder. Table 6 shows examples of those differences for Column A.

Since each basic entity means something different from the perspectives of the different cells in the

Figure 7 The OCRA example in the six-column framework

	WHAT? ENTITY / REL	**HOW?** FUNCTION / ARG	**WHERE?** NODE / LINK	**WHO?** AGENT / WORK	**WHEN?** TIME / CYCLE	**WHY?** ENDS / MEANS
SCOPE PLANNER	OZ, OCRA, CARS, FEES, LICENSES, CAR HISTORIES	REGISTER, TRANSFER, COLLECT, ENFORCE	EMERALD CITY, MUNCHKIN LAND KANSAS, HOLLYWOOD	DIRECTOR, MANAGERS, CLERKS, CAR OWNERS	TIME OF SALE, TRANSFER, REGISTRATION, DESTRUCTION	REGULATE SALES, RAISE MONEY, TRACE CARS
ENTERPRISE MODEL OWNER	EACH CAR IS OF A PARTICULAR MODEL	OWNERSHIP IS TRANSFERRED BY REGISTRATION OF THE TRANSFER	REGISTRATIONS ARE RECORDED AT OFFICES OF OCRA	AN OCRA CLERK MUST RECORD EACH REGISTRATION	WHEN A CAR IS CONSTRUCTED, TRANSFERRED, OR DESTROYED	KEEP ACCURATE RECORDS AND COLLECT FEES
SYSTEM MODEL DESIGNER	FUNCTIONAL DEPENDENCY FROM CAR TO MODEL	CAR HISTORY UPDATED BY TRANSFER MODULE	EACH OFFICE MUST HAVE A CONNECTION TO OCRA HQ	A CLERK MUST ENTER INFORMATION AT A TERMINAL	DB UPDATES OCCUR AT IRREGULAR INTERVALS	OLD BATCH SYSTEM DOES NOT RESPOND FAST ENOUGH
TECHNOLOGY MODEL BUILDER	CAR RELATION HAS A COLUMN FOR MODEL IDENTIFIER	TRANSFER DONE BY COBOL PROG XFR397A	B.O. RECORDS ARE BACKED UP AT OCRA HEADQUARTERS	CLERK COMPLETES FORM REG972 TO INITIATE REGISTRATION	EACH MODULE IS INVOKED BY A MENU SELECTION	EFFICIENT, RELIABLE SERVICE WITHIN BUDGET
COMPONENTS SUB-CONTRACTOR	MODELID PIC X(15)	SELECT SNO FROM HIST WHERE...	INSTALL TCP/IP LINK TO OZNET	INSTALL CORDONS TO GUIDE QUEUE FOR CLERKS	USE POP-UP WINDOWS SELECTED BY MOUSE	MEET SPECIFICATIONS FOR EACH MODULE
WORKING SYSTEM	**DATA**	**FUNCTION**	**NETWORK**	**ORGANIZATION**	**SCHEDULE**	**STRATEGY**

Table 5 Components of the generic metamodels for Columns A, B, and C

	Column A Data (what)	**Column B Function (how)**	**Column C Network (where)**
Basic entity	Entity	Function	Node
Connector	Relationship	Argument	Line

Table 6 Changes in the meaning of entity from row to row of Column A

Row	Perspective	Basic Entity
2	Owner	Business entity (real-world thing)
3	Designer	Data entity (logical representation)
4	Builder	Technology entity (physical representation)

data column, the semantic contents of the cells in the column are different, which means, in turn, that the structure of the cell models in the same column is likely to be different. Note how the meaning changes for all rows and all columns of Figure 6.

Rule 5. *Each cell is unique.* Since each column has a unique basic model that makes each column unique, and since each row has a different perspective that makes the meaning of the basic model unique to each row, each cell in the framework is unique. That is, no meta entity can show up in more than one cell. For example:

- Business entity can only be found in Cell A2.
- Data entity can only be found in Cell A3.
- Business process can only be found in Cell B2.
- Application function can only be found in Cell B3.

Therefore, the ISA framework serves as a convenient classification scheme or "periodic table" for information entities. Like chemical elements,

these entities may be combined in endless ways to produce compounds or information systems of interest to an enterprise.

A corollary to the rule that each cell is unique (that is, each cell represents a different abstraction and different perspective—therefore different motivation, different purpose, different design issue, different constraints, etc.) is that different techniques and different graphic representations are appropriate for different cells. This corollary explains the plethora of information systems (I/S) design formalisms that have emerged over the years. They are all likely to be relevant for some purposes, but they each address a different set of issues, and none is completely adequate in itself. Also, when the formalism for any one cell is expanded to incorporate the notation from another cell in an attempt to enrich the formalism, it complicates the design problem and may lead to inadvertent suboptimization of the other independent variable. For example, in some application designs, it may seem appropriate to show a data store on a data flow diagram for Cell B3. Yet it may not be clearly understood that the data store is actually an aggregation of attributes of entities from the logical data model of Cell A3. There is a risk that the designer may design a customized file to satisfy the local requirements of the process. In this event, the integrity of the data as specified by the logical data model may be compromised, and the data are unlikely to be reused or shared by any other process.

The uniqueness of each of the cells also explains the plethora of methodologies that have evolved. It would appear that a specific methodology elects to produce some set (or subset) of cells in some sequence. The sequence determines the value system being applied in making the design tradeoff decisions within the cell. That is, the structure of a cell can be derived from the cell above, the cell below, or a cell in the same row. The cell that a methodology causes to be produced first is likely to have a strong influence on the design tradeoffs made in the structure of a subsequently designed cell.

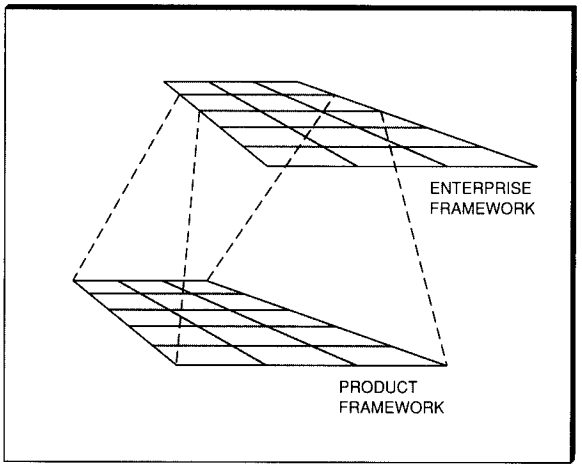
Rule 6. *The composite or integration of all cell models in one row constitutes a complete model from the perspective of that row.* This rule derives from the fact that any one cell of one column is merely a single abstraction of reality. Therefore, the sum of all cells in a given row is the most complete depiction of reality from the perspective of that row. The significance of this rule is that as additional columns are defined, each new cell description must be consistent with the perspective of the row. That is, each cell in a given row can be defined and has relevance independent of any other cells in the row, yet each cell is but one abstraction of the same reality. Therefore, at a minimum, each cell is related to every other cell in the same row. In some cases, there may even be a dependence upon other cells in the row. In these cases, a change in the structure of one cell would likely have some kind of effect in any other cell where a dependency exists. This not only holds true across a row, but it most certainly would be true within a column where, by definition, there is a dependency between any one cell and the cell above and the cell below. Thus, a change in any given cell would likely affect the cell above, the cell below, and potentially, other cells in the same row where a dependency exists.

It is worthwhile noting that if the nature of the dependency between cells could be understood and stored in the repository along with the cell models, it would constitute a very powerful capability for understanding the total impact of a change to any one of the models, if not a capability for managing the actual assimilation of the changes.

Rule 7. *The logic is recursive.* The framework logic can be used for describing virtually anything, certainly anything that has an owner, designer, and builder who make use of material, function, and geometry. The logic was initially perceived by observing the design and construction of buildings. Later it was validated by observing the engineering and manufacture of airplanes. Subsequently it was applied to enterprises during which the initial material on the framework was published. In the current paper, it is being applied to an information systems "enterprise" wherever the "meta" concept is being used. Similarly, it could be applied to a CASE tool manufacturer.

These four applications of the framework were selected for illustration, not by accident, but because the examples are related. Examination of the framework graphically depicts the relationship between the product, the enterprise, information systems, and the CASE tool manufacturer. It shows that:

Figure 8 The enterprise framework as a metaframework



- The owner of the product is the customer of the enterprise.
- The owner of the enterprise is the customer of information systems.
- The owner of information systems is the customer of the CASE tool manufacturer.

Similarly,

- The enterprise transforms the owner's view of the product, through a series of product model transformations, into the product itself.
- The I/S organization transforms the owner's view of the enterprise, through a series of enterprise model transformations, into the enterprise itself.
- The CASE tool manufacturer transforms the owner's view of the I/S organization, through a series of I/S model transformations, into the I/S organization itself.

Since the product is related to the enterprise, which is related to the I/S organization, which is related to the CASE tool manufacturer, the respective frameworks are also related.

For example, Cell A2 of the enterprise framework (owner's row, data column) is a model of the product framework because in manufacturing the product, the enterprise, by definition, is producing all of the cells of the product framework. Therefore, the semantic model of the enterprise would necessarily have to incorporate all of the design artifacts required to build the product, plus extensions to the model to describe the enterprise resources being used in the manufacturing process. Thus, Cell A2 is a metamodel of the product framework with extensions that include the enterprise resources. See Figure 8.

Cell B2 of the enterprise framework (owner's row, process column) is a model of the functions required to produce all of the cells of the product framework, extended to include the processes required to manage the enterprise resources. In this fashion, Cell B2 is a metamodel (or process model) of the product framework plus some extensions.

Cell C2 of the enterprise framework (owner's row, network column) is a model of the locations required to produce all of the cells of the product framework, extended to include those locations required for managing the enterprise resources. In this fashion, the cells of Row 2 (owner's row) of the enterprise framework are metamodels of the product framework extended as required to manage the enterprise resources.

By the same token, the Row 2 (owner's row) models of the I/S framework are the metamodels of the enterprise framework, with extensions required to manage the I/S resources.

Similarly, the CASE tool manufacturer framework Row 2 (owner's row) cells are the metamodels of the ISA framework, with extensions required to manage the CASE manufacturer's resources.

The CASE tool manufacturer's framework has some interesting peculiarities in that it looks very similar to the I/S organization framework. The reason is the products of both of these organizations are applications. The only difference is that the CASE tool products are applications for building applications, whereas the I/S organization products are applications for building (enterprise) products. Although these frameworks may be quite the same generically, the instances of the cell models may differ substantially because the two organizations are likely to have dramatically different strategies, methodologies, geography, etc., which would mean the structure of the models would depart dramatically. Figure 9 shows the metarelationships between the frameworks.

There is another dimension to the recursiveness of the framework logic in that in any given enterprise, there may be as-is versions and to-be versions of each of the cell models. Therefore, the total set of frameworks that may be of interest (and therefore may require managing) might be depicted as in Figure 10.

Still another dimension of recursiveness is possible and is being considered as part of the conceptual schema for the IRDS. It is the possibility of applying the logic of the framework to the framework itself. That is, any given cell is a complex engineering product in its own right. It has an owner, designer, builder, material, function, and geometry. Therefore, the framework logic could be applied to each of the cells of the framework to analyze the design and construction issues that affect that cell.

Although these three dimensions of recursiveness (related frameworks, framework versions, and nesting frameworks) expose the considerable complexity of the architecture issue, the fact that the simple logic of the framework can be employed recursively opens up the possibility of:

- Leveraging the reusability of the logic to advance the state of the art and extend the body of knowledge
- Technically managing the relationships between all of the models (for the purposes of configuration management and change assimilation) through such techniques as versioning

Since the storage mechanism (repository) could not in itself differentiate between one framework and another, the same repository could be used for managing all of the frameworks merely as versions. This factor brings architecture management into the realm of feasibility. For this reason it is imperative to begin to acquire the capabilities for producing and managing architectures. It is only a matter of time before the technology will allow managing enterprise change beyond the limits of our current imagination.

Before employing the rules of the framework for defining the other three columns (who, when, and why), it is necessary to state the caveat once again. That is, there is not a lot of precedent in the data processing community for cells in the columns that represent these other three abstractions. Examples are abundant for the process and

Figure 9 Set of interesting metaframeworks

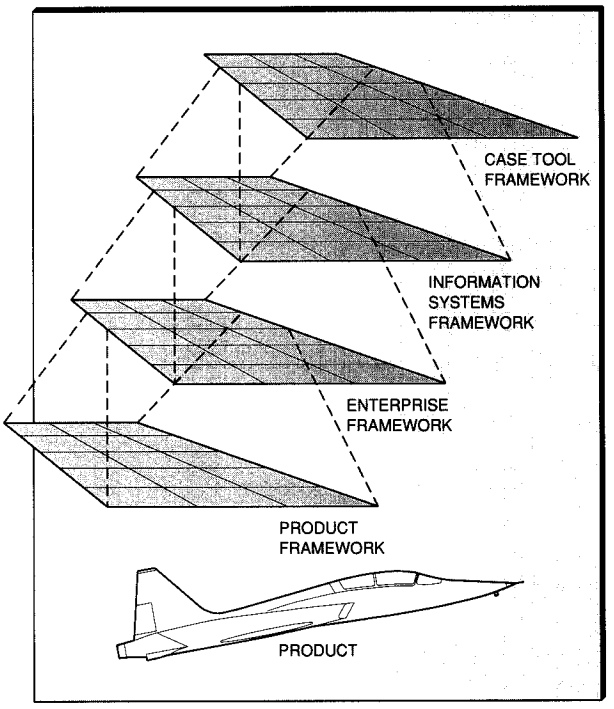


Figure 10 Set of framework versions

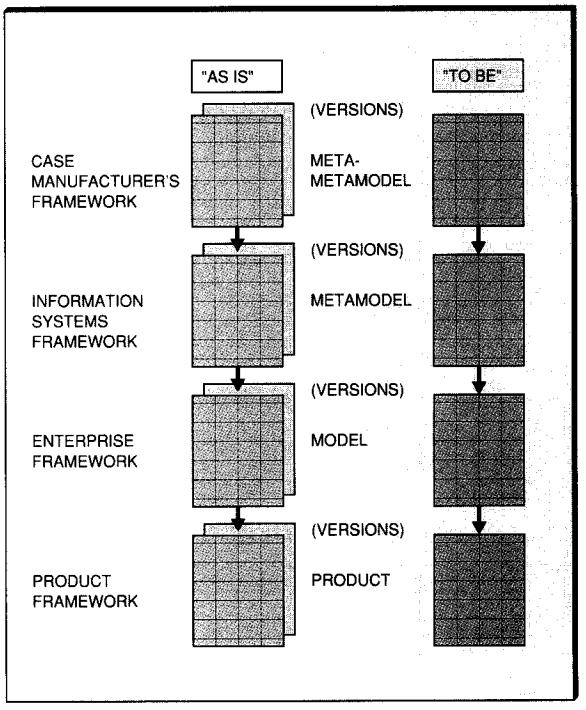
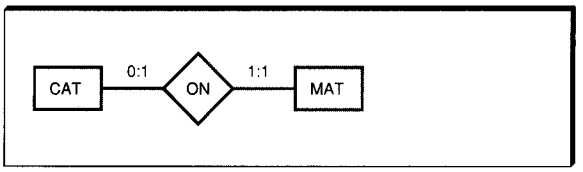


Figure 11 An entity-relationship diagram



data columns. Fewer are available in the network column. But there is a scarcity of good examples in the people, time, and motivation columns. Therefore, any definition of cells in these other three columns must necessarily be more hypothetical and less empirical. The rules of the framework must be adhered to in order to preserve the conceptual integrity of the classification scheme.

It is also important to note that examples of the cells in the last three columns have been developed in other disciplines, including some research areas in computer science. A great deal of thought has been devoted to these issues in psychology, sociology, industrial engineering, organizational dynamics, artificial intelligence, real-time systems, game theory, distributed systems, business administration, and other fields. These fields have a rich supply of knowledge yet to be tapped and mapped into the framework in a form that can be used by the data processing profession. Therefore, even though the basic concepts defined by the framework rules are likely to be stable over time as a classification system, the specific names and examples are likely to change as more is learned and as other disciplines can be surveyed for appropriate contributions.

Overview of conceptual graphs

Conceptual graphs are a system of logic that can be used in conjunction with other graphic notations, such as entity-relationship diagrams.[12] Unlike E-R diagrams, however, conceptual graphs are as general as predicate calculus and can express all of the relationships and constraints that affect an enterprise and its information system. To illustrate the issues, consider the ON relation for cats on mats, as expressed in three different modeling languages: E-R diagrams, symbolic logic, and conceptual graphs. Figure 11 shows an E-R diagram for the entity type CAT linked by the ON relation to the entity type MAT.

The pairs of numbers on the arcs of Figure 11 are called *participation counts*. They show the lower and upper bounds on the number of instances of each entity type that may be associated with entities of the other type. The pair 0:1 on the left shows that zero or one cat is on each mat, and the pair 1:1 on the right shows that one and only one mat is associated with each cat. Together, they imply that every cat is on a unique mat but that some mats may not have any cats. In symbolic logic, that same information may be stated in the basic notation for *first-order predicate calculus*:

$$(\forall x)(\exists y)(\mathrm{cat}(x) \supset (\mathrm{mat}(y) \wedge \mathrm{on}(x,y)) \wedge (\forall z)((\mathrm{mat}(z) \wedge \mathrm{on}(x,z)) \supset z=y) \wedge (\forall w)((\mathrm{cat}(w) \wedge \mathrm{on}(w,y)) \supset w=x)).$$

The first line of this formula says that every cat is on a mat: literally, it may be read *For every x, there exists a y, where if x is a cat, then y is a mat and x is on y*. The second line says that there is only one mat for each cat: *For every z, if z is a mat and x is on z, then z is identical to y*. The third line says that there is only one cat on each mat: *For every w, if w is a cat and w is on y, then w is identical to x*. The complexity and unreadability of formulas like these is the main reason why database designers and systems analysts do not like to use predicate calculus.

The need for an extended notation to simplify such formulas was recognized as early as 1910. In the *Principia Mathematica*, Whitehead and Russell[13] introduced the relational operators E! for exactly one and E!! for uniqueness. For the operator E!, there is a corresponding quantifier $(\exists!x)$, which means that there exists exactly one x. Uniqueness is more complex, since it must be expressed by a pair of quantifiers; the operator E!! corresponds to the quantifiers $(\forall x)(\exists!!y)$, which mean that for every x there exists a unique y. With such quantifiers, the second and third lines of the preceding formula can be eliminated:

$$(\forall x)(\exists!!y)(\mathrm{cat}(x) \supset (\mathrm{mat}(y) \wedge \mathrm{on}(x,y)).$$

This formula may be read *For every x, there exists a unique y, where if x is a cat, then y is a mat and x is on y*. In 1938, the logician Arnold Schmidt introduced *sorted logic* with sort or type labels for each variable.[14] Such a notation simplifies the formula further:

(∀x:cat)(∃!y:mat)on(x,y).

This formula may be read *For every cat x, there exists a unique mat y, where x is on y*. The variables x and y are the main features that make this formula sound unnatural in comparison to English. A graph notation can reduce or eliminate the need for variables by showing connections directly.

Conceptual graphs are a system of logic designed to map to and from natural languages in as simple and direct a manner as possible.[4] They are based on the existential graphs by the logician Charles Sanders Peirce,[15] the dependency grammars by the linguist Lucien Tesnière,[16] and the semantic networks that are widely used in artificial intelligence.[17] They combine extended quantifiers and type labels in a readable graphic notation. They have been used and implemented by research and development groups around the world. And they have been the subject of seven annual workshops from 1986 to 1992. For these reasons, the ANSI Task Group X3H4.6 has chosen them as the basis for the normative language of the IRDS conceptual schema. Figure 12 shows the conceptual graph for the sentence *Every cat is on a unique mat.*

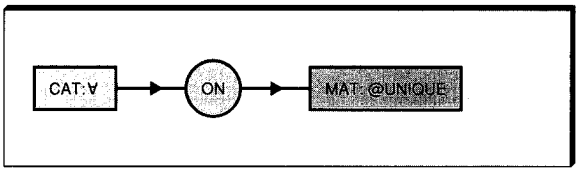
The boxes in a conceptual graph represent *concepts*, and the circles represent *conceptual relations*. CAT and MAT are *type labels* that correspond to the sort or type labels in sorted logic, and ∀ and @unique are quantifiers that correspond to ∀ and ∃!! in predicate calculus. The graph notation eliminates the variables x and y by using arcs that link the concepts and relations directly. When conceptual graphs are mapped to predicate calculus, variables are assigned to the concept nodes. The arrow pointing toward the circle shows the first argument of the relation, and the arrow pointing away shows the second argument (relations with more than two arguments have numbers on the arcs).

To save space on the printed page, Figure 12 can also be written in a linear notation that uses square brackets for the concepts and rounded parentheses for the circles:

[CAT: ∀]→(ON)→[MAT: @unique].

The linear form can also be written with only the ASCII character set:

Figure 12 A conceptual graph



[CAT: @every]->(ON)->[MAT: @unique].

The graphic form is usually the most readable, but the linear form takes less space on the printed page, and the ASCII form is convenient for interchange between systems.

Putting quantifiers in the boxes with the type labels allows a more direct mapping to English than the participation counts in E-R diagrams. The concept [CAT: ∀] represents the English phrase *every cat,* and [MAT: @unique] represents a *unique mat*. They also allow a direct mapping to the quantifiers in sorted predicate calculus: (∀x:cat) and (∃!!y:mat). The pair of participation counts 0:1 and 1:1 correspond to the quantifier @unique, but other combinations do not have such a simple correspondence. To express participation counts such as 2:7 and 5:15, two conceptual graphs would be needed:

[CAT: ∀]→(ON)→[MAT: {*}@5:15].
[CAT: {*}@2:7]→(ON)→[MAT: ∀].

The first graph may be read *Every cat is on 5 to 15 mats*; and the second may be read *2 to 7 cats are on every mat*. The symbol {*} is the *generic plural* marker, which represents a set of unspecified elements whose type is determined by the type label of the concept. The quantifier @5:15 indicates that the count or cardinality of the set ranges from 5 to 15. Getting the cats to sit still long enough for such a situation to be set up may be a challenge, but if it can be done, conceptual graphs can describe it. Furthermore, the description can be mapped to English in a readable way.

E-R diagrams are primarily used as a metalanguage for talking about database designs. They cannot be used to represent actual instances of data in the database. Conceptual graphs, however, can express statements about instances in

Figure 13 A conceptual graph with instances

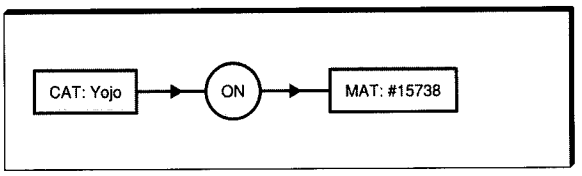
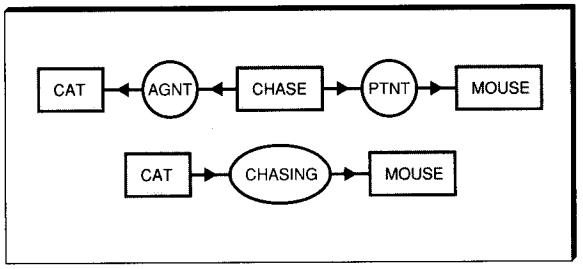


Figure 14 Two graphs for showing a cat chasing a mouse



the database as well as quantified statements that represent E-R diagrams. Figure 13, for example, represents the statement *The cat Yojo is on the mat #15738*. In each concept box, the colon separates the *type field* on the left from the *referent field* on the right. The referent field may contain quantifiers like ∀ and @unique, plurals like {*}@5:15, proper names like Yojo, serial numbers like #15738, or even variables like *x or *y. The pure graph notation does not require variables, but the linear form needs them to show cross references. The referent field may also be blank: the concept [CAT] means that there exists a cat, but its identity is not known.

Since conceptual graphs are designed to represent the semantics of natural languages, the basic conceptual relations are derived from the *case relations* or *thematic roles* of linguistic theory. Examples of those relations include AGNT for the *agent* of an action, PTNT for the *patient* or thing acted upon, RCPT for the *recipient*, and INST for the *instrument* or means by which an action is performed. Case relations would be familiar to a speaker of ancient Latin or modern Russian, but not to most English speakers. A linguist who is designing a natural language system to map English into conceptual graphs would have to know the case relations. But even without linguistic training, a database designer or systems analyst could use conceptual graphs with no more linguistic detail than E-R diagrams.

Figure 14 shows two conceptual graphs that illustrate two different levels of detail. Both graphs represent the sentence *A cat is chasing a mouse*. The first graph represents the verb *chase* by the concept [CHASE]. It uses linguistic relations to show that the cat is the agent and the mouse is the patient. The second graph uses the relation CHASING to link the concepts of the cat and mouse directly.

Both conceptual graphs in Figure 14 are equally valid, but they are optimized for different purposes. The first graph with the relations AGNT and PTNT is more appropriate for mapping conceptual graphs to English and other natural languages. The second is more appropriate for a database design where the linguistic details are not relevant. To show how the two graphs are related, the following definition relates the high-level relation CHASING to the concept type CHASE and the lower-level relations AGNT and PTNT:

relation CHASING(x,y) **is**
[ANIMATE: *x]←(AGNT)←[CHASE]-
(PTNT)→[MOBILE-ENTITY: *y].

This definition says that the relation CHASING relates an animate being x to a mobile entity y, where x is the agent of CHASE and y is the patient. The type labels ANIMATE and MOBILE-ENTITY specify the most general types that could do the chasing or be chased. They would include a boy chasing a kite or a dog chasing a truck. By expanding the definition of CHASING, the second graph in Figure 14 could be converted to the first; by contracting the definition, the first graph could be converted to the second. A top-down design could start with high-level relations such as CHASING and later define them in terms of the more primitive ones. The definitional mechanisms provide a way to restructure the description in different sets of primitives.[18]

Besides concepts for entities and actions, the full ISA framework requires concepts and relations for showing times and purposes. In Figure 15, the graph for a cat chasing a mouse is nested inside a concept of type SITUATION. The *inner context* with the nested graph describes the situation, and the *outer context* contains concepts and relations that say how the situation relates to external

times, places, people, and things. The relation DUR for *duration* shows that the situation lasted for a time period of 13 seconds. The relations FROM and TO show that the time period started at the time 19:29:32 Greenwich Mean Time (GMT) and ended at 19:29:45 GMT.

Flowcharts and Petri nets are often used to describe processes in Column B of the ISA framework. Such diagrams can also be represented in conceptual graphs by using nested graphs linked by the SUCC or *successor* relation. Figure 16 shows a concept of type PROCESS, which contains a nested state s1, followed by an event e, followed by another state s2. The state s1 has a duration of 15 seconds, the event e occurs at a *point in time* (PTIM) of 20:23:19 GMT, and the state s2 has a duration of 5 seconds.

Conceptual graphs are a system of logic that remains readable at many different levels of detail. Predicate calculus, by contrast, is not very readable at any level of detail. The *formula operator* $\phi^{3,4}$ would translate Figure 16 to the following:

```
(∃p)(process(p) ∧ descr(p,
  (∃s1)(∃e)(∃s2)(∃t1)(∃t2)
    (state(s1) ∧ event(e) ∧ state(s2) ∧
        succ(s1,e) ∧ succ(e,s2) ∧
        time-period(t1) ∧ time-period(t2) ∧
        time(20:23:19 GMT) ∧
        dur(s1,t1) ∧ ptim(e,10:23:19 GMT) ∧
        dur(s2,t2) ∧
        measure(t1,15sec) ∧ measure(t2,5sec)))).
```

The unreadability of such formulas has given logic a bad reputation among practicing programmers. Yet that is not the fault of logic, but of the predicate calculus notation. Conceptual graphs are just as formal and precise, but they are a readable notation for representing any level of the ISA framework: enterprise models, information system models, technology models, or component models. Furthermore, they can be translated directly into English or other natural languages. Figure 16, for example, could be read as the following sentence in structured English: *There is a process p consisting of a state s1 of duration 15 seconds, followed by an event e at time 20:23:19 GMT, followed by a state s2 of duration 5 seconds.* Such English may not be elegant, but it would be useful for comments and help facilities. The possibility of generating it automatically from the formal description would ensure that the implementation and the documentation would always agree.

Figure 15 Showing the duration of a cat chasing a mouse

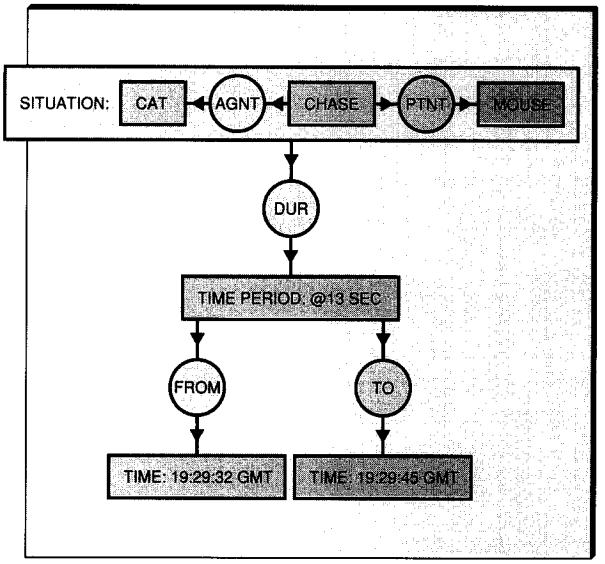
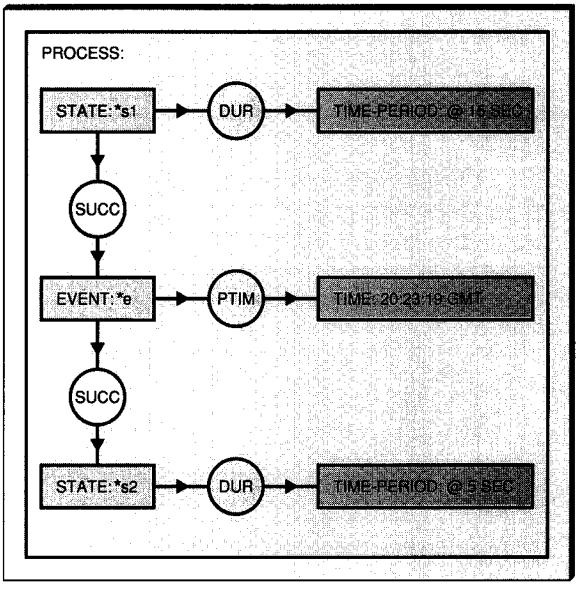
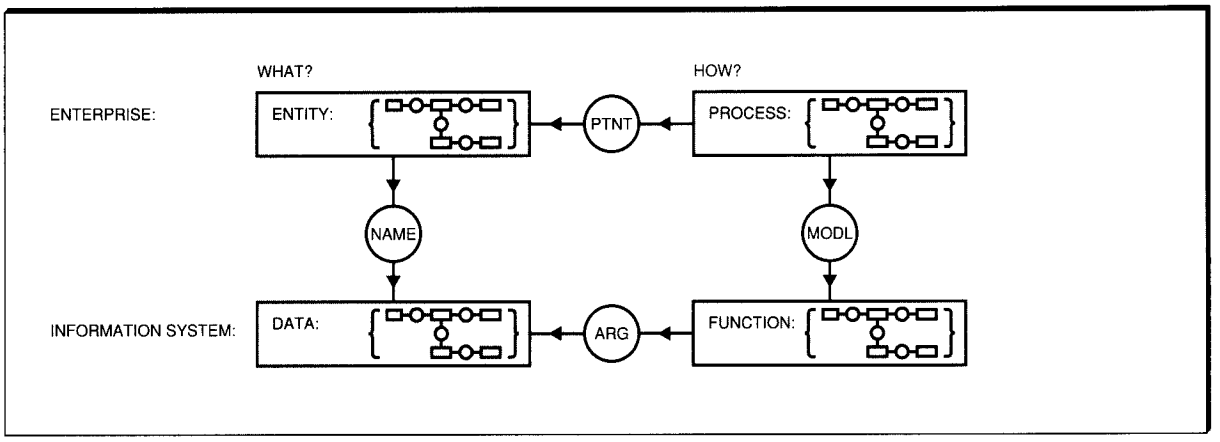


Figure 16 A process described by a conceptual graph



Although conceptual graphs are general enough to represent every cell of the ISA framework, it is not necessary to replace the older notations. E-R

Figure 17 Representing four cells of the ISA framework



diagrams are adequate to represent a subset of logic, and they can be formally translated into conceptual graphs. Any design that has been represented in E-R diagrams, data flow diagrams, or even flowcharts need not be rewritten. Instead, it can be mapped into conceptual graphs without change. Systems analysts who are familiar with the older technology need not change their ways of thinking until they feel the need to do so. The ANSI IRDS X3H4.6 Task Group has established some guidelines for migrating from one version of the IRDS standards to another.[5]

- Any conceptual schema represented in the current ANSI standards must be migratable to the new standards without manual intervention.
- Design tools based on the current standards may continue to be used indefinitely, and new tools should be upward compatible with them.
- Logic is general enough to represent any design; conceptual graphs are a readable graphic notation for all of logic, but no systems analyst should be forced to use the new notations for any task for which the old notations are adequate.

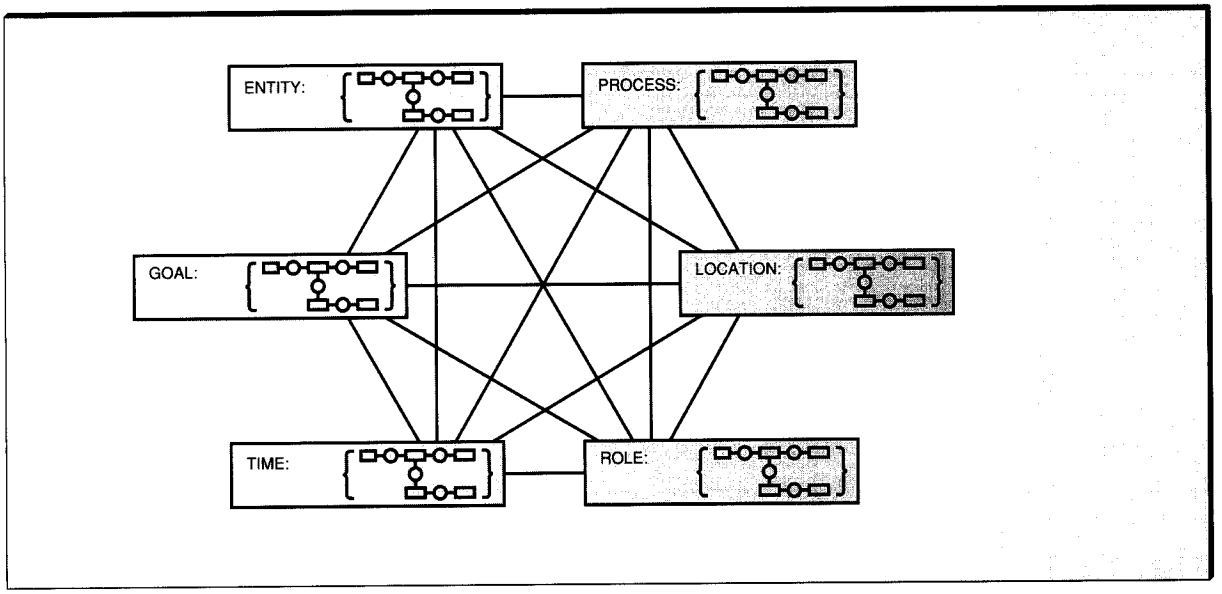
Conversions from one system to another rarely happen overnight, and new systems must be able to coexist with the old.

Representing the ISA framework in conceptual graphs

Conceptual graphs can describe the content of any cell in the ISA framework. Even more importantly, they can describe the relationships between cells. All of the information in one cell of the ISA framework can be placed in a single concept box. That concept would be a context that contained a set of graphs representing the contents of the cell. Figure 17 shows four concepts, each of which contains a set of graphs that represent one of the ISA cells. These four concepts represent the cells of Columns A and B, Rows 2 and 3. The NAME relation shows that the entities in Row 2, the enterprise row, are named by data in Row 3, the information system row. The MODL relation shows that the processes in Row 2 are modeled by functions in Row 3. The PTNT relation shows that the processes in Column B operate on the entities (the patients) in Column B. And the ARG relation shows that the functions in Column B take their arguments from the data in Column A.

At the overview level of Figure 17, the graphs inside the boxes are not readable. But with an interactive display, it would be possible to zoom in on any box to examine its contents. It would also be possible to zoom out and see all 30 cells of the framework nested inside a larger concept box. At an even higher level, the concept box representing version 1 of a framework could be related to the box for version 2 and another box for a version 3 that was still in the planning stage. Conceptual graphs can be used as the language for describing each level as well as the metalanguage for talking about how the different levels relate to one another.

Figure 18 All columns have equal status



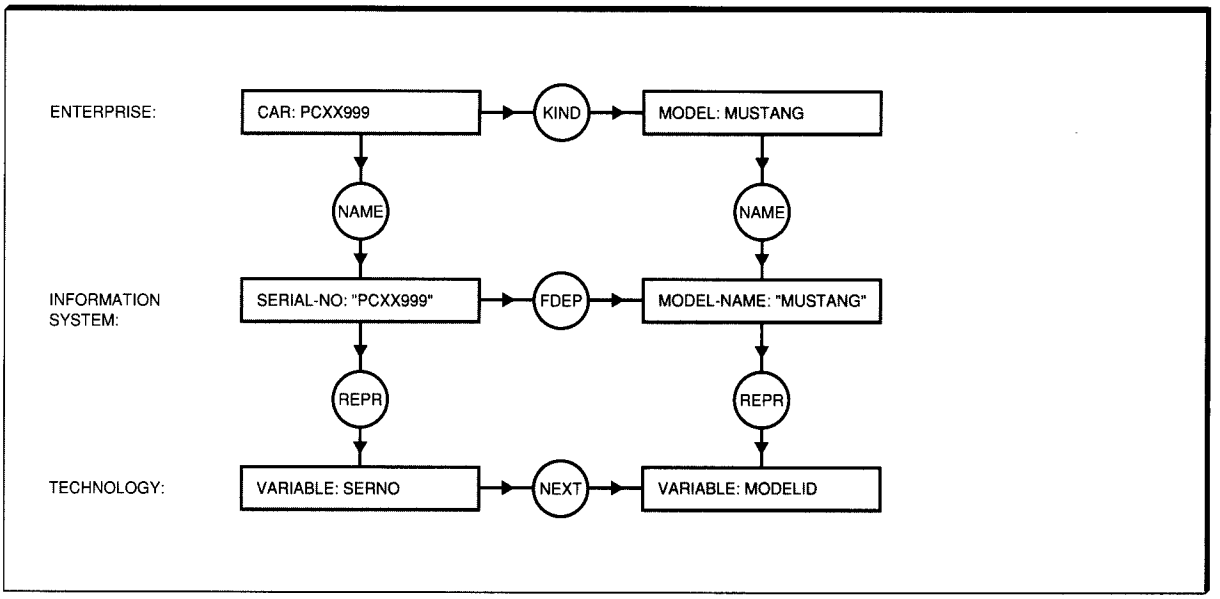
Rule 1. *The columns have no order.* Figure 1 shows the original ordering of the ISA framework. Figures 6 and 7 show the additional three columns. But the particular ordering is merely a historical accident. Figure 18 shows the six columns in a hexagon, where each one is related to every other. The traditional tabular order is merely a concession to paper or flat computer displays. Graphs eliminate the restrictions and permit anything in any column to be linked directly to anything in any other column.

Rule 2. *Each column has a simple, basic model.* In terms of conceptual graphs, this rule implies that there is one basic concept type for each column, which answers the question word at the head of the column. All of the graphs that describe any cell in that column assert some information about some subtype of that basic type. Following are the basic concept types as depicted in Figure 18 for each column:

1. The answer to the question *what* is some type of *entity*. For Rows 1 and 2, the entities are real-world objects. For Row 3, they are logical information types in the I/S model. For Row 4, they are physical data types in the technology model. For Row 5, they are more specialized data types for each component.
2. The answer to the question *how* is some type of *process*. For Rows 1 and 2, they are real-world processes. For the lower rows, they are computational functions that model the processes.
3. The answer to the question *where* is some type of *location*. For the top two rows, they are locations in the world. For the lower rows, they are logical or physical nodes in a computer network.
4. The answer to the question *who* is some type of *role* played by a person or a computational agent. For Rows 1 and 2, they are persons who play some role in the enterprise. For the lower rows, they may be programs that act for the user at the higher level.
5. The answer to the question *when* is *time*, a subtype such as *date*, or a *time* that is coincident with some event.
6. The answer to the question *why* is some *goal* or subgoal that provides the reason that motivates the model for that row.

Each of these basic concept types is related to other concept types by various relations. Those other types may be included in the graphs specified in that column, but they provide auxiliary information that is subordinate to the basic concept type for the column.

Figure 19 Representing three rows in Column A



Rule 3. *The basic model of each column must be unique.* Since each column provides the answer to a different question, no two columns focus on exactly the same information. Since all columns are related, the graphs in each column may contain concepts and cross references to other columns. But the central concept types in each column are unique.

Rule 4. *Each row represents a distinct, unique perspective.* Since each row presents a perspective on a different model from the point of view of a different role (planner, owner, designer, builder, subcontractor), each row contains different concepts that provide a different level of description. Figure 19 shows three different rows in Column A.

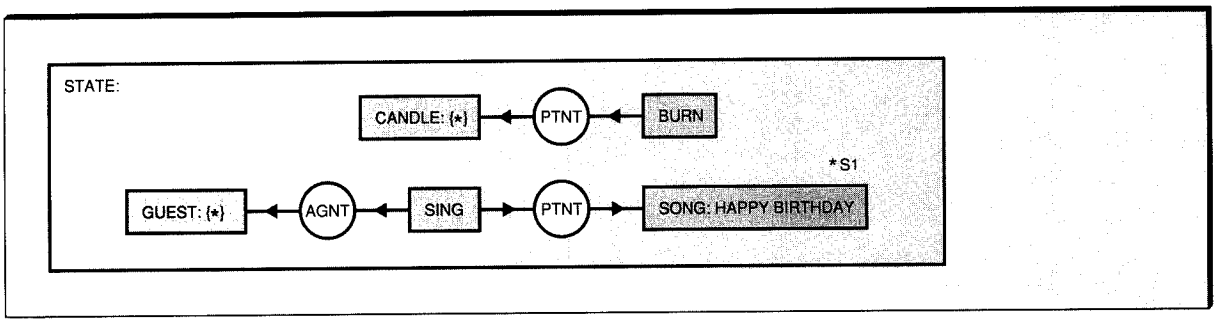
All the concept types in Figure 19 are subtypes of *entity*: CAR, MODEL, SERIAL-NO, MODEL-NAME, and VARIABLE. The types in Row 2, the enterprise model, describe real-world entities, such as cars and models. The types in Row 3, the I/S model, describe logical information types, such as serial numbers and model names. The types in Row 4, the technology model, describe implementation details, such as variables in some programming language. The NAME relation links the entities in Row 2 to the information types in Row 3. The REPR (representation) relation links the information types in Row 3 to the implementation types in Row 4.

Rule 5. *Each cell is unique.* This rule follows from Rules 3 and 4.

Rule 6. *The composite or integration of all cell models in one row constitutes a complete model from the perspective of that row.* When a system is completely specified, all the conceptual graphs in each cell of a given row represent a complete specification of the system at that level. To see the kind of information at each level, read across the rows of the OCRA example in Figure 7.

Rule 7. *The logic is recursive.* The ISA framework is recursive in several different ways. In one sense, it can serve as a metamodel to describe itself: since it is general enough to describe the construction of any system, it can also describe its own construction. As another kind of recursiveness, it can describe entities and states that have parts and subparts nested inside one another to any depth. Figure 16, for example, might represent the process of blowing out the candles on a birthday cake. The state s1 would represent the candles burning for 15 seconds while the guests sing "Happy Birthday." Then event e is the act of blowing out the candles, and state s2 represents the candles smoking for 5 seconds. Figure 20 is an expansion of

Figure 20 Expanded description of state s1 in Figure 16



state s1 to show the nested graphs that describe the candles burning and the guests singing.

In Figure 20, the details of the singing are not described. Even though singing is a process with sound and movement, those details are unimportant at this level of description, and the entire process may be considered a single, unchanging state. If the details of the singing were significant, the box of type SING could be expanded to a process with each note represented as a separate event. On a sheet of paper, it is not possible to show all of the nested levels with equal clarity, but an interactive display would allow the viewer to zoom in or out on any box.

At a larger level, Figure 21 shows the entire birthday party with the process box of Figure 16 nested inside. In the box for the birthday party, the top graph says that 40 guests x are giving presents to a person named Marvin. There are also 50 candles y on a cake. Inside the nested process box, the first state is described by graphs for the candles y burning and the guests singing "Happy Birthday." The next event is described by a graph for Marvin blowing out the candles y. And the last state is described by a graph for the candles generating smoke.

The example of a birthday party illustrates the conceptual graph notation with a familiar situation. But exactly the same techniques could be used to describe a manufacturing process, a courtroom trial, or the steps in the execution of a computer program. For any of these purposes, the subtypes of SITUATION could be described by nests of contexts containing conceptual graphs. Other notations for describing processes and events—flowcharts, state-transition diagrams, data flow diagrams, or Petri nets—could be translated to similar nests of conceptual graphs.

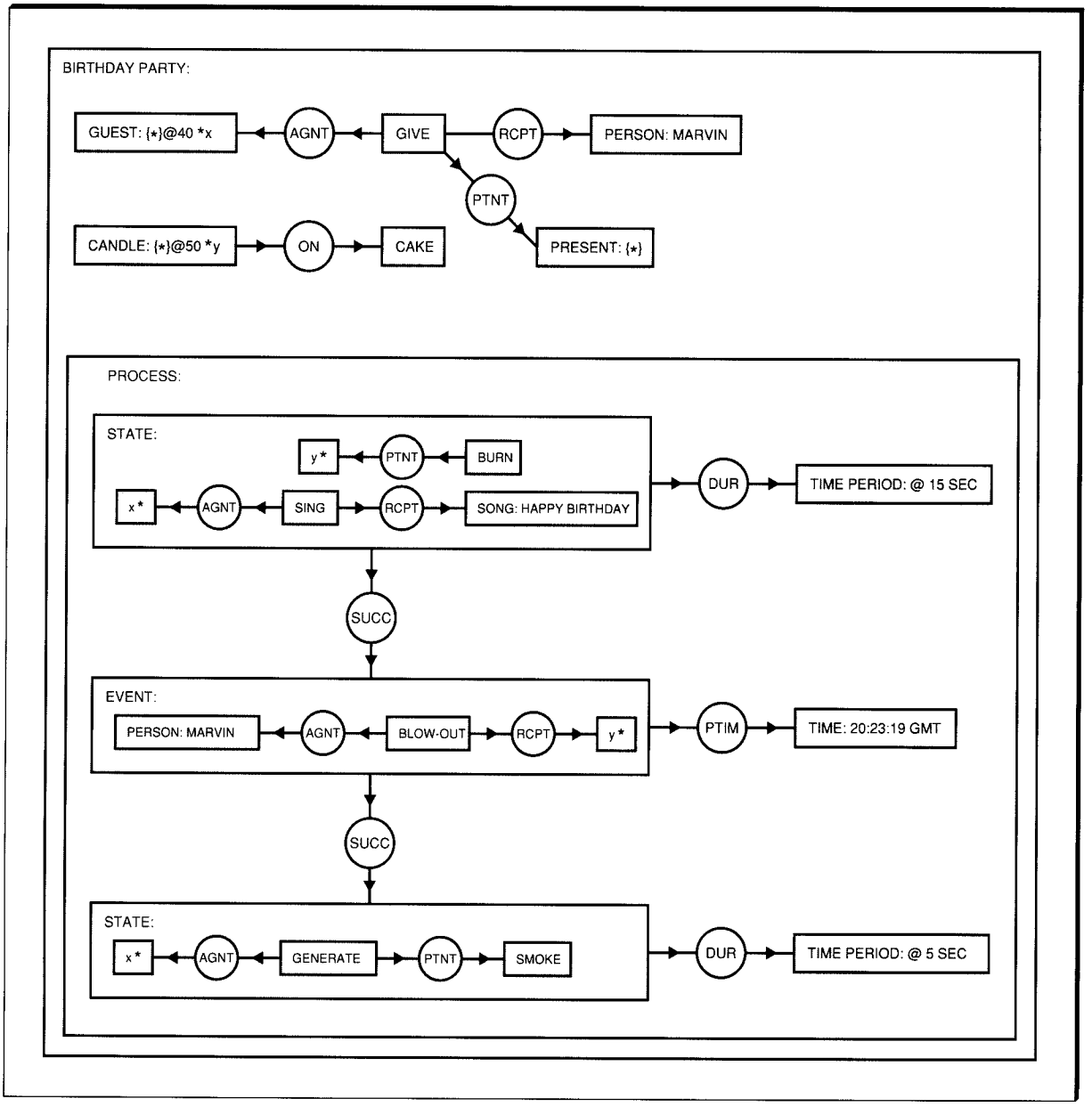
An architecture for the information age

Dramatic improvements in the price-performance of information technology and the escalation of the rate of change show no signs of abatement. In the words of Alvin Toffler, "Knowledge is change . . . , and accelerating knowledge, fueling the great engine of technology, means accelerating change."[19] Gone are the days of computers for simple calculations. We are only now beginning to see the enormous complexity of integrating information technology into the very fabric of our enterprises. Soon, the enterprise of the information age will find itself immobilized if it does not have the ability to tap the information resources within and without its boundaries.

In this scenario, it is little wonder that a framework for information systems architecture finds such widespread applicability. It would be impossible for man or machine to successfully accommodate the complexities of today's enterprise without some kind of logic structure. Every discipline apparently finds a classification scheme or periodic table for organizing knowledge and forming a basis for constructing more complex theses. The ISA framework is a contribution in this regard. It is not so much an invention as it is an observation—an observation of some (apparently) natural rules for segmenting an enterprise into understandable parts without losing the definition of its total integration.

The logic structure or rules of the framework are generic. They can be used for structuring the description of any complex object. The framework was first discovered by observing how the manufacturing discipline segments the descriptions of complex engineering products for the purposes of

Figure 21 Nested contexts for describing a birthday party



design and manufacture. It would appear that the use of the design artifacts are several including:

- Partitioning the design tradeoff decisions into manageable, independent variables
- Ascribing appropriate design formalisms for each variable
- Establishing a baseline of descriptive representations for managing changes in the product during and after its production

These are precisely the same reasons why the ISA framework is interesting for segmenting the descriptions of the enterprise: for separating inde-

pendent variables into understandable, designable components; for developing appropriate design formalisms; and for establishing an enterprise infrastructure in which change can be assimilated in a manageable fashion.

Until recently, these architecture and modeling concepts were somewhat theoretical and merely intellectually entertaining to the practicing data processing professional. Modeling formalisms had evolved and were maturing, but the resultant models were of minimal value since they tended to be of such a high level of generality that they were useless for design purposes, or at such a low level of detail that they could communicate to no one but the person who built them. Furthermore, there was nowhere to put them except on paper, or on large walls. That made it virtually impossible to locate a given design component, search for patterns, change the structure, or keep it current, much less perform configuration management and version control or zoom in and out for communicating to different audiences.

It is only the advent of an automated model storage facility or repository that brings any of this into the realm of feasibility and makes architecture a reality. It does not mean to suggest that all of these ideas will be immediately available in any particular repository product. It only means that they come into the realm of feasibility as repository technology becomes a reality. Even though early repository-type products are nowhere near ready to perform the kinds of services mentioned, the very existence of an automated storage mechanism for models makes it clear that architecture is no longer mere intellectual entertainment. It will become an imperative for any enterprise that intends to be a serious player in the information age.

Acknowledgments

John Zachman is indebted to Steve Pryemybida of Northrop for work relative to the IDEF activity and to Keri Anderson Healy of Model Systems, Michael Eulenberg of the City of Seattle, and Bill Babichuk of Ontario Hydro for work relative to the GUIDE Project on Information Architecture. Their contributions were invaluable in the efforts to define the last three columns. John Sowa would like to thank the members of the AD/Cycle Architecture Group and the ANSI X3H4.6 Task Group for many stimulating discussions in analyzing and representing a variety of I/S issues. Those discussions have helped to clarify the relationships between the levels of the ISA framework and represent them in conceptual graphs.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references and note

1. J. A. Zachman, "A Framework for Information Systems Architecture," *IBM Systems Journal* **26**, No. 3, 276–292 (1987).
2. C. Loosley, "Separation and Integration in the Zachman Framework," *Database Newsletter*, Database Research Group, Boston **20**, No. 1, 3–9 (1992).
3. J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley Publishing Co., Reading, MA (1984).
4. J. F. Sowa, "Towards the Expressive Power of Natural Language" in *Principles of Semantic Networks*, J. F. Sowa, Editor, Morgan Kaufmann Publishers, San Mateo, CA (1991), pp. 157–189.
5. ANSI X3H4.6 Task Group, *Model Unification for Data Repositories*, Technical Report X3H4/92-003, American National Standards Institute, New York (July 1992).
6. V. J. Mercurio, B. F. Meyers, A. M. Nisbet, and G. Radin, "AD/Cycle Strategy and Architecture," *IBM Systems Journal* **29**, No. 2, 170–188 (1990).
7. S. L. Montgomery, *AD/Cycle: IBM's Framework for Application Development and CASE*, Van Nostrand Reinhold, New York (1991).
8. The original article[1] called this the "model of the information system." But the words "information system" imply the technology used for implementation. If the technology consisted of paper and filing cabinets handled by clerks, the design would likely be called a "system" rather than an "information system." Therefore, the label has been generalized to "system model" to avoid excluding any kind of system.
9. T. W. Malone, J. Yates, and R. I. Benjamin, "Electronic Markets and Electronic Hierarchies," *Communications of the ACM* **30**, No. 6, 484–497 (June 1987).
10. T. A. Bruce, *Designing Quality Data Bases*, Dorset House (1991).
11. *Concepts and Terminology for the Conceptual Schema and the Information Base*, J. J. van Griethuysen, Editor, ISO/TC97/SC5—N 695, International Organization for Standardization, Geneva (1987).
12. P. P. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems* **1**, No. 1, 9–36 (1976).
13. A. N. Whitehead and B. Russell, *Principia Mathematica*, 2nd edition, Cambridge University Press, Cambridge (1925).
14. A. Schmidt, "Über deduktive Theorien mit mehreren Sorten von Grunddingen," *Mathematische Annalen* **115**, 485–506 (1938).
15. D. D. Roberts, *The Existential Graphs of Charles S. Peirce*, Mouton, The Hague (1973).

16. L. Tesnière, *Éléments de Syntaxe Structurale*, 2nd edition, Librairie C. Klincksieck, Paris (1965).
17. J. F. Sowa, "Semantic Networks," *Encyclopedia of Artificial Intelligence*, Second Edition, S. C. Shapiro, Editor, Wiley, New York (1992), pp. 1493–1511.
18. J. F. Sowa, "Definitional Mechanisms for Restructuring Knowledge Bases," in *Methodologies for Intelligent Systems, 5*, Z. W. Ras, M. Zemankova, and M. L. Emrich, Editors, North-Holland Publishing Co., New York (1990), pp. 194–211.
19. A. Toffler, *Future Shock*, Random House, New York (1970).

Accepted for publication April 30, 1992.

John F. Sowa *IBM Education Center, 500 Columbus Avenue, Thornwood, New York 10594.* Mr. Sowa is a member of the IBM Systems Research Education Center. After graduating with a B.S. in mathematics from the Massachusetts Institute of Technology in 1962, he joined an applied mathematics group at IBM. Four years later he attended graduate school at Harvard University, earning an M.A. in applied mathematics under the IBM Resident Graduate Study Program. At IBM, he has worked in various areas of computer systems, including compilers, programming languages, and system architecture. Since 1976, he has been doing research and teaching on artificial intelligence and on applications to natural languages, expert systems, and database query. His theory of conceptual graphs has been adopted by a number of research and development groups throughout the world. Recently he has been working on logic-based standards for information interchange with the ANSI X3H4 Committee on IRDS, the ISO Special Group on Conceptual Schemas, and the DARPA-sponsored Knowledge Sharing Effort.

John A. Zachman *Zachman International, Suite 337, 2222 Foothill Boulevard, La Cañada, California 91011.* Having retired from IBM in 1991, Mr. Zachman now operates his own education and consulting business. He joined IBM in 1965 and held various marketing-related positions. He has been focusing on planning and information strategies and architectures since 1970 and has written a number of articles on those subjects. Known not only for his work on information systems architecture, he was also an early contributor to IBM's Business Systems Planning and to the "intensive planning" technique. He travels worldwide, teaching and consulting. Mr. Zachman holds a degree in chemistry from Northwestern University. Prior to working for IBM, he served for a number of years as a line officer in the U.S. Navy and is a retired Commander in the U.S. Naval Reserve. Among his current activities, he serves as a special advisor to the School of Library and Information Management at Emporia State University, is a member of the Advisory Council to the School of Library and Information Management at Rosary College, River Forest, Illinois, and serves on the board of directors for the Repository/AD Cycle Users Group.

Reprint Order No. G321-5488.