Jeffrey G. Long and Dorothy E. Denning

Ultra-Structure: *A Design Theory for Complex Systems and Processes*

**The physicist and Nobel laureate Ilya Prigogine states that "our understanding of nature is undergoing a radical change toward the multiple, the temporal, and the complex. Curiously, the unexpected complexity found in nature has not led to a slowdown in the progress of science, but on the contrary to the emergence of new conceptual structures that now appear as essential to our understanding of the physical world" [11]. We believe the challenges posed by complex systems arise primarily from the use of conceptual structures that worked well for static systems but do not work as well for more dynamic systems. We therefore propose new conceptual structures based on a different metaphysical view of the nature of complex systems.

In this article we use the word *system* to denote a system in the real world (for example, a corporation, a biological system, a legal system), and *application* to denote the representation of a system. *Object* and *entity* are used interchangeably, and may refer to an object, classifications of objects, or relationships between objects.

Current approaches to complexity presume an ontology of objects and relationships, sometimes explicitly including processes and, in science, sometimes explicitly including observers. They represent these in various ways. Objects and their relationships are usually declared in data. Procedural aspects rarely are, and instead are encoded as software with certain branching point values (parameters) possibly defined as data. Semantic, hypersemantic, object-oriented, and functional conceptual data modeling approaches are evolving to represent "business knowledge" such as heuristic rules and temporal characteristics of objects as data [7, 10]. We agree that far more information about a system can usefully be represented in data rather than in software, but believe there is a fundamental need to identify new conceptual structures that are more stable than objects, relationships, rules, or processes and are therefore more appropriate as the basis for representing and comprehending complex systems.

Ultra-Structure is a general theory regarding the improved representation of complex rules. It was originally derived from the linguist Noam Chomsky's work on transformational grammar. In applying Chomsky's theory over the past nine years to businesses and other complex systems, we have substantially modified his theory. Ultra-Structure is based on two key hypotheses:

1) *The Ruleform Hypothesis:* Complex system structures and behaviors are generated by not-necessarily-complex processes; these processes are generated by the animation of operating rules. Operating rules can be grouped into a small number of classes, whose form is prescribed by *ruleforms*. While the operating rules of a system change over time, the ruleforms remain constant. A well-designed collection of ruleforms can anticipate all logically possible operating rules that

might apply to the system and constitutes the *deep structure* of the system.

2) *The CORE Hypothesis:* There exist complex operating rule engines, or *COREs,* consisting of <50 ruleforms, that are sufficient to represent all rules found among systems sharing broad family resemblances, for example, all corporations. Their *definitive deep structure* will be permanent, unchanging, and robust for all members of the family, whose differences in manifest structures and behaviors will be represented entirely as differences in operating rules. The animation procedures for each engine will be relatively simple compared to current applications, requiring less than 100,000 lines of code in a third-generation language.

Ultra-Structure notes that the objects, relationships among objects, and processes that we see manifested by complex systems are evanescent and that even the rules that generate these objects, relationships, and processes are subject to ongoing change. To design for change, we can specify only ruleforms and their associated animation procedures, as only these are stable and fundamental. These can be implemented as relational tables and ordinary software, respectively. Instances of operating rules are defined via relations (records) within those tables, with each record defining one rule or part of one rule. Representing the operating rules of a system as data leads to a different and more maintainable structuring of the information: an application can be reconfigured by changing data (the records of the tables) rather than changing the software or table structures. And only a relatively small number of animation procedures are required.

Ultra-Structure has been applied to create several applications in real business organizations. Order entry, inventory, billing, cash application, and other applications have been implemented and maintained for years, and their metrics—both financial and computational—appear to be quite favorable compared to current approaches. It appears that applications that would normally require tens of thousands of records and several million lines of code can be implemented with tens of thousands of records and tens of thousands of lines of relatively unchanging code.

In addition to presenting the basic concepts of Ultra-Structure, this article illustrates how a moderately thorny business requirement, namely customer billing, can be represented as ruleforms and animation procedures. However, it is beyond the scope of this article to describe the complete deep structure of business or any other system.

Ultra-Structure Overview

The principal goal of an Ultra-Structure analysis is to construct a "rule engine" consisting of a small set of invariant data structures and software. We call such a machine a *Complex Operating Rule Engine (CORE).* The key word here is "invariant": Ultra-Structure aims to replace the standard software lifecycle requiring development, maintenance, and application replacement with a design-for-change approach that focuses on data maintenance. As James Hager notes in an article reviewing the Navy's Software Cost Reduction program, "Unfortunately, current methodologies for specifying, designing, documenting, coding, and testing software do not provide adequate visibility to maintenance concerns. The difficulty of generating software that is easily modified becomes evident when the full engineering life cycle costs are examined" [4].

Experience to date indicates that each class of systems having broad family resemblances—e.g., all games, all scientific arguments, all laws, all corporations—will require a separate engine, reflecting its distinct deep structure and animation procedures. The proposed engine for business is called the CORE/650 (650 is the Dewey Decimal classification for management). We would expect such engines to be judged by the usual criteria, such as functionality, maintainability, flexibility, user-friendliness, response time, and initial and lifecycle cost.

In 1985, the Office of Technology Assessment reported: "Modern computer/communications systems are among the most complex technologies ever assembled by human beings . . . . To be able to understand, predict, and control the behavior of these technologies requires a powerful theory of complex processes. No such theory yet exists, although it remains a major goal of computer science" [12]. Two years later, Fred Brooks wrote, "If, as I believe, the conceptual structures we build today are too complicated to be specified accurately in advance, and too complex to be built faultlessly, then we must take a radically different approach." [2] He stated his belief that current approaches have reached their natural limits and are not likely to create order-of-magnitude increases in software productivity. We must therefore consider the possibility that our existing design approaches (including entity-relationship, object-oriented, and relational models), useful as they are within certain domains, are somehow missing or even misrepresenting the essence of complex systems.

The fundamental problem may be that we look at complex systems using ontological assumptions that work only with simple systems. In particular, we in Western society all seem to share the substantialist presumption, first enunciated in ancient Greece and built deeply into our language, that systems are comprised of (can be reduced to) objects and that properties can be predicated of those objects. More recent process-oriented thinkers such as Alfred North Whitehead and Ilya Prigogine emphasize a greater focus on relationships between the objects, including the observer as an object in the system. Current analytical approaches derive from these widely held and often unconscious assumptions, which constitute our paradigm or way of looking at the world. Should our basic metaphysical presuppositions turn out to be mistaken or true only in limited cases, then the results we achieve will also fall short. We will, in fact, hit a perplexity barrier—until we consciously define new and more valid premises.

Key Tenets and Definitions

Ultra-Structure is based on the fol-

lowing key premises:

- complex systems are comprised of many objects (including observers) and their relationships;
- these objects and their relationships are the evanescent result of the execution of processes;
- these processes, however simple or complex, are the result of executing a series of rules; and
- these rules can and often do change, but they change within certain limits.

While unnecessary neologisms are undesirable, there is no uniform terminology or framework for discussing a broad range of systems. Expressing these ideas clearly required us to carefully select the following terminology to reinforce key distinctions:

- The manifest structure and behavior of any system is composed of a set of *particulars*, with these particulars comprising the *surface structure* of the system. Particulars include objects, relationships, and processes of a system and their classifications or generalizations.
- All particulars, however complex, seemingly incomprehensible, or riddled with exceptions, can be described by a relatively small number of *operating rules*. These rules comprise the *middle structure* of the system and permit a substantial compression of information.
- While operating rules may change over time, there are empirical limits within which they do so. Constraints on rules are defined in terms of their form by *ruleforms*. Any actual rules are defined as the content of these ruleforms. A small, simple, and invariant collection of ruleforms comprises the *deep structure* of a system and permits a further substantial compression of information. Ruleforms usually operate in *clusters* that generate, describe, and constrain particulars.
- Ruleforms are constructed from a *substructure* of *universals*. Universals, like ruleforms, are forms; the content of universals is symbols, such as letters and numbers.
- The complex behavior and structure seen by the observer of a system is simply the consequence of the execution or *animation* of the operating rules. In this worldview, undetectable animation procedures generate real-world surface structure from undetectable middle structure.

Ultra-Structure models can be implemented within the constraints of the RM/V2 relational data model advocated by Codd [3]. But Ultra-Structure offers a different and useful semantic interpretation of the meaning or purpose of the relations defined in a relational database: relations do not represent objects, relationships, classifications, procedural aspects, or anything else: they represent only rules.

Table 1 shows some of the ontological levels proposed by Ultra-Structure and how these levels relate to current approaches. Limiting our discussion for the balance of this article to business as an example of complex systems, we discuss these ontological levels and their representation in an Ultra-Structure model for billing.

Surface Structure

The surface structure of a system is its manifest structure and behavior. In a business this includes everything that employees do all day, everything that can be physically seen, and business events, such as issuance of customer orders, purchase orders, manufacturing orders, paychecks, shipping orders, and invoices. It also includes such intangibles as work processes, market segmentation, and organizational structure.

Surface structure is complex and ever-changing. New products are introduced, employees are reorganized, prices change, new merchandising programs are introduced, branch offices are opened or closed, work processes are modified or changed wholesale, subsidiaries are acquired or sold, and so on. A primary goal of Ultra-Structure is to anticipate and facilitate this evolving surface structure, even though no one can predict what that surface structure might look like in the future.

In an Ultra-Structure application, some aspects of surface structure must be recorded in order to trigger or properly execute the operating rules. Such information is represented as an attribute of an operating rule, for example, a "quantity-on-hand" attribute of a product sourcing (inventory) rule. Such information is called a *consideration*, for it affects the subsequent execution of rules.

Middle Structure

Ultra-Structure does not focus on surface structure per se, but instead studies the operating rules that generate, account for, and constrain that surface structure, as presented by end users in small joint design sessions. These operating rules comprise the middle structure of the system. Middle structure is far simpler and more stable than surface structure because it abstracts and generalizes surface structure; the laws of science constitute one (mathematical) expression of the middle structure of nature. By moving from particulars to general laws we achieve an enormous economy of expression: ideally we achieve simplicity without simplification, a sort of dehydrated reality that can be regenerated at will by just adding water.

An operating rule in Ultra-Structure has one or more *factors* and one or more *considerations*, both of which are attribute values. The factors determine whether the rule and its considerations will be inspected by the animation procedures and possibly executed. In many cases, the mere existence of a rule permits certain things to happen without any inspection of considerations. Once a rule has been selected for inspection, the considerations are used to determine in the context of other rules whether that rule will be executed. The considerations are often used as foreign keys into other tables, whereby other considerations are inspected.

Abstractly, a rule with, say, three factors *A*, *B*, and *C* and three considerations *X*, *Y*, and *Z* is a conditional statement that is interpreted as follows:

if A **and** B **and** C **then consider** X, Y, Z.

The factors of a rule are always ANDed together; logical OR is represented by multiple rules; and each record is unique. Unlike AI production systems, selecting a rule does not

Table 1. Ultra-Structure Ontological Levels

Ontological Level	Real-World System	Ultra-Structure	Current
Surface Structure	Particulars (Manifest Behavior and Structure)	Rule Considerations, and Application Behavior	Data Structures and Data/Objects (in the OO sense)
Middle Structure	Operating Rules	Rule Records (relations)	Software/Methods
Deep Structure	Ruleforms	Data Structures (tables)	(No equivalent concept)
Sub-Structure	Universals	Attributes	(No equivalent concept)
Animation Procedures	Energy Imbalances; People and Machines	Software	(No equivalent concept)

mean that it will be fired, only that it will be inspected. The following simple pricing rule states that EXEMPT agencies may purchase ABC486 products for $1,500 each during calendar year 1994:

if (PRODUCT = ABC486) **and** (AGENCY = EXEMPT)
then consider
(UNIT_PRICE = 1500),
(UNIT_MEASURE = EACH),
(TIME_PERIOD = CY_1994).

Rules can be either positive (under these conditions, do this) or negative (under these conditions, don't do this). Almost all rules, such as the preceding rule, are positive. The ratio of positive and negative rules relates to how often more specific rules need to undo more general ones, for example, "all customers except EXEMPTs get a surcharge." Negative rules can be implemented as a series of positive rules, but creating a consideration to indicate the negation of a rule enables users to avoid enumerating positive rules for all but the exception cases.

There are far fewer operating rules for systems than one might expect, because most operating rules are defined for large classes (e.g., all tax-exempt customers, all external customers, all employees) rather than at the level of individual objects (e.g., customer #101).

Rules are implemented as relations (records), consisting of at least one factor and one consideration. Table 2 shows a simple PRICING ruleform and three operating rules. The first row in this and all subsequent tables defines the ruleform and gives the attribute names (in bold); the rows that follow give instances of operating rules, that is, relations. A vertical bold line separates the factors (primary indexed key field(s)) of the table from its considerations. An asterisk (*) following an attribute name indicates that the attribute is a foreign key; that is, a reference (key) to a record in another table.

The first rule in Table 2, where AGENCY says "(ANY)," gives the standard price for any customer. The second, where AGENCY = EXEMPT, gives a special price for non-profit organizations, and corresponds to the example given previously. The third rule gives a price for a specific customer, namely GU (Georgetown University), for the first quarter of 1994.

An Example: Process Model for Pricing Rules

The rules of a business are traditionally implemented in two ways:

- as code to be executed by its computers, and
- as policies and procedures to be carried out by employees in conjunction with—or in spite of—the rules carried out by the computers.

While modeling efforts based on substantialist assumptions have been successful at representing objects and relationships, they have been less successful thus far in representing work processes and other kinds of rules as data. We introduce a simplified but robust general work process model for a business to illustrate these concepts and the others that follow as they apply to pricing. Pricing may not seem to be much of a challenge, but in most businesses it is a headache. Tables 6 through 17, which contain the ruleforms for this example, can be found in the appendix.

When an order is placed, various work steps must be initiated to process the order, such as performing a credit check, picking a product from inventory, shipping it, and billing for it based on a fee plus taxes and discounts. These steps constitute a *work protocol*, and are defined by WORK PROTOCOL rules (Table 15). An animation procedure selects the appropriate rules from the table using WORK METARULES (Table 16) to generate "masks" that specify the selection criteria (factors). The selected rules are then placed in a WORKORDERS table (Table 17), where they are picked up and executed by other agencies (e.g., employees, subcontractors, other computers). A more detailed description of the model and examples will be given as the concepts are introduced.

Rule Types. Decision tables (DTs), decision trees, and production rules have been shown to be intertranslatable [9], and representing rules as relations may remind us of these approaches. DTs specify rules where, for a list of all possible input conditions, actions are listed that should be taken in the case of each combination of conditions that may occur. One may think of a DT as a ruleform having factors, which list all possible input conditions and permutations thereof, and considerations, which define all possible actions and

permutations thereof; in other words, condition-action rules. But this obscures the critical semantic differences among various classes of rules. There are a number of more technical differences as well, having to do both with how DTs are implemented and with how they are used:

- DTs link conditions and actions by means of binary (Y/N) values.
- DTs aim to represent all possible factors (completeness) and to represent each combination of possible factors only once (exclusivity). Ultra-Structure models approach this differently, allowing in the former case general catchall rules and in the latter case multiple proposed actions that are then evaluated by metarules.
- DTs are used differently, primarily acting to externalize certain branch points in software while leaving the bulk of rules still hard-coded in software.

Ultra-Structure proposes two basic types of operating rules: existential and compound.

Existential rules. For any table with one field as its primary key, each record in that table defines an *existential rule*. Such rules declare what entities exist, specify unique identifiers for these entities, and provide certain considerations regarding those entities for use in subsequent rule inspection. Existential rules declare not just obvious concrete objects (including activities) but also groupings or classifications of those objects, and relationships among the objects. Examples of existential rules are:

- GU exists as an agency.
- There is a class of agencies called EXEMPT.
- There is an activity called CHK_CRD.
- There is an activity called PRT_CUST_DEC.
- There is a relationship called IS_WITHIN_CLASS.

All of these entities are equally "real" from the point of view of rulemaking, in the sense that a rule that operates on a class or a relationship is just as real as a rule that operates on an individual object.

Tables 6–8 show existential ruleforms and rules for agencies, products, and locations; and Table 17 shows an existential ruleform and rules for work. The AGENCY ruleform has one factor (AGENCY) and one consideration (NAME). The rules define two legitimate objects that exist as customers (GU and ORG_A), four classifications of customers (EXEMPT, EXT_CUST, MNFR, and NON_EXMPT), three internal "departments" (CPU, CREDIT, and BILLING_CPU), and a wildcard (represented by (ANY)) that can refer to any customer. CPU and BILLING_CPU, although invoking computer functions, are regarded as agencies that perform work in the same way as a human-run agency (such as the credit department).

Table 6 does not indicate how any specific customer (e.g., GU) is to be classified (e.g., EXEMPT); those rules are defined separately in Table 10, which will be discussed later. The AGENCY ruleform operationally includes external agencies (e.g., customers, buying groups), internal agencies (e.g., employees, departments, divisions), and other third parties (e.g., taxing bodies, customs regulators, independent salespeople, field engineers).

The PRODUCT ruleform in Table 7 includes tangible goods (ABC486 and CHEM_B), classifications of these goods (FROZEN, ROOM_TEMP, and NON_EXEMPT), and services (e.g., CHK_CRD, FEE, PICK, and STAX). It operationally includes finished goods, components of finished goods (parts), work in progress, raw materials, services, warranties, freight costs, and sales tax collection services.

The LOCATIONS ruleform in Table 8 shows a variety of different types of object that refer to places: streets, cities, countries, areas, factories, and bins. Although the current practice is to treat STREET, CITY, STATE, and ZIP as separate attributes of a single location within a single record, Ultra-Structure treats them as a semantically repeating group, along with bins, telephone numbers, fax numbers, Internet addresses, and other means for contacting agencies under various conditions. LOCATIONS is thus not limited to geographic locations but is the basis of contact rules.

The different types of relationships that can be formed are also regarded as objects and are represented by an existential ruleform, as illustrated in Table 9. The AREA object, for example, is used for associating locations with larger areas. Similarly, the STAX_TYPE object is used for associating agencies with their sales tax classification (EXEMPT or NON_EXEMPT). Ultra-Structure defines relationship types to be objects in themselves that are just as "real" for rulemaking purposes as physical objects. The relationships among objects are specified through network rules, a special type of compound rule (explained next).

Compound rules. For any table with more than one field as its primary key, each record in that table defines a *compound rule*. Conversely, a compound rule is a rule that has more than one factor. It does not declare an entity to exist, as existential rules do; instead, it relates entities previously declared by existential rules. These rules have multiattribute keys, which are foreign keys pointing to the existential rules. Most operating rules seem to work with two to four

Table 2. PRICING Ruleform and Rules

<- Ruleform ->

<- Factors -> <- Considerations ->

PRODUCT*	AGENCY*	UNIT_ PRICE	UNIT_ MEASURE	TIME_ PERIOD	
ABC486	(ANY)	1750.00	EACH	CY_1994	Rule 1
ABC486	EXEMPT	1500.00	EACH	CY_1994	Rule 2
ABC486	GU	1250.00	EACH	Q1_1994	Rule 3

factors, but some of the most important (including process definition) have been found to require as many as fourteen factors. Tables 10–16 illustrate compound rules.

There are four types of compound rules:

- Network,
- Authorization,
- Protocol, and
- Metarule.

Network rules correspond roughly to the ordering activity that some biologists call "classification" or "division," while authorization rules correspond to ordering by "systematization" or "arrangement." Philosopher Robert O'Hare states, "Classification is the grouping of objects into classes on the basis of the properties of the objects being classified. The objects in each group are spoken of as being members or instances of their class. In contrast, systematization is the arrangement of objects into some larger whole object, and the relation that the constituent objects bear to the systematic whole is that of a connected part rather than a member" [8]. Protocol rules define processes, and metarules are rules about reading and evaluating other rules; these will be further explained later.

Network rules. Hierarchical, one-parent-per-child relationships (i.e., trees) fail to represent the potential richness of relationships among entities. *Network rules* permit definition of true networks—not just trees—which define how objects declared within a single existential ruleform relate to each other in a variety of ways. They declare relationships of the form "Class A, with respect to relationship type R, is a member of Class B," where A and R are factors and B is a consideration. If more than one instance of a type of relationship is possible, or if relationships vary as a function of time or some other factor, then the network ruleform must be broadened to include those factors.

Tables 10–12 illustrate network rules for agencies, products, and locations. Each of these tables has a RELCODE attribute, which names the relationship type and is a foreign key into Table 9. For example, the first rule in Table 10 says that GU, with respect to the relationship type CUST_TYPE, belongs to the class EXT_CUST.

Although it is current practice to implement classifications of an object as attributes of that object, that approach does not facilitate changing the type or number of ways in which objects are classified. This is particularly important because the transition of an object from one class to another—thereby bringing it under wholly different rules—gives complex systems much of their seemingly emergent and/or random behavior. Furthermore, that approach is misleading from a semantic perspective, as it implies objects and their classifications are fundamentally different; from the perspective of a rulemaker they are not different.

Unless specified otherwise in a network, objects inherit the relationships of their parents. Thus, if A belongs to B with respect to R1, and B belongs to C with respect to R2, then A also belongs to C with respect to R2. For example, in the LOCATION NETWORK of Table 12, WASH[ington] inherits EAST_COAST with respect to REGION and US with respect to AREA; the more specific address 225RSB inherits these plus DC with respect to STATE.

Authorization rules define what inputs are allowable, authorized, or expected. The existence of a matching record for factors A, B, and C indicates they are an input combination (i.e., a *situation*) for which specific behavior has been defined; now consider X, Y, and Z. Tables 13 and 14 illustrate authorization rules.

The SOURCING AUTHORIZATIONS rules in Table 13 declare what locations each product may exist in, and who may order each product or service. It has as considerations (non-key attributes) quantity on hand, quantity committed, and reorder point. By including a generalized location (LOCN) as a factor, the ruleform permits the definition of rules for products that are at a customer site, a field engineer site, a third-party warehouse, or even a competitor's site. AGENCY indicates the owner (normally the seller whom the application is supporting), permitting the ruleform to include products on consignment, installed equipment that has already been sold or is on lease or loan, and so forth.

The SEQUENCING AUTHORIZATIONS rules in Table 14 impose sequencing constraints on work processes. For example, credit checking precedes inventory picking if both are selected from the WORK PROTOCOL table by the animation procedure that handles customer orders. Sequencing constraints are interpreted as precedent relations rather than as prerequisites; thus, for example, PRT_CUST_DEC does not need to take place in order for SHIP to be executed. The SEQ_NO factor indicates whether CHILD_PROD is the first, second, etc., child of PARENT_PROD. (The ordering of children is not, however, significant.)

The existence of a factor in a ruleform does not mean that factor must be used by a particular rule or even by any rule. The factor may state "(ANY)," meaning it is a *wildcard* and any value is acceptable. For example, AGENCY = (ANY) in rule 1 of Table 2, which gives a default price for any agency, and several fields are defined as (ANY) in Table 15.

Differences in structure or behavior among systems having family resemblances (e.g., different corporations or departments within a corporation) are caused primarily by differences in operating rules. Further, each system may use only a subset of the factors available to it; thus, a business may have a factor that is initially (ANY) and later change its operating rules to include other values for that factor, thereby bringing that factor into play in determining what rules will be selected. And different companies may have similar authorization rules but use different ways of categorizing input, that is, different networking rules that map customers into different customer categories. All such different behaviors can be accommodated without the need to restructure ruleforms or animation procedures; one may simply add or modify records in various existing tables.

Protocol rules define the contents of a work process. The WORK PROTOCOL shown in Table 15 specifies the work steps that may be triggered

when an explicit order is placed (how they are triggered is discussed in the forthcoming section on animation procedures). The table has six factors and four considerations. In more advanced versions of the CORE/650, this table has as many as fourteen factors and is intended to define all of the work processes of an organization.

The AGENCY factor indicates the agency by whom the explicit request was placed, while the ASSIGN_TO consideration specifies the internal agency that will handle the work process. PROD1 indicates the request type (e.g., DELIVER); PROD2 indicates the product ordered; PROD3 indicates the service to be performed; PROD4 indicates the product that is the recipient of the service (often the same as PROD2); and QTY indicates the amount of the product to be serviced. The SEQ_NO attribute is used to ensure uniqueness of the primary key. A ditto (") in a record field means "substitute the value specified in the work request." The sample operating rules shown in the table are interpreted as follows:

1) If any external customer (AGENCY = EXT_CUST) wants delivery (PROD1 = DELIV) of any product (PROD2 = (ANY)) from any US warehouse (SHIP_FROM = US) to anywhere in the US (SHIP_TO = US), then generate a work order for the computer (ASSIGN_TO = CPU) to check his or her credit (PROD3 = CHK_CRD).
2) Same as 1), but if the SHIP_TO location is Europe, generate a work order for the people in the Credit Department (ASSIGN_TO = CREDIT) to check their letter of credit (PROD3 = CHK_LTR).
3) If any external customer wants delivery of an ABC486 product from any U.S. warehouse to any European location, then generate a work order to the computer to print a customs declaration (PROD3 = PRT_CUST_DEC).
4) If any agency, whether internal or external (AGENCY = (ANY)) wants delivery of any product (PROD2 = (ANY)) from any warehouse (SHIP_FROM = (ANY)) to any location (SHIP_TO = (ANY)), then generate a work order to the default factory (ASSIGN_TO = FACTORY1) to pick (PROD3 = PICK) the specific product they requested (PROD4 = ") and the specific quantity they requested (QTY = ").
5) Same as 4), but generate a work order to ship (PROD3 = SHIP).
6) If any external customer wants delivery of an ABC486 product from any U.S. warehouse to any location (SHIP_TO = (ANY)), then generate a work order to the billing computer (ASSIGN_TO = BILLING_CPU) to compute a fee (PROD3 = FEE) with a unit price of $1500.00 (QTY = 1500.00).
7) If any nonexempt agency (AGENCY = NON_EXEMPT) wants delivery of a nonexempt product (PROD2 = NON_EXEMPT) from any location to any D.C. location, then generate a work order to the billing computer to compute a sales tax (PROD3 = STAX) with a tax rate of 5.75% (QTY = 0.0575).
8) If any external customer wants delivery of an ABC486 product from any U.S. warehouse to any European location, then generate a work order to the billing computer to compute a sales discount (PROD3 = DISCNT) with a discount rate of 5% (QTY = 0.05).
9) If Georgetown University (AGENCY = GU) wants delivery of an ABC486 product from a U.S. warehouse to one of its D.C. campuses, then generate a work order to the billing computer to compute a fee (PROD3 = FEE) with a unit price of $1.250.00 (QTY = 1250.00).

Metarules are rules for reading rules. They specify a sequence of RELCODE transformations on keys to produce various *masks* in a specified order. A mask is a set of values for the factors of a rule, that is, the key attributes in the table. The animation procedures use the masks as patterns for matching operating rules; the rules that match are then selected for consideration and possible execution. Metarules rules may be embedded in the animation procedures, but in cases where such rules are particularly complex or changeable, they are better defined explicitly as data.

Table 16 shows WORK METARULES that are used with the WORK PROTOCOL rules in Table 15. The attributes PROD1, AGENCY, PROD2, SHIP_TO, and SHIP_FROM correspond to the primary key attributes in Table 15. Note, however, that only PROD1, which specifies the request type, is part of the primary key in Table 16. The SEQ_NO factor, which forms the other half of the primary key, is used to pick rules in sequence; thus, the rules are picked in the order in which they are listed. The rules all apply to external sales orders for a delivery (PROD1 = DELIV) and are interpreted as follows:

1) Create a mask by generalizing the SHIP_TO based on RELCODE = STATE and SHIP_FROM based on RELCODE = AREA. AGENCY and PROD2 must match exactly (denoted by "SAME").
2) Create a mask by generalizing AGENCY based on RELCODE = CUST_TYPE and SHIP_TO and SHIP_FROM based on RELCODE = AREA.
3) Same as 2), but the mask should match any product (PROD2 = (ANY)).
4) Same as 2), but the mask should match any SHIP_TO location (SHIP_TO = (ANY)).
5) Create a mask by generalizing AGENCY and PROD2 based on RELCODE = STAX_TYPE and SHIP_TO based on RELCODE = STATE; SHIP_FROM should match anything.
6) Create a mask that matches anything.

Deep Structure

We presume that the operating rules (laws) of natural systems remain unchanged over time, but this presumption clearly does not work for intentional systems where the rules can and do change frequently. The Ruleform Hypothesis states that we may abstract rules into broad classes based on their form. Ruleforms are to collections of rules what abstract numbers are to collections of things. They are the result of a process of ignoring everything in the set except one feature of the members of the set. Ruleforms abstract morphology, while numbers abstract quantity; in a sense, ruleforms model the geometry of rules, and letters abstract pho-

nemes. Although Ultra-Structure postulates that it is possible to find ruleforms that are invariant, the benefits of Ultra-Structure can be obtained as long as the ruleforms seldom change.

Designers acquiescing to initial end-user perceptions tend to see each class of objects as fundamentally different and therefore as requiring its own separate ruleform (table). Thus, there may be separate tables to specify each customer's tax status, pricing status, sales territory, etc. Only careful examination of the use of an entity will determine whether it is really the same as or different from an existing class of entities. Thus, "customers" may ultimately be grouped with "employees" and "vendors" under the rubric of AGENCIES, since it turns out they are all subject to similar rules. But the class called AGENCIES is not likely to include geographic locations, which are subject to different rules. However, "telephone numbers," "fax numbers," and "email addresses" may ultimately be grouped with geographic locations, as these designate rules for contacting an agency.

This form of semantic analysis is central to identifying the invariant classes of entity that form the basis for modeling a system. Ultra-Structure therefore asks designers to transcend both the surface structure of natural language and the surface structure of a system's own history, with their often false distinctions, to determine operationally (i.e., by usage) whether an entity is subject to the same types of rules as another entity and should therefore be defined in the same ruleform.

Rarely is a single ruleform adequate to prescribe a behavior; typically a group or *cluster* of ruleforms is used by an animation procedure. The ruleforms in Tables 6–16, for example, form a cluster for the work process animation procedure, but the individual ruleforms within the cluster are also used in several other clusters for completely different purposes (e.g., compute price quotation, check inventory, lookup customer). We have found that in practice, the animation procedures must often inspect four to ten ruleforms before making a decision. A frequent error of new designers is to attempt to define rules by a single ruleform rather than via a cluster of ruleforms; even such a simple rule as "three strikes and you're out" requires a cluster of ruleforms to implement the generalized capability in a game of determining when a player's turn is over.

The term *deep structure* refers to the collection of all ruleforms and their interactions in a specific application. Deep structure is remarkably simple and offers an enormous new economy of expression.

The "correctness" of a deep structure is not a simple yes or no decision; there are degrees of *breadth* of a deep structure. The CORE Hypothesis postulates that with enough experience in searching for counter-instances, one can discover a definitive deep structure, which is, by definition, permanent, unchanging, and robust in the face of any middle structure it may be called on to represent for that family of systems.

By going from surface structure to deep structure we move from how a system appears to what it *is*: We move away from phenomenology to glimpse the ontology of a system. This will be a crucial tool in helping us gain a better understanding of the fundamental nature of business, of games, of laws, and of other systems.

Substructure

Ruleforms contain rules, but ruleforms are not the atomic units out of which complex systems and processes are built; rather, they themselves consist of relationships of universals. Universals collectively constitute the substructure of a system. Each universal is a distinguishable facet of a system. Work thus far suggests that about 400 universals will be required to define a complete business system, e.g., any corporation.

As rows (records) are the horizontal division of behaviors into rules, columns (attributes) are the vertical division of behaviors into their fundamental domains (universals), such as locations, descriptions, time periods, and weights. Each constitutes an attribute (factor or consideration) from which ruleforms, and thereby rules, are constructed. They are the domains within which real-world systems may be modeled. Contrary to conventional philosophical use of the term, "redness" would not be an example of an Ultra-Structure universal because redness is the value of the universal, not the universal itself; rather, "color" might be an example, or "weight."

Looking at the "real meaning" of each universal in an application frequently leads a designer to observe that many universals with different names actually describe the same underlying concept. In Ultra-Structure, such cases are interpreted as a *semantic repeating group*. Ultra-Structure does not permit ruleforms to contain such groups; instead, they are replaced by a single, more inclusive universal.

As an example, Table 3 contains a general ledger table with attributes for each month. Ultra-Structure interprets the months as a semantic repeating group of the universal "time period" (TIME_PER). Table 4 shows the effect of replacing the group with the universal.

Identifying and eliminating semantically equivalent universals is critical to designing ruleforms that are invariant. Yet such analysis smacks of pure philosophizing: how is a product different from a service? How is posting to a GL account like fixing a machine at a customer site? How is a vendor different from an employee? How is a bank account like a drill press? End users often have little patience for such inquiries until they see the power, breadth, and even truth given to them by the resulting model.

Whether attributes are the same or not cannot be determined solely by their common-sense or current interpretation; instead, the designer must examine their operational definitions, i.e., how they are used, regardless of initial appearances. This is very similar to the process of determining whether entities defined in existential rules are fundamentally the same or not. In that case, if the ruleforms the entities participate in are similar, then the entities are fundamentally similar (e.g., employees and vendors are both agencies be-

cause they are both subject to the same types of constraints). Likewise, in this case, if the attributes (e.g., JAN, FEB, MAR) are subject to the same kinds of operations, then they are fundamentally similar and should be treated as such. The fact that a company now uses a 12-period accounting cycle is surface structure; it could at any time go to a 13-period cycle, or an *n*-period cycle!

Animation Procedures

Animation procedures define and implement the generative process whereby behavior is produced according to the operating rules defined in a ruleform cluster. As with ruleforms, animation procedures are simple and small; experience to date indicates that no CORE will require more than 100,000 lines of 3GL code to implement its animation procedures.

Animation procedures inspect the operating rules, determining which operating rules are looked at in what order and what to do in the face of *collisions* (selecting among competing rules), and performing *compression* (eliminating redundant rules that apply). An animation procedure for a protocol rule (e.g., Table 15) takes an input (e.g., the customer order for a delivery) and uses metarules (e.g., Table 16) to translate the input into masks. The masks are then used as patterns on the protocol rules to select candidate rules for inspection. Some of the selected rules may be eliminated by collision or compression; the remainder will be written to the WORKORDERS table for some agency (either people or computers) to process.

If the mask formed directly from a stimulus or input fails to match any rule, then it usually is successively generalized until a match occurs or the metaprotocol is exhausted. This generalization is done using network rules. For example, if a mask with ORG_A for AGENCY1 fails to match any of the rules, then there may be a metarule that says to use CUST_TYPE to generalize, whereby the software would then find EXT_CUST using rule 3 in the AGENCY NETWORK rules of Table 10.

Table 3. GENERAL LEDGER

GL_ACCT	JAN	FEB	MAR	APR . . .
123-45-678	350.25	600.15	27.50	12.75
123-45-679	150.00	450.00	100.25	19.50
123-46-000	111.90	350.00	75.00	125.00

Table 4. GENERAL LEDGER with New Factor

GL_ACCT	TIME_PER*	QTY
123-45-678	JAN_94	350.25
123-45-678	FEB_94	600.15
123-45-678	MAR_94	27.50

Metarules are usually constructed so that operating rules are read in the following order: specific rules first; increasingly generalized rules next (based on specific RELCODES); and wildcards last.

The following summarizes the animation procedure for the rule cluster consisting of Tables 6–16. The example illustrates the use of several masks during rule selection.

1) Read an incoming event, in this case a customer order for delivery. The event will specify five attributes: AGENCY, PROD1, PROD2, SHIP_TO, and QTY, where AGENCY is the agency placing the order, PROD1 = DELIV, PROD2 is the product to be delivered, SHIP_TO is the ship-to location, and QTY is the quantity ordered.

2) Ensure that the product and quantity specified are available by creating masks of PROD (using PROD2) and (ANY), then key-sequentially reading the SOURCING AUTHORIZATIONS (Table 13). If no record is found for that product, or if the computed quantity available is inadequate, notify the user; otherwise, determine its warehouse location (SHIP_FROM) using LOCATIONS (Table 8) and LOCATIONS NETWORK (Table 12).

3) To determine what WORK PROTOCOL rules to apply, create a mask with values for PROD1, AGENCY, PROD2, SHIP_TO, and SHIP_FROM by selecting the first record in the WORK METARULES (Table 16) and performing the specified translation on the data obtained in Steps 1 and 2. "(SAME)" means do not translate the input but use it as entered; "(ANY)" means ignore the input and utilize (ANY) in the mask; and any other value signifies a RELCODE used to navigate the appropriate network table, obtain a translation of the input, and then use that translation in the mask. The mask specifies patterns for the factors in Table 15. It is concatenated with values for the considerations, where ASSIGN_TO is set to the warehouse location for the product as found in Step 2; PROD4 is set to the original (untranslated) value of PROD2 (the product ordered); and QTY is used as given.

4) Select matching records from the WORK PROTOCOL (Table 15). "(SAME)" as a consideration means use the value as it was set in step 3; otherwise use the constant in Table 15.

5) Read the next record of the WORK METARULES PROTOCOL (Table 16) to determine the next translation rule for the input data, in order to create the next mask (the considerations are not altered). Repeat Steps 4 and 5 until all translations have been performed.

As work protocol records are found, the animation procedure determines whether they are to be executed; if so, it writes them to the WORKORDERS table (Table 17). Assuming one order from Georgetown University for delivery of 2 ABC486 products to 225RSB, the generated work protocol would include the records shown in Table 17, records #1-0 through 1-4. Assuming

another order from ORG_A for Paris delivery of two ABC486 products, the generated work protocol would include records #2-0 through 2-6 of Table 17. In a real application 25–30 such records might be generated for a product sale to an external domestic customer.

Records 1-0 and 2-0 correspond to the original orders. The two numbers under (M, P) to the right of the other rows show which metarule (M) was used to generate the mask and which protocol rule (P) was selected and used to generate a work order event. For example, the FEE record 1-1 was generated from metarule 1 in Table 16 and protocol rule 9 in Table 15; the FEE record 2-4 was generated from metarule 4 and protocol rule 6. Although this pair could have generated a FEE for GU as well, it was not selected because the more specific metarule 1 had already selected a FEE work order. This example illustrates a collision.

After generating the proper work content, the application would then read the SEQUENCING AUTHORIZATIONS (Table 14) to determine the sequencing of the work, i.e., what work order records should have their STATUS consideration set to (A)vailable in the WORKORDERS table (Table 17). The rest are left, temporarily, as (U)navailable. For the GU order, the work process CHK_CRD is the only one marked Available.

The various ASSIGN_TO agencies that process work orders (e.g., CREDIT, BILLING_CPU) scan the Events table and select (A)vailable records for processing. When an agency finishes with a record, it issues a command that invokes an animation procedure to mark the record (C)omplete and consult the SEQUENCING AUTHORIZATIONS (Table 14) to determine which work orders can have their STATUS changed to (A)vailable.

Animation procedures typically execute interactively, invoked from a menu of commands selected by an end user. An order entry workstation, for example, would have commands to compute price quotations, enter orders, change orders, look up customer names and addresses, check inventory, and provide other customer support capabilities. That workstation would access Tables 6–16 or any other ruleforms in the ruleform clusters for price computations, process definitions, order changes, and so forth. Other workstations might share some of these commands but also have unique commands of their own, based on their desired functionality. Animation procedures may also be executed as standard background or batch tasks, triggered, of course, by rules.

Ultra-Structure ruleforms and animation procedures can be implemented using any commercially available RDBMS toolset.

The Deep Structure of Organizations

Development of ruleforms for various businesses has led to the observation that, fundamentally, they all manage work. The content of that work, the specific work processes followed, the volume of work, whether it is performed in-house or by outside vendors, and many other current distinctions simply reflect the differing surface and middle structures of these organizations. The CORE Hypothesis predicts that there is a common, definitive deep structure for all systems having certain family resemblances, for example, all organizations. As developed thus far, this CORE/650 can be defined as a feedback loop having the stages illustrated in Figure 1.

The first stage accepts explicit work orders and generates the implicit work called for by these explicit requests; this is called *preprocessing*. The second stage assigns, sequences, and releases implicit work orders to assigned agencies and provides facilities (function-specific workstations) for end users to complete the work assigned to them (*main processing*). The third stage accumulates completed work orders and allows executives to analyze these results, which may lead to better business policies for the future (*postprocessing*). The preprocessing step of this process model was illustrated by the example in Tables 6–17.

Pre-Processing. Each different type of work process that can be performed by an organization is called a *work protocol*. We have identified over 50 different kinds of work protocols that organizations invoke by means of various types of *explicit requests*, for example:

- Request from an external customer (domestic or foreign) to purchase finished goods
- Request from an external customer (domestic or foreign) to lease finished goods
- Request for an external customer (domestic or foreign) to borrow finished goods
- Request to an external customer (domestic or foreign) to return or destroy finished goods
- Request for field services at the site of an external customer
- Request for in-house services (e.g., equipment refurbishing) for an external customer
- Request from an internal customer (e.g., a department) to utilize finished goods
- Request from an internal customer (e.g., a department) to utilize supplies
- Request from an internal customer (e.g., a department) to utilize raw materials
- Request for a vendor (internal or external) to perform cycle counts
- Request for a vendor (internal or external) to perform preventive maintenance
- Request for a vendor (internal or external) to make a new product

The essential task of the WORK PROTOCOL Rules described in Table 15 is to translate these explicit requests from authorized agencies into the appropriate detailed work orders that must be executed to satisfy all parties to the transaction: seller, customer, taxing jurisdictions, other domestic or foreign governmental agencies, salespeople, vendors, and any others with an interest in the transaction. Work protocols are prescribed for a given set of circumstances by policies, procedures, laws, and traditions. Work protocols therefore involve the following:

- work content or *implicit requests* (the steps in the process—not when, but what must be done),
- the necessary sequencing of the steps in the process,

- the preferred assignment of the steps to various persons, departments, vendors, etc., and
- consequences of certain contingencies (e.g., a rejected credit request).

The example illustrated the first two of these requirements. Our experience to date indicates that the following existential rules are the basis of all business rules:

- Agencies: entities within or outside the firm that can generate or complete work for each other.
- Products: any activity, service, or good used internally or externally by the organization.
- Locations: any physical location or grouping of locations, or any mode of contact.
- Relationships: any legitimate potential relationships between any like objects.
- Time periods: dates and groupings of dates, e.g., Mondays, fiscal quarters.
- Commitments: any binding or nonbinding promises of performance by any agency.
- Scenarios: modes of existence, e.g., actual data or marketing projections.
- Existential types: general types of object, e.g., product or activity.
- Units of measure: any semantics associated with a number.
- Priorities: any precedence status, including special handling and timeouts.
- Reasons: causes for the generation of work or explanations for how work completed.

Simple versions of the first four were illustrated by Tables 6–9. It may seem problematic to try to force all business objects, relationships, and processes into these categories, and in many cases considerable analysis effort must be invested in determining where in this particular ontology a particular business concept belongs. Our experience to date is that there are always appropriate and technically sensible but sometimes nonobvious categories from the preceding list for every business concept. For example, "bank accounts" are "agencies" that can perform certain kinds of work. "GL accounts" identify specific accounting services performed for shareholders, regulators, and management; they therefore are actually services (therefore "products") offered by the organization to some of its most important customers.

Main Processing. Each agency (vendor, department, or individual) that does work has a logical or physical queue as defined by the ASSIGN_TO agency attribute of Table 17. Each agency may specify various selection criteria in order to see records assigned to it, for example, "all workorders" or "available work only." These may then be sorted as desired, by any field shown on the screen. Finally, the agency will complete each work order using various commands. Work orders that are completed cause animation procedures to check the SEQUENCING AUTHORIZATIONS (Table 14) and update the STATUS of any new work orders that are now (A)vailable, so other agencies may start work on the next step as soon as possible. When a computer is the assign-to agency, the work orders are executed in the order in which they are received.

Postprocessing. As work orders for a particular request are completed, they may be summarized for subsequent management analysis in terms of: work content statistics (each step tracked separately); work process statistics (inter-step metrics); and financial statistics (both cost and financial accounting).

The ruleforms for these summarizations are not shown. The data defined may then be read at the lowest level it was stored in (e.g., sales by product by month) or may be read and summarized at higher levels through use of the Network Rules (e.g., sales by product line by fiscal quarter).

Scalability. The same ruleforms may be used in multiple contexts. For example, SOURCING AUTHORIZATIONS (Table 13) may be used for: tracking of internal finished goods, work-in-process, or raw materials at the bin, warehouse, or other (e.g., regional) level; tracking of products or equipment at customer, vendor, or field engineer sites; and indicating what customers or customer types are allowed to order what products or services.

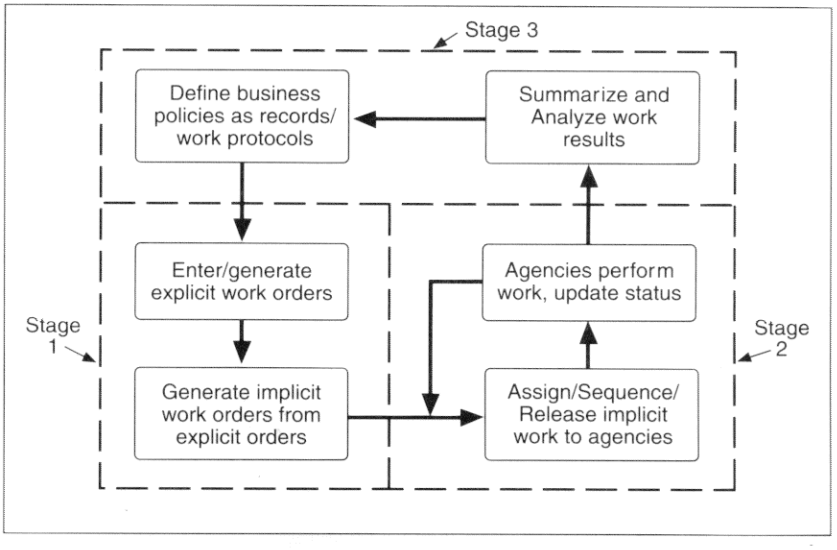

Figure 1. General business work process cycle

Because of this reuse, achieved by broadening the definitions of PRODUCT, LOCATION, AGENCY, etc., the number of ruleforms in an application increases very slowly as more functionality is added. Almost any application will require 10–15 ruleforms; but that application taken to its most sophisticated extreme is not likely to require more than 50 ruleforms. The goal is not simply to

minimize the number of tables (which would be foolish), but instead to be semantically "correct," and thereby to gain a more profound understanding of the ontology of the system we are modeling.

Experience to date indicates an asymptotic approach to <50 ruleforms having <20 attributes per ruleform as more and more functionality is supported by an application. While a large company will likely have more customers than a smaller one, and thus have more existential rules declaring the existence of those customers and their classifications, it is not logically necessary that the number of compound rules (largely based on groupings anyway) will also increase. The complexity of rules is independent of their number, and Ultra-Structure addresses these independent issues in ways that should work as well for representing large systems as for smaller systems.

Benefits for Business. These observations permit a new and different vision of business software in the future:

- Instead of automating the generation of millions of lines of code and/or maintaining a large library of reusable code, we may see an end to the need for software maintenance and an increase in the need for end users who are trained as *facilitators*, expert at defining heuristic business rules as data.
- Instead of patchworking together dozens of large applications developed by different vendors or departments, we may see a new and extremely high level of integration that transcends the current application divisions of, for example, manufacturing and accounts payable.
- We may see companies selecting from among several standard business engines, offered by different vendors, that would not define the rules of their business in advance through software but would define merely the format of all logically possible rules, permitting companies to establish policies and procedures based on the complex conditional rules that are now necessary to meet today's global competitive challenges.

Results

The theories presented here were initially developed by Intellinomics Corporation, a small California firm incorporated in 1980 and wound up in 1993. Under Intellinomics Corporation, Ultra-Structure was used to build applications for companies with revenues ranging from $25 million to Fortune 100 size, on DEC VAX, PC LAN, and HP MPE-XL platforms. Various transaction volumes were supported, up to about 50,000 orders/day. While some pieces of the CORE/650 have been implemented in production, most have not. Proving the CORE Hypothesis will require, among other things, that a single version of a CORE/650 be implemented to support different companies in different industries.

To demonstrate these analytical principles and associated technology in situations where the problem is known to be relevant and nontrivial, some firms hired Intellinomics to build a working prototype of one of their most complex applications on a PC. These prototypes were developed over short periods of three to five days. Prototypes developed for various companies include:

- Lot-specific manufacturing resource planning (MRP) for a $200-million drug manufacturer.
- Trust accounting for a $36-billion mutual fund.
- Military goods exporting and tracking for a major international freight forwarder.
- Order entry through billing for the U.S. division of a $10-billion manufacturer.
- Corporate cash management for a $35-billion oil company.
- Planned reorders for distributors of a major greeting card manufacturer.
- Product configuration validation for a $1-billion manufacturer.
- Factory and warehouse sourcing for a $200-million agricultural products company.
- Global billing for a major parcel delivery firm.

In each case, Intellinomics provided a designer and a programmer, and the client provided from two to ten technical and subject experts familiar with the application area. Every prototype successfully demonstrated substantial functionality to the satisfaction of customer experts. At least one was loaded with production data and found to be quite robust. They worked not because of any industry-specific expertise held by Intellinomics' developers, but rather because of an ability to discern and implement simple ruleforms and animation procedures quickly.

Measured Results in Existing Applications. In 1989 Intellinomics measured certain key metrics across eight production applications built by Intellinomics or its clients after 1985. These applications include order entry, inventory, and billing; sales analysis; financial products tracking and management; and governmental freight forwarding. At the time of measurement, the systems built by Intellinomics had been in production for an average of 18 months and those built by others for 20 months. The applications utilized DEC, PC LAN, or Datapoint platforms, in DATABUS (one application) or "C" (seven applications) languages. In Table 5, column 2 indicates Intellinomics' average and column 3 indicates the average achieved by two customers who built applications for themselves or other clients. These numbers are for introductory discussion and have not been adjusted for constant dollar value, for varying overheads or billing rates, or for other criteria a thorough economic analysis should consider.

A Case History. Intellinomics used an early version of these concepts to develop order entry, inventory control, and billing applications for a wholesale replacement auto glass business in 1985, after the customer had unsuccessfully tried to automate for seven years. In the first nine years after installation, the customer more than tripled its gross revenues, to about $40 million per year. Its DEC MicroVAX II with a 9-MB internal memory supported 48 active on-line users and 12 background tasks at eight sites around California. It initially processed about 30,000 orders per month having one to 99 line items per order. By 1991 the company had moved to a VAX 3600 with

a 32MB internal memory and was handling over 50,000 orders a month through more than 60 active users. The company has no software staff at all; Intellinomics maintained the application with less than 15 person-hours per month. The customer asserts that it has the lowest operating costs (not just MIS, but all areas) in its industry. It now has 21 sites around the U.S.

Life-Cycle Costs of a CORE for Business. Fortune 100 companies average 35 million lines of code to implement their business rules [5]. If we attribute 95% of this to interdivisional duplication (e.g., 25 different inventory applications, 20 different GL applications) and cross-application bridges, we may speculate that ideally such a company could be run with 2 million lines of code. Using Barry Boehm's COCOMO model [1] of the development costs for such a (semidetached) application without cost drivers factored in and an industry rule of thumb for maintenance costs, we get the following lifecycle costs:

- number of lines of 3GL code in conjunction with DBMS—2 million
- initial person-years of development effort—1,245
- initial project cost @ $5,000/person-month—$75 million
- maintenance/year as percent of initial cost—40%
- annual maintenance cost (uninflated)—$30 million
- development cost per line of code—$37
- five-year cost per line of code—$112

These figures indicate that an ideal, wholly integrated enterprise-wide business application might cost about $225 million for development and five years' use. The application might then be wholly replaced, starting a new life cycle. Myers uses several approaches to estimate cost and effort for a hypothetical two-million-line command-and-control application, and generally observes that, depending on organizational productivity, the effort is in the range of 2,250–10,833 person-years at an uninflated cost of $300 million to $1.6 billion [6]. This excludes long-term maintenance, a critical indicator of design robustness that at least doubles the software development cost [1, 4]. Certainly for MIS-type applications the costs would be lower.

Table 5. Ultra-Structure Application Metrics

Application Profiles	Intellinomics	Others
# Applications:	4	4
Orders Entered/Year:	102,300	684,000
$ Value of Orders/Year:	3+ billion	366 million
# Users/Application:	25	18
# Screens/Application:	82	31
# Reports/Application:	42	24
Basic Metrics		
Calendar Months to Develop:	8	10
Person-Months to Develop:	11	23
Person-Months to Maintain:	2	31
# Tables/Application:	18	28
Lines Hand-Written Code (HWC):	5,248	20,505
$ Development Cost:	88,196	124,500
$ Maintenance Cost:	34,346	155,000
Derived Metrics		
$ Development Cost/Line HWC:	16,81	6.07
Annual Maintenance as % of Development $:	12	40

By way of illustration, we present an ideal CORE/650. We have not put an enterprise-wide complex operating rule engine into production, so our estimates of its life-cycle costs are largely conjectural. Nevertheless, as a first approximation, we believe such an application might look as follows:

- number of lines of 3GL code in conjunction with DBMS—100,000
- initial person-years of development effort—43.5
- initial project cost @ $5,000/person-month—$2.6 million
- maintenance/year as percent of initial cost—15%
- annual maintenance cost (uninflated)—$0.4 million
- development cost per line of code—$26
- five-year cost per line of code—$46

These figures indicate that an ideal enterprise-wide business application—stating rules as data rather than software, but otherwise using standard software development tools—might cost only $4.6 million for development and five years' use. Unlike applications that must be discarded after the surface structure they support changes too much and they can no longer be patched economically, this application could be supported indefinitely if its deep structure did in fact anticipate all logically possible rules. Although simple comparison of these figures is risky, we are interested here in general orders of magnitude, not minor percentages. These two ideals differ by a factor of 49.

A company using standard off-the-shelf packages rather than creating custom applications will need to either adapt itself to the surface structures supported by the packages or modify the packages. In semiregulated areas such as financial accounting, this is not usually a problem, because different businesses tend to follow the same work processes anyway (e.g., for posting of journal entries), encoding their accounting differences as data via their chart of accounts. In application areas where inter- or intracompany surface structure differences can be great, package modifications can pose significant technical and financial risks.

In theory, a CORE/650 should not have to be custom written, as was assumed in the preceding cost predictions, but could be installed simply by entering appropriate operating rules for a given business. That is, in fact, the objective of Ultra-Structure. In that case, the cost savings would be

substantially greater than we have speculated.

Conclusion

Humans are adept at formulating apparent rules from the observed behavior of a system. Seeking this truth behind appearances is an ancient and important activity, and we have developed this capability over the last three centuries into an endeavor we now call science. But what happens when there are so many rules—tens of thousands and more—that we cannot begin to comprehend them, or their implications, as a whole? That is the state of our current understanding in business, science, government, and other areas of complex systems and processes.

Rules specify processes, which specify systems. To understand complex systems we must understand complex processes, and to understand complex processes we must understand complex rules. By abstracting complex rules to a higher level, we find systems that obey different rules can still share the same rule formats. This abstraction of rules provides additional simplicity without simplification, and is made possible through the new conceptual structure of ruleforms. In the process of discovering the ruleforms of a system, we also obtain a deeper understanding of the fundamental nature or ontology of that system.

Because we can represent the rules of systems with very few ruleforms and relatively simple animation procedures, we can implement prototypes very quickly on a computer. The ability of a model to accurately represent complex rules, including exceptions, can be experimentally verified. Alternative deep structures for systems can be tested experimentally, and Occam's razor can be applied to shed unnecessary ruleforms. New *development versions* of an application can be developed and explored every four to six weeks until no counterinstances of unsupportable rules are found. This process and its objectives contrasts sharply with traditional approaches.

Future Work. Intellinomics Corporation was closed and The George Washington University Notational Engineering Laboratory (NEL) was established in order to devote full time to studying the comparative historical and theoretical foundations of various formal symbol systems (notations). Work is under way to:

- Understand the structure of notational revolutions (such as the move from Roman to Arabic numerals) by examining the evolution of various notational systems (the alphabet, mathematics, chemistry, music, dance, logic, cartography, time, and money).
- Specify criteria or characteristics that major new notational systems might have.
- Apply Ultra-Structure to derive the deep structure of a variety of different types of systems, including scientific arguments, law, biology, physics, games, and music.

Preliminary results are very encouraging. The discovery of simplicity through the study of semantic form may yet permit us to overcome the complexity barrier we now face. ◘

Acknowledgments

We are grateful to Peter Denning and Robert Fearing for many constructive comments.

References

1. Boehm, B. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, N.J., 1982.
2. Brooks, F.P., Jr. No silver bullet: Essence and accidents of software engineering. *Computer* (Apr. 1987).
3. Codd, E.F. *The Relational Model for Database Management, Version 2.* Addison-Wesley, Reading, Mass., 1990.
4. Hager, J.A. Software cost reduction methods in practice. *IEEE Trans. Soft. Eng. 15* (Dec. 1989), 1638–1644.
5. Melymuka, K. Managing maintenance: The 4,000-pound gorilla. *CIO* (Mar. 1991), 74–82. Quoted in Zwass, V. *Management Information Systems.* Wm. C. Brown, 1992.
6. Myers, W. Allow plenty of time for large-scale software. *IEEE Soft 6* (July 1989), 92–99.
7. Navathe, S.B. Evolution of data modeling for databases. *Commun. ACM 35,* 9 (Sept. 1992), 112–123.
8. O'Hare, R.J. Systematic generalization, historical fate, and the species problem. *Syst. Biol.* (Sept. 1993).
9. Palvia, S.C., and Gordon, S.R. Tables, trees, and formulas in decision analysis. *Commun. ACM 35,* 10 (Oct. 1992).
10. Potter, W.D., and Trueblood, R.P. Traditional, semantic, and hyper-semantic approaches to data modeling. *Computer* (June 1988), 53–63.
11. Prigogine, I. *Order Out of Chaos: Man's New Dialog With Nature.* Bantam Books, New York, 1984.
12. U.S. Congress Office of Technology Assessment. *Information Technology R&D: Critical Trends and Issues.* Government Printing Office, Washington, DC, 1985.

JEFFREY G. LONG is the director of the Notational Engineering Laboratory at George Washington University. Current research interests include the comparative evolution of notational systems, the philosophy of notation, and the critique of tools for the representation of general complex systems. **Author's Present Address:** George Washington University/CMEE, Academic Center, Room 703A, Washington, DC 20052; email: jlong@seas.gwu.edu

DOROTHY E. DENNING is Professor of Computer Science at Georgetown University. Current research is focused on policy and technical issues relating to cryptography and wiretapping. **Author's Present Address:** Georgetown University, Department of Computer Science, 225 Reiss Science Building, Washington, DC 20057; email: denning@cs.georgetown.edu

Appendix: Sample ruleforms for work orders

Table 6. AGENCY Ruleform and Rules

AGENCY	NAME
BILLING_CPU	Billing Computer
CPU	Computer
CREDIT	Credit Department
EXEMPT	Exempt from Sales Tax
EXT_CUST	External (Paying) Customers
FACTORY1	Factory #1
GU	Georgetown University
MNFR	Manufacturer
NON_EXEMPT	Subject to Sales Tax
ORG_A	Organization A
(ANY)	(Any Agency)

Table 7. PRODUCT Ruleform and Rules

PROD	DESCRIPTION
ABC486	Laptop Computer
CHK_CRD	Check Customer Inhouse Credit
CHK_LTR	Check Customer Letter of Credit
DELIV	Deliver Product
DISCNT	Compute Fee Discount
FEE	Compute Fee
FROZEN	Frozen Products
NON_EXEMPT	Subject to Sales Taxes
PICK	Pick Inventory
CHEM_B	Chemical B
PRT_CUST_DEC	Print Customs Declaration
ROOM_TEMP	Room Temperature Products
SHIP	Ship Inventory
STAX	Compute Sales Tax
(ANY)	(Any Product or Service)

Table 8. LOCATION Ruleform and Rules

LOCATION	DESCRIPTION
225RSB	225 Reiss Science Bldg.
37BHT	37 Bret Harte Terrace
BIN01	Bin #1
BIN15	Bin #15
DC	District of Columbia
EAST_COAST	East Coast
EURO	European Locations
FACTORY1	Factory 1
FRANCE	France
PARIS	Paris
US	U.S. Locations
WASH	Washington
(ANY)	(Any Location)

RELCODE	DESCRIPTION
AREA	Economic Community
CITY	City
CUST_TYPE	External or Internal Customer
REGION	General Region
STATE	State
STAX_TYPE	Sales Tax Status
STORAGE_TYPE	Type of Storage Required
STREET	Street

Table 9. RELCODE Ruleform and Rules

Table 10. AGENCY NETWORK Ruleform and Rules

AGENCY1*	RELCODE*	AGENCY2*
GU	CUST_TYPE	EXT_CUST
GU	STAX_TYPE	EXEMPT
ORG_A	CUST_TYPE	EXT_CUST
ORG_A	STAX_TYPE	NON_EXEMPT

PROD1*	RELCODE*	PROD2*
ABC486	STORAGE_TYPE	ROOM_TEMP
ABC486	STAX_TYPE	NON_EXEMPT
CHEM_B	STORAGE_TYPE	FROZEN

Table 11. PRODUCT NETWORK Ruleform and Rules

Table 12. LOCATION NETWORK Ruleform and Rules

LOCN1*	RELCODE*	LOCN2*
BIN01	AREA	FACTORY1
BIN15	AREA	FACTORY1
FACTORY1	STREET	37BHT
225RSB	CITY	WASH
37BHT	CITY	WASH
WASH	STATE	DC
DC	REGION	EAST_COAST
EAST_COAST	AREA	US
PARIS	REGION	FRANCE
FRANCE	AREA	EURO

Table 13. SOURCING AUTHORIZATIONS Ruleform and Rules

AGENCY*	PROD*	LOCN*	QTY_ON_ HAND	QTY_ COMM	REORDR_ PT
(SELLER)	ABC486	BIN01	50	20	15
(SELLER)	ABC486	BIN15	150	100	15
GU	ABC486	DC	12	12	0

Table 14. SEQUENCING AUTHORIZATIONS Ruleform and Rules

PARENT PROD*	SEQ_NO	CHILD_PROD*
CHK_CRD	1	PICK
PICK	1	PRT_CUST_DEC
PRT_CUST_DEC	1	SHIP
PRT_CUST_DEC	2	FEE
CHK_LTR	1	PICK

Table 15. WORK PROTOCOL Ruleform and Rules

	PROD1*	AGENCY*	PROD2*	SHIP_ TO*	SHIP_ FROM*	SEQ_ NO	ASSIGN_ TO	PROD3*	PROD4*	QTY
1	DELIV	EXT_ CUST	(ANY)	US	US	1	CPU	CHK_ CRD		
2	DELIV	EXT_ CUST	(ANY)	EURO	US	1	CREDIT	CHK_ LTR		
3	DELIV	EXT_ CUST	ABC486	EURO	US	1	CPU	PRT_ CUST_DEC		
4	DELIV	(ANY)	(ANY)	(ANY)	(ANY)	1	FACTORY1	PICK	"	"
5	DELIV	(ANY)	(ANY)	(ANY)	(ANY)	1	FACTORY1	SHIP	"	"
6	DELIV	EXT_ CUST	ABC486	(ANY)	US	1	BILLING_ CPU	FEE	"	1500.00
7	DELIV	NON_ EXEMPT	NON_ EXEMPT	DC	(ANY)	1	BILLING_ CPU	STAX	"	0.0575
8	DELIV	EXT_ CUST	ABC486	EURO	US	1	BILLING_ CPU	DISCNT		0.05
9	DELIV	GU	ABC486	DC	US	1	BILLING_ CPU	FEE	"	1250.00

Table 16. WORK METARULES Ruleform and Rules

PROD1*	SEQ_NO	AGENCY	PROD2	SHIP_TO*	SHIP_FROM*
DELIV	1	(SAME)	(SAME)	STATE	AREA
DELIV	2	CUST_TYPE	(SAME)	AREA	AREA
DELIV	3	CUST_TYPE	(ANY)	AREA	AREA
DELIV	4	CUST_TYPE	(SAME)	(ANY)	AREA
DELIV	5	STAX_TYPE	STAX_TYPE	STATE	(ANY)
DELIV	6	(ANY)	(ANY)	(ANY)	(ANY)

Table 17. WORKORDERS Ruleform and Rules

SEQ_NO	ASSIGN_TO	AGENCY*	PROD1*	PROD4*	SHIP_TO*	SHIP_FROM*	STATUS	QTY	M
1-0		GU	DELIV	ABC486	225RSB	FACTORY1		2	
1-1	BILLING_CPU	GU	FEE	ABC486	DC	US	U	1250.00	1,
1-2	CPU	EXT_CUST	CHK_CRD		US	US	A	2500.00	3,
1-3	FACTORY1		PICK	ABC486			U	2#	6,
1-4	FACTORY1		SHIP	ABC486			U	2	6,
2-0		ORG_A	DELIV	ABC486	PARIS	FACTORY1			
2-1	FACTORY1	EXT_CUST	PRT_CUST_DEC		EURO	US	U		−2
2-2	BILLING_CPU	EXT_CUST	DISCNT		EURO	US	U	0.05	2
2-3	CREDIT	EXT_CUST	CHK_LTR		EURO	US	A	3000.00	3
2-4	BILLING_CPU	EXT_CUST	FEE	ABC486		US	U	1500.00	4
2-5	FACTORY1		PICK	ABC486			U	2	6
2-6	FACTORY1		SHIP	ABC486			U	2	6