REACH: Automated Database Design Integrating First-Order Theories, Reconstructive Expertise, and Implementation Heuristics for Accounting Information Systems

Stephen R. Rockwell[1*] and William E. McCarthy[2]
[1]The University of Tulsa, USA
[2]Michigan State University, USA

ABSTRACT **This paper describes the integrated use of different knowledge types in an automated software engineering tool called REACH. These types include: (1) a *first-order domain theory* called the REA accounting model, (2) *reconstructive expertise* gleaned from textbooks on accounting system design in a bookkeeping environment, and (3) *implementation compromise heuristics* derived from the design experience of database designers. This tool aids an enterprise database analyst in the conceptual design stages of view modeling and view integration.**

INTRODUCTION

Early expert systems concentrated to a large degree on the modeling of human cognitive processes. This means that the knowledge structures embedded in such systems were primarily restricted to those that directly emulated the heuristic methods by which a human decision maker would attempt to control the complexity involved in solving a given problem. By contrast, more recent systems attempt to incorporate knowledge structures of both the heuristic and non-heuristic types into their consultation sessions. As many AI professionals had predicted in the 1980s, knowledge-based systems have become more successful as they have migrated away from 'single-subject emulation' into integrated components of larger computer systems.

This paper describes the integrated use of different types of expertise in a computer-aided software engineering (CASE) tool for account-

*Correspondence to: Stephen R. Rockwell, School of Accounting/Department of MIS, 313 BAH, The University of Tulsa, Tulsa, OK 74104-3189, USA.
E-mail: steve-rockwell@utulsa.edu

Contract grant sponsor: Department of Accounting at Michigan State University
Contract grant sponsor: Ameritech
Contract grant sponsor: Arthur Andersen & Co.

CCC 1055-615X/99/030181-17$17.50

ing system design. This system is called **REACH**, and initial discussion of its proposed functioning is found in McCarthy and Rockwell (1988). REACH is designed to aid in the process of database design in general and in the subprocesses of view modeling and view integration in particular. To do this, REACH uses three kinds of accounting domain knowledge:

- First-order theories of accounting derived from conceptual (i.e. semantic) analysis of accounting practice and accounting theorists
- Reconstructive expertise of accounting system implementers largely derived from textbook descriptions of 'typical' bookkeeping systems
- Implementation heuristics for construction of events-based accounting systems derived from the database design experiences of the authors in such work.

For its consultation process, REACH must also avail itself of both **methods** knowledge (of data modeling, normalization, structured analysis, etc.) and **target system** knowledge (of possible hardware restrictions, for example). However, such use is described elsewhere (e.g. Loucopoulos and Harthoon, 1988; Ryan, 1988; Dogac *et al.*, 1989; Lloyd-Williams and Beynon-Davis, 1992; Loucopoulos and Theodoulidis, 1992; Storey and Goldstein, 1993; Storey, 1993), and we concentrate here on the integrated use of the different types of accounting domain knowledge in REACH. While Storey *et al.* (1997) use domain-specific knowledge structures in database design work, their structures were developed from a small number of example cases, as opposed to our use of what we refer to as reconstructive domain knowledge. Their domain structures also lack the unifying metamodel that our use of REA theory provides.

The remainder of this paper is organized as follows. The next section describes the process of structured database design in an accounting context and the REACH approach to problem solving in this arena. The third section describes in more detail the design task and the different types of domain knowledge enumerated above. The fourth section discusses the use of such knowledge in solving design problems and in making decisions about implementation compromise. The fifth section describes the current implementation of REACH, and the sixth section ends the paper with a summary and discussion of future directions suggested by this research.

DESIGN OF SHARED ENVIRONMENT ACCOUNTING SYSTEMS

Virtually all business enterprises have accounting information systems that model the inflow and outflow of economic resources (like inventory and cash) and that allow periodic compilations of company profitability. In their most primitive forms, such systems comprise account classifications and bookkeeping conventions that accommodate tracking of financial transactions. In their more modern forms, these systems consist of various computer files and programming modules, such as general ledger systems, payroll systems, accounts-receivable systems, etc.

In a shared data environment in which nonaccounting decision makers need common access to economic transaction information, many computerized accounting and bookkeeping systems are found wanting, and the need becomes apparent for a process that will redirect the design of these systems toward a database orientation. Such a design process is described in McCarthy *et al.* (1989) and in Geerts *et al.* (1996), and the embedding of its knowledge structures in a CASE tool is the goal of the REACH implementation. A summary of the accounting systems design process used here is illustrated in Figure 1 and explained below.

Accounting information system design begins with a desire to model the economic events, resources, and agents of a given business enterprise, and it ends with a specified and implemented computer system. Not all aspects of the business enterprise need to be captured in the formal information system, and many characteristics actually have little or no decision or accountability use (for example, the number of bricks in a company's headquarters building or the romantic interludes among its employees). Additionally, there will be many

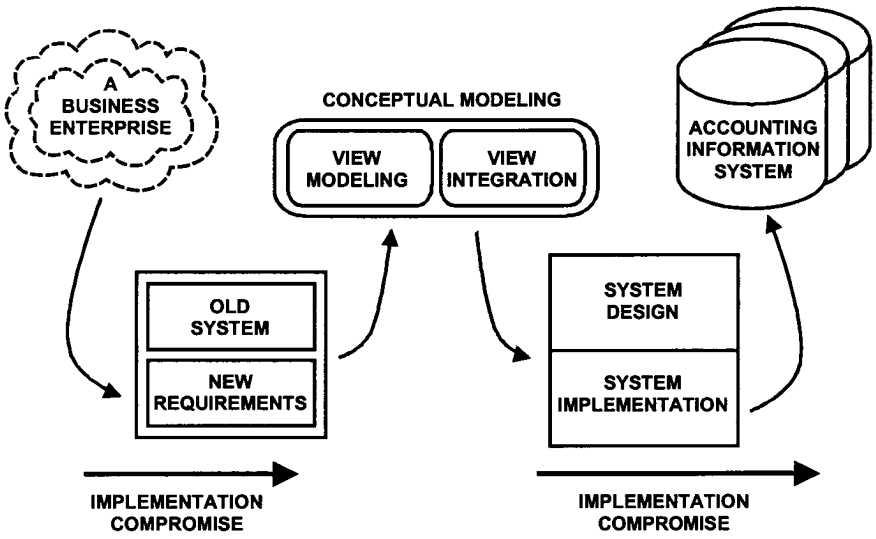

Figure 1 Accounting information system development

characteristics that are certainly desirable from a decision usefulness perspective (and hence, desirable components of an enterprise information system), but which will prove later to be incapable/infeasible of being measured or represented (for example, the productivity-enhancing skill level of certain employees). The information system design and implementation process is then one of compromise (what to model in the system and what to leave out) as it proceeds from start to finish.

In REACH, we are using a systems development methodology adapted from structured analysis (Yourdon, 1991) and conceptual database design (Lum *et al.*, 1979, Batini *et al.*, 1992). The requirements definition phase of this methodology begins the process of implementation compromise by extracting both old and new requirements from potential users and by documenting those requirements formally within a structured analysis CASE tool.

As illustrated in Figure 1, we then proceed into conceptual database modeling, a phase in which we attempt to temporarily suspend the process of implementation compromise through the use of first-order accounting theories and reconstructive enumeration of 'typical' system elements. At this stage of database design, we are most interested in producing a set of individual **user views** or **external schemas**. A user view typically provides details of some subset of the database in which a particular group of users is interested. User views frequently focus on transaction processing activities, such as tracking sales transactions, or on forms used for exchanging information among people. Examples of such forms include information output, such as aged accounts receivable reports or customer billing statements, and data input, such as purchase orders or employee timecards.

After all the external schemas have been produced, we enter the next phase of **view integration**. The output of this phase includes the **conceptual schema**, which combines and reconciles the external schemas into a global description of the whole database. In this and later phases proceeding to final implementation, **implementation compromise** resumes its central focus. Some of REACH's knowledge structures are used here, either by dictating directions for leaving out certain types of conceptual components of events-based accounting systems or by directing the resolution of view integration conflicts.

The actual implementation domain of REACH is illustrated in Figure 2. Our tool

works primarily in the second step—conceptual design—of the four-step database design process outlined first by Lum *et al.* (1979) and extended by Batini *et al.* (1992). We assume that requirements definitions are completely finished with a CASE tool and handed to us in the form of data flow diagrams (DFD) and a data dictionary. REACH output consists of an integrated schema that is given to an implementation team for translation into a particular DBMS environment during the implementation design phase. Some of the compromise necessary in this 'translation' phase may be accomplished during view integration, as the knowledge necessary for such compromise already exists within REACH. Thus, in Figure 2, the dashed line marking the boundaries of our system encompasses part of the **target system knowledge** and **implementation design** structures.

As mentioned, three kinds of accounting knowledge structures are utilized within REACH, and it is to this utilization that we next turn.

DESIGN TASK AND DOMAIN KNOWLEDGE REPRESENTATION

Design Task

REACH aids the database design steps of view modeling (or view design) and view integration. **View modeling** takes a list of data elements (from a DFD data store or a potential user 'wish list') and expresses them in the form of a conceptual Entity-Relationship (E-R) schema (Chen, 1976). A **view** consists of the actual data elements needed for a particular computer program to be run or for a particular decision to be made. These views ignore details of the physical storage structures that will be

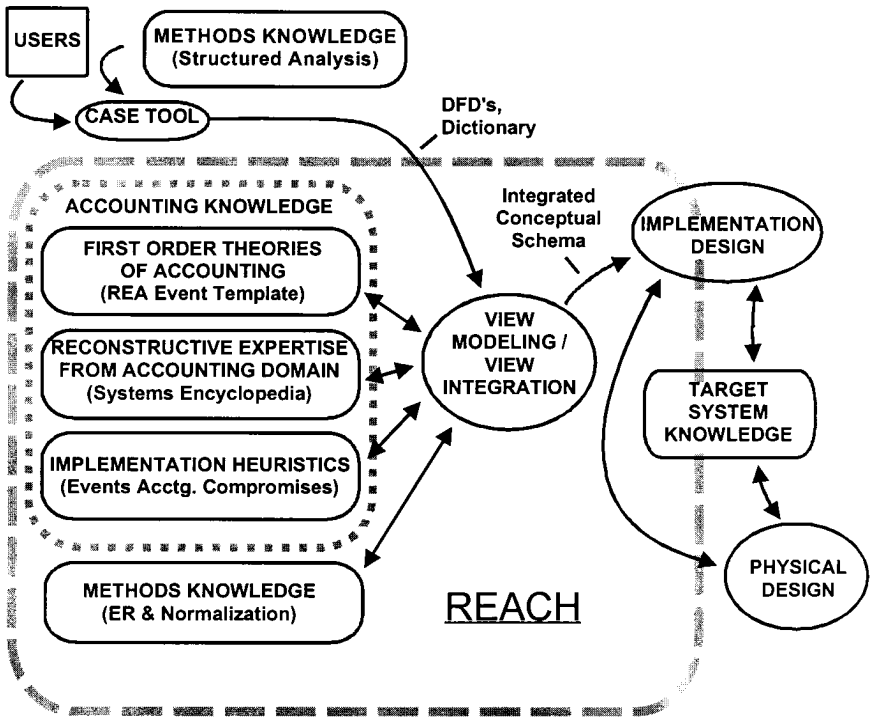

Figure 2 The REACH domain

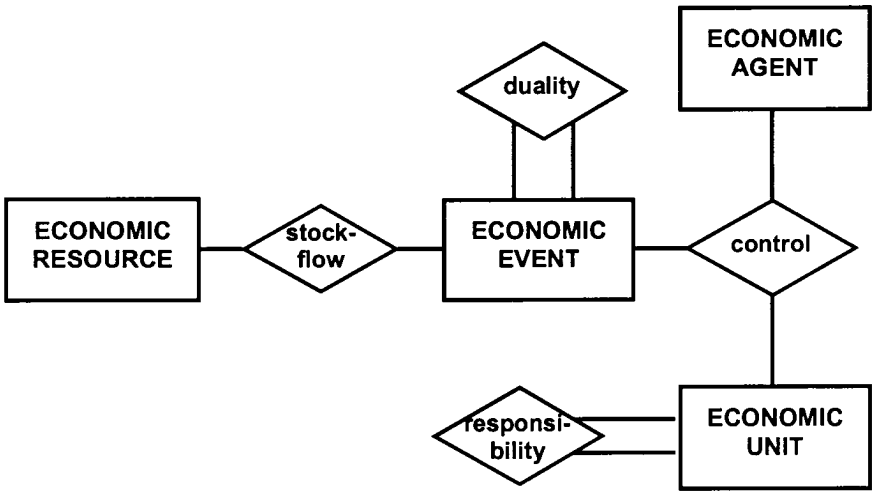

Figure 3 The REA accounting model (source: McCarthy, 1982)

used eventually to implement the database. They instead focus on the entities of interest (resources, events, and agents), data types assigned to the entities, relationships among the entities, and constraints placed upon the entities and relationships. Judgments involved in view modeling include the identification or reconstruction of entities, relationships, and attributes within the data set and correct specification of them in E-R form.

View integration takes a series of modeled views and iteratively combines them into one integrated E-R schema for the entire business enterprise. Judgments involved in this process include identification of combination and/or integration bases for various views, decisions either to omit some previously identified elements or to add some previously unidentified elements, and resolution of integration conflicts. Such conflicts occur when separate user groups model the same real world objects of interest in different ways. For example, the same entity might be assigned different names, an entity might be modeled using different data types, or a relationship may be given different constraints. Batini *et al.* (1986) provide a comprehensive discussion of integration conflicts and integration methodologies, and we discuss integration conflicts later in this paper.

In a certain way, view modeling and view integration can be conceived as analogous to the process of combining certain chemicals (with known desirable properties) into a compound that will meet a variety of needs in an integrated fashion. The first step involves correct specification of the individual components, while the second involves correct identification of inconsistent or superfluous elements and a process for seamless integration.

Knowledge Representation

Within REACH, three distinctly different types of accounting knowledge are brought to bear on the conceptual modeling process. We now discuss each at greater depth.

First-order Theories of Events Accounting Systems

Our starting point for the conceptual design of shared environment accounting systems is the REA model illustrated in Figure 3 (McCarthy, 1982). This 'accounting event template' was derived by semantic analysis of current practice and its component names appeal to the ideas of well known accounting theorists such as Ijiri (1975) and Mattesich (1964). Its model name is derived from the essential components of the

transaction template, which are its economic **r**esources, **e**vents, and **a**gents. The REA framework derives from fundamental accounting principles, such as 'accountability' and 'duality', that characterize the tracking of economic events in a business enterprise. REACH is a software system designed to augment CASE support for REA-oriented accounting system design, and it uses REA theory in each of the database design stages of view modeling and view integration. The use of an REA metamodel provides leverage both in the actual modeling consultation and in the acquisition and modeling of reconstructive domain knowledge, described later in this paper.

A view modeling consultation session in REACH will be structured around a specific instance of an REA event type. Users are guided in their interpretation of view elements by this template, and they are also encouraged to use their own judgment in augmenting or reorienting a given set of data names. In a sense what they are asked to do (among other things) is to interpret the situation in front of them as a specific instance of a known constellation of entities and relationships. What these users will often find is that a given set of data elements is incomplete and misinterpreted in light of this theoretical framework because of unapparent implementation compromise. In essence, what is happening is that REACH attempts to identify the correct REA event template for a given user view. Once that is known, the user view is compared to a **full-REA** stereotype for the events being modeled. From that comparison, REACH can identify potentially missing objects, make recommendations about data types, or reconcile differences in naming conventions.

During view integration, the REA template is used to guide model synthesis by identifying 'intersection points' between views. For example:

- Every increment event needs to be paired with at least one decrement event via a duality relationship and vice versa; therefore, a view of sales processing is linked to a view of remittance advice processing via a **Cash Receipt** *pays for* **Sale** interpretation.
- Every resource must have both inflow and outflow events; therefore, sale processing in a retail enterprise could be linked to purchase processing via a **Purchase** *is inflow to* **Inventory** *has outflow of* **Sale** interpretation.
- Economic units are arranged in responsibility hierarchies; therefore, a view of sale transaction processing could be linked to a regional sales report view via a **Salesperson** *is assigned to* **Sales Region** interpretation.

Reconstructive Expertise of Accounting System Implementers

In the bookkeeping days of fifty years ago, part of an accountant's expertise would be the knowledge needed to design a chart of accounts (i.e. a declarative classification structure) and a set of bookkeeping procedures for a particular company. These accountants either carried in their head or had library access to a 'journal and ledger template' for a particular type of business enterprise. Thus for instance, a good accountant would be able to identify an adequate chart of accounts for a particular store, for a particular funeral home, for a particular hospital, etc. before he or she even visited the actual establishment. Quite obviously, this framework would be tempered and altered by actual experience, but it would serve an invaluable role in summarizing the past experiences of knowledgeable experts in the field and in preventing sins of omission (as opposed to discouraging sins of commission) in accounting system design.

Reconstructive expertise of the type explained above is built into REACH. For a given set of industry types, we are using the *Encyclopedia of Accounting Systems* (Plank and Plank, 1994; Pescow, 1976), which provides managerial advice, sample chart of account structures, and representative bookkeeping procedures by industry classification. We transform this advice into entity-based enterprise models and use them during view integration in lieu of the 'management view' starting point often advocated by database theorists. This means that the industry entity template becomes the overall conceptual schema to start with and that view integration proceeds by adding individual modeled views onto it. We call this

management view the **management schema**, or **m-schema**. There are certainly definite needs for this industry template in the view modeling process, but we are deferring that implementation to a later prototype stage.

Implementation Heuristics for Events-Based Accounting Systems

A full events-based accounting system is a theoretical ideal that realistically would not be implemented. Nobody would keep full event histories perpetually unless storage technologies become absolutely costless and methods for abstracting from detail become absolutely painless. Event system implementation involves essential compromise (i.e. throwing both intensional and extensional database features away). Since this process is inevitable, we are attempting to provide in REACH heuristic guidance gleaned from our own considerable experience in events-based accounting implementations.

The next section discusses in greater detail the use of these three types of knowledge in resolving some conceptual modeling problems and in making decisions regarding implementation compromise.

INTEGRATION CONFLICTS AND IMPLEMENTATION COMPROMISE

In the previous section, we indicated that accounting knowledge played a major role within view integration by identifying intersection points among views. That accounting knowledge came primarily from the **principles** level via use of REA theory. We also indicated that reconstructive expertise was used for constructing the **m-schema**, which was used as a starting point for the integration process. Within the integration process itself, there are two other areas in which this knowledge can be used: (1) when trying to resolve conflicts between a new user view and the existing **m-schema**, and (2) when modeling common implementation compromises.

Integration Conflicts

The process of conceptual modeling is complex, and considerable research activity has been devoted to it. Some of the research efforts focus on creating knowledge-based systems (KBSs) to aid in conceptual modeling tasks. While a number of systems of varying complexity have been developed in both academic and commercial settings, in general they rely heavily upon the human user during the view integration phase. To a greater or lesser extent, these systems 'fail' when attempting to resolve automatically some of the problems of view integration, and consequently they must 'ask' the user to do much of the work. Table 1 defines the most common integration conflicts that cause 'failures' in such integration systems.

Through its use of accounting knowledge, REACH provides increased problem-solving assistance on several levels. In some cases, integration conflicts can be resolved by reference to internal knowledge structures, without further input from the user. For example, an REA template for the **Sale** event would typically have a separate entity for **Employee**. That entity is one about which we would normally wish to capture a variety of information (name, address, Social Security Number, etc.). In addition, that entity participates in numerous business events, filling, at various times, the roles of Economic Unit and Economic Agent. If a user view contained a **Sale** event with no **Employee** entity, but with an *Employee#* attribute, REACH would instantiate an **Employee** entity. The user would be notified as to why this occurred.

In other cases, conflicts can be resolved by querying the user for accounting or business-oriented data. This is an improvement over many modeling systems that query the user in database terms. REACH requests data in the business terms a user is more likely to understand, and thus it can lessen the need for a systems analyst to act as an intermediary. This eliminates one possible source of communication errors, as the user's knowledge of the business can be acquired directly from that specific person and need not be 'translated' through an intermediary, who

Table 1. Integration conflicts

Type	Conflict	Description
Naming	Homonym	The same name is used for two different concepts, giving rise to inconsistency unless detected. For example, merging two entities of this type in the integrated schema would result in producing a single entity for two conceptually distinct objects.
	Synonym	The same concept is described by two or more names. Keeping each name modeled as a distinct entity in the integrated schema would result in modeling a single object by means of multiple entities.
Structural	Type conflict	The same concept is represented by different modeling constructs in different schemas. For example, a class of objects may be represented as an entity in one schema and as an attribute in another schema.
	Dependency conflict	A group of concepts are related among themselves with different dependencies in different schemas. For example, a relationship between two entities may be shown as 1:1 in one schema, but m:n in another schema.
	Key conflict	The same concept is assigned different keys in different schemas. For example, SS# and Emp_id may be the keys of Employee in two component schemas.
	Behavioral conflict	The same class of objects is assigned different insertion/deletion policies in distinct schemas. For example, in one schema a department may be allowed to exist without employees, whereas in another, deleting the last employee associated with a department leads to the deletion of the department itself.

Source: Rockwell (1992) adapted from Batini *et al.*, (1986).

would often be less familiar with the problem domain.[1]

Implementation Compromise

In its most pure form, an REA-based accounting system designed as a relational database would implement each entity and relationship as a separate table. In addition, full event histories would be kept forever, and many double-entry accounting artifacts would be materialized with procedures rather than be directly represented in the system's tables. For example, rather than store accounts receivable balances, those numbers would be calculated (when needed) as the imbalance between sales to a particular customer and cash receipts from that customer. Additionally, many of the normal period expense (e.g. advertising expense) would be modeled and implemented as two (as opposed to one) economic events with one event showing the acquisition of a resource, and the other showing the consumption of that resource (McCarthy, 1982, p.573).

There are many reasons why we would not expect to see such an implementation with technology of the present or near future. Costs of data storage and processing, with the attendant time delays for processing vast quantities of data, prohibit such a system from an efficiency standpoint. From an accounting decision-making perspective, some of the data would be of limited value. For example, if expenditures for insurance are primarily relevant as part of quarterly or annual reports, trying to track insurance expenditures as separate asset acquisition and use events provides no incremental decision value beyond current treatment of those expenditures as period expenses. It should also be noted that there exist many non-database accounting systems, and such systems might require modification of the pure REA-based conceptual model

[1] For example, a typical system might point out a conflict in structural constraints between two schemas modeling the **Sale** and **Payment** events. A business user might not fully understand the concept of structural constraints and thus be uncertain as to the correct resolution. REACH would instead ask about how the company handles payments for sales, and use the response to identify the correct structural constraints.

beyond the compromises just suggested. For example, in a traditional file-oriented application with multitudes of sequential processing, the types of fully normalized data structures that occur in a relational database system would probably impose tremendous storage and processing inefficiencies on the user. These various pressures for design compromise may be categorized into two major groups: (a) compromises based upon **information** use characteristics, and (b) compromises based upon **physical implementation** characteristics. Each of these is discussed in turn in the next two sections.

Information Use Compromise Heuristics

Figure 4 shows examples of four types of information use compromise heuristics. Such compromise is based upon the underlying concept that an accounting system should store data and produce information that has relevance to the users of that system's outputs. The business attributes that have such relevance will vary from company to company. The following explanations will clarify the heuristics of Figure 4 where the proposed compromises are depicted with dotted lines.

- **Temporal aggregation of event histories.** In cases where detailed transaction histories are not needed, the effect of flow events can be aggregated in attributes of economic resources or agents. Faced with the implementation of the data model in Figure 4(a) for example, an implementer could choose to not keep **cash receipts** and **cash disbursements** as separate entities and to only aggregate their effects in **cash** accounts (on the *amount* attribute, for example). Such a decision would presume no decision usefulness to the event histories, a situation that might not be warranted in an enterprise using certain types of quantitative cash management models.
- **Representation and use of a subset or superset.** In many decision cases, it makes more sense to maintain and use entities at either a more specific or a more general level than an REA interpretation might specify. Two examples of such use are portrayed in Figure 4(b). If **sale orders** become either filled or unfilled depending upon inventory circumstance, most of the decision usefulness accrues to the unfilled subset. Therefore, only that specific subset **(unfilled order)** would be declared and used. In a similar fashion, certain kinds of companies might not need to view **sales** and **cash receipts** as different entities, only as **revenue cycle transactions** that relate to a certain customer whose status is determined with periodic batching of this transaction data. That is, there may be no usefulness in modeling the resource outflow event separately from the resource inflow event where they can instead be modeled with one superset construct and coded as different transaction types.
- **Substantive non-implementation or procedural-declarative tradeoffs for entity sets.** The maintenance of some REA components can be dismissed on a substantive basis if there is no decision need for their data. Certain firms might not need inside agents, for example, if they don't track financial responsibility for costs or revenues. Similarly, if the decision use for certain types of relationship connections is infrequent, it might make more sense to materialize those connections procedurally rather than maintaining them declaratively. In Figure 4(c), the **pays for** relationship might be dismissed substantively if the company tracks payables only by amount (i.e., a balance-forward system). Similarly, the **party to** relationship is one that could be materialized monthly from the rest of the intact structure if there was no more compelling use for its maintenance.
- **Conceptual congruency of closely related entities**. Two objects can be defined as conceptually congruent if they always occur together, or in data modeling terms, if they exhibit the structural constraint pattern—(*1,1*)-(*1,1*)—shown in Figure 4(d). In such cases, a common compromise is to fold the multiple entities into one. For example, a company that received all of its **cash receipts** instantaneously from **sales** would not need separate representations for these two event sets.

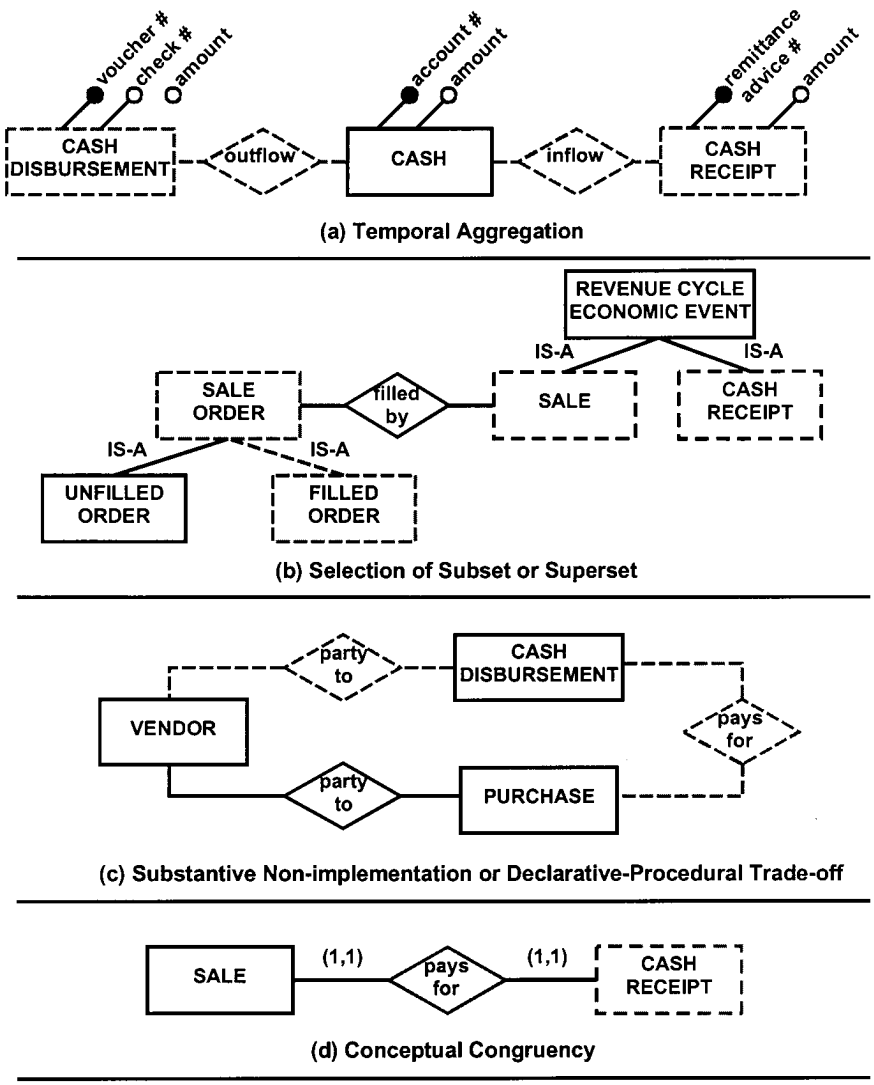

Figure 4 *Information Use* compromise heuristics

Within REACH, heuristic guidance of the type described above may be provided in two ways. First, diagnostic questions can be posed to the user after view integration. The answers to these questions would guide implementation stage suggestions. Second, knowledge of these heuristics can be embedded in the view modeling and integration stages. Such knowledge can be used to identify instances where past system designers have made implementation compromises that might or might not be warranted in present circumstances (i.e. reverse engineering of past systems).

Physical Implementation Compromise Heuristics

Another type of compromise arises from the nature of the eventual physical implementation of the accounting system. The discussion here is directed toward a traditional file-oriented implementation. However, similar discussions could be made for accounting systems using a

non-relational database model, such as networks or hierarchies. In addition, there are modifications to the relational implementation that may also arise from certain physical implementation (efficiency) characteristics. For example, in a relational database implementation, we might not choose to instantiate every relationship in our conceptual schema separately. Consider the case where a relationship has a connection cardinality of *1-to-1* or *1-to-n* (Chen, 1976). For efficiency purposes, it usually is better not to instantiate the relationship with a separate table if participation of the *n* entity is required in the relationship or if the relationship is heavily loaded (i.e. close to 100% participation). Instead, the primary key of one entity could be added as an attribute to the other entity.

Figure 5 shows examples of three types of physical implementation compromise heuristics for a file-based implementation of an REA-modeled accounting system. Such compromise is based upon the underlying concept that an accounting system can be implemented in a variety of hardware/software configurations, each of which will have specific performance characteristics and cost. These characteristics may suggest certain types of compromise that can be implemented to an REA-oriented conceptual schema for the sake of efficiency. The following explanations will clarify the heuristic depictions of Figure 5.

- **Between-Cycle, Resource-Oriented Events**. For these types of events, the item of primary interest is a resource that connects events from different accounting cycles. In this case, the resource becomes a good candidate for a master file, with the flat part of the file being the relevant attributes of the resource and the repeating groups of the file being the attributes of the actual transactions (events). For example, with **Raw Material**, we may be most interested in being able to determine the value of and changes to the resource, so we could instantiate a master file with flat fields like *RM-product#* and *quantity-on-hand* and with repeating fields like *receiving report#* (transfer in) and *requisition#* (transfer out).
- **Within-Cycle, Agent-Oriented or Claim-Oriented Events**. This compromise is similar to the between-cycle events described above, but the primary entity of interest is the agent, and the corresponding master file is based on the data components of that agent. For example, in the case of tracking acquisition cycle events, the flat part of the file would be **vendor** attributes including *address* and *payables-balance* and the repeating groups would represent individual **purchase** and **cash payment** transactions. Additionally, sometimes the imbalances between same-cycle events become master-file candidates themselves, such as happens with **debt** or **equity** master files. The judgements involved in treating these imbalances as base objects (explained in McCarthy, 1982, p.571) are embedded in the compromise heuristics.
- **Limited Dimension Resources and Events**. In this case, the entities of interest are economic resources and events where most of the information to be maintained about either type of object is financial in nature (such as a cash balance or a summarized total for a period expense). These limited dimension entities become good candidates for records in a **general ledger** master file, with facet codes being used to track non-monetary dimensions such as organizational unit or time period.

In the cases of all the physical implementation heuristics described immediately above, where the target implementation platform is a file-oriented environment, it should be apparent that the process of implementation compromise can become quite severe, leading to uncontrolled redundancy and/or significant information loss. In REACH, we intend that these disbenefits be flagged and analyzed before the heuristics are used.

CURRENT IMPLEMENTATION OF REACH

The first (partial) implementation of the REACH system is a KBS named **REAVIEWS** (**REA** **V**iew **I**ntegration with **E**xpertise from **W**ritten **S**ources—see Rockwell, 1992). It uses

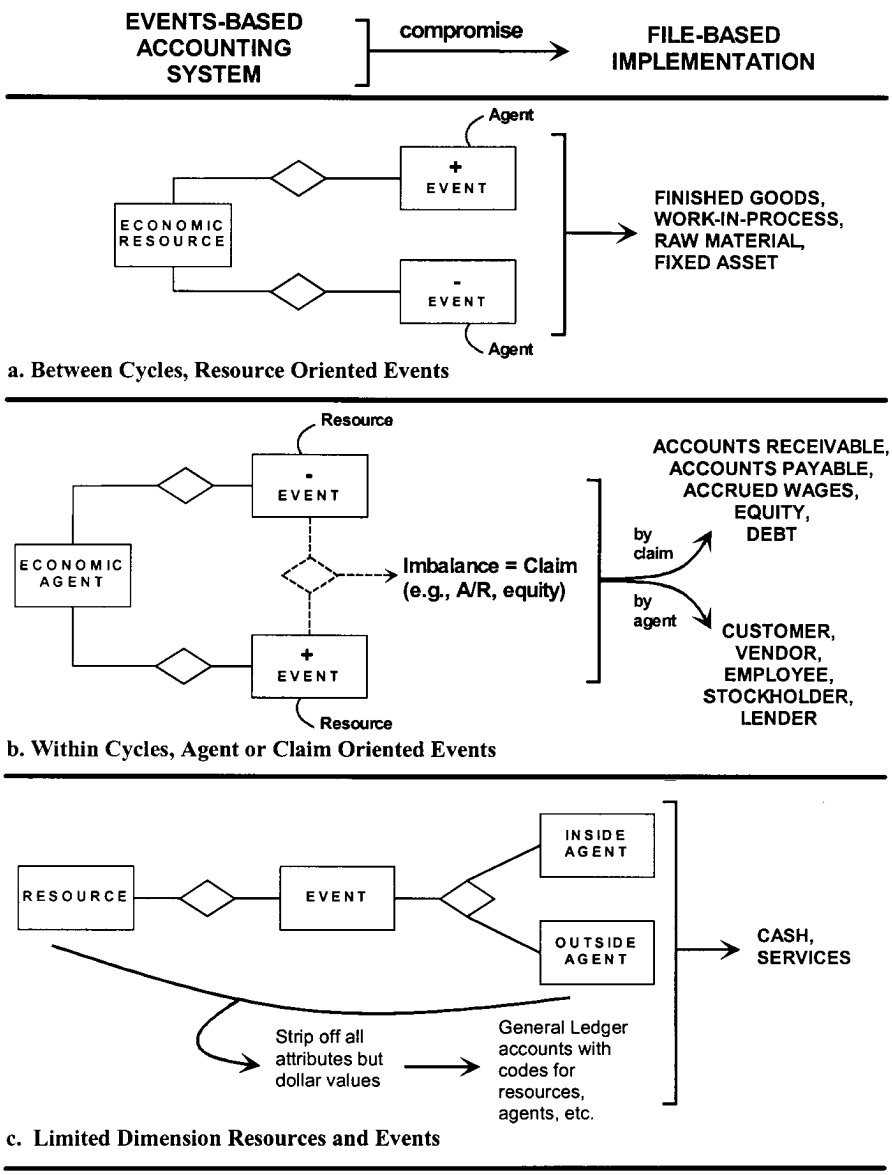

Figure 5 *Physical Implementation* compromise heuristics

the four levels knowledge depicted in Figure 2 in the design ask of view integration. REAVIEWS demonstrates the additional problem-solving ability provided by the combination of such knowledge domains. The application of database design and artificial intelligence theory in REAVIEWS provides additional insight into both the **process** of accounting system design and **methods for modeling that process** in knowledge-based systems. The use of well-developed accounting theory and the expertise of experienced accounting system designers provides insight into database design and methods for overcoming difficulties in the automation of certain database design processes.

Problem Domain

REAVIEWS's problem-solving domain is the machine shop industry, in part due to the availability of a number of modeling cases drawn from actual business enterprises in that industry. In addition, there is readily available much reconstructive accounting knowledge about such companies. This body of industry information allowed us to explore one of our major research goals—testing the proposition that using domain knowledge enhances our ability to resolve common view integration conflicts.[2] Also, manufacturing operations offer a rich business environment for the database design process, with results potentially generalizable to a wide range of actual businesses.

Accounting Knowledge in REAVIEWS

The three types of accounting knowledge described earlier are implemented in the knowledge structures and reasoning used by REAVIEWS. **Principles Level** knowledge is embodied by using REA accounting theory and templates as the metamodel for constructing the schemas used by the system. **Reconstructive Expertise** derived from the *Encyclopedia of Accounting Systems* is implemented in the **m-schema**. That schema is used as the starting point for a 'ladder' view integration strategy.[3] **Implementation Heuristics** are used in both the m-schema and in the reasoning strategies that integrate each user view into the developing enterprise schema. While not defined earlier, nor modeled as a specific type of knowledge, **company-specific** information enters the design process during integration. REAVIEWS gathers company-specific knowledge from the original user views and by querying the user during a consultation. This knowledge provides additional help as REAVIEWS refines the original m-schema into a final enterprise schema specific to the target company.

REA Theory in the Other Knowledge Levels

REA theory actually contributes to problem solving at several levels in REAVIEWS. As mentioned, REA-based 'templates' are used as the primary structures for modeling industry- and company-level knowledge in the m-schema and integrated enterprise schema. By adding more-specific domain knowledge from the industry and company levels to the more-generic REA templates, we gain the ability to make additional inferences about enterprise schema objects. Still, this lower-level knowledge is structured and constrained by use of the REA templates. REA theory is also embedded in REAVIEWS's problem solving structure. One example was described previously, in the setting where the **Employee** entity was modeled as an attribute of the **Sale** event. In that case, REAVIEWS would model the **Employee** entity. Another example occurs at the end of an integration session, where REAVIEWS uses the REA templates to identify important entities that may have been incorrectly omitted from the user schemas.

Outputs of System

When the user views have all been integrated, the internal frame-based representation must be translated into a form understandable by the users. REAVIEWS output will be the complete schema as a list of entities, attributes, relationships, and structural constraints. Figure 6 shows a short section of the output file from a sample REAVIEWS session, while Figure 7 shows an E-R diagram produced from that output. Both of those figures show only part of the actual output from a test case consultation.

SUMMARY AND FUTURE DIRECTIONS

Current-generation knowledge-based systems avail themselves of a much wider array of knowledge structures than did earlier systems, and in that sense, they allow designers to be

[2]This is not an *ad hoc* proposition, but one developed from observation of and performing the view integration task itself. For example, when resolving integration problems such as name conflicts, the analyst must determine if two entities are referring to the same real world object or not. This determination is frequently made using knowledge from the application domain.

[3]For a discussion of various view-integration strategies, see Batini *et al.* (1986).

```
The entity named SALE is of type EVENT.
Its primary-key attribute is: INVOICE-NO.
It's non-primary-key attributes are:
(TOTAL-AMOUNT DATE).
The primary-key attribute synonyms are: (SALES-NO).

The entity named FINISHED-GOOD is of type RESOURCE.
Its primary-key attribute is: ITEM-NO.
It's non-primary-key attributes are:
(PRICE).
Synonyms for FINISHED-GOOD are: (PRODUCT)

The entity named CUSTOMER is of type AGENT.
Its primary-key attribute is: CUSTOMER-NO.
It's non-primary-key attributes are:
(NAME).

The entity named SALESPERSON is of type AGENT.
Its primary-key attribute is: EMPLOYEE-NO.
It's candidate-key attributes are:
((SOCIAL-SECURITY-NO)).
It's non-primary-key attributes are:
(COMMISSION-RATE NAME).
Synonyms for SALESPERSON are: (SALES-WORKER)
The primary-key attribute synonyms are: (EMP-NO).

The relationship SALE-SALESPERSON-C is a CONTROL
relationship.
The entities joined by this relationship are:
(SALE SALESPERSON).
The structural constraints for these entities are:
((SALESPERSON 0 N) (SALE 1 1)).

The relationship CUSTOMER-SALE-C is a CONTROL
 relationship.
The entities joined by this relationship are:
(CUSTOMER SALE).
The structural constraints for these entities are:
((CUSTOMER 1 N) (SALE 1 1)).

The relationship FINISHED-GOOD-SALE-S is a
STOCK-FLOW relationship.
It's non-primary-key attributes are:
((QUANTITY)).
The entities joined by this relationship are:
(FINISHED-GOOD SALE).
The structural constraints for these entities are:
((FINISHED-GOOD 0 N) (SALE 1 N)).
```

Figure 6 Partial output from REAVIEWS

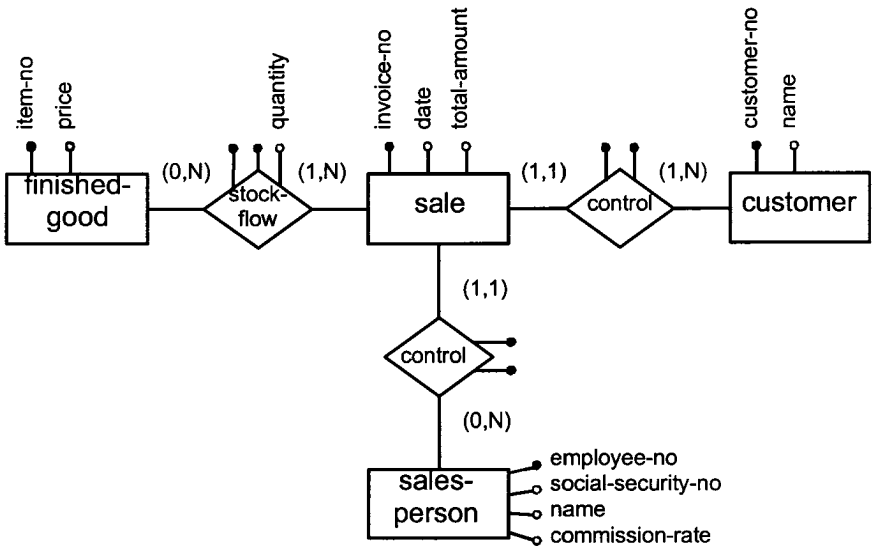

Figure 7 Partial output from REAVIEWS (in E-R format)

very ambitious with regard to final objectives. Our approach to the task for which REACH will provide consultation support—events-based accounting system design with a pervasive emphasis on suspension of implementation compromise—is one that no present human actually does. It is far too ambitious a decision environment, and it would require detailed expertise in all three areas of domain knowledge described in this paper. However, our KBS blueprint for REACH is deliberately challenging, and we intend to selectively attempt different modules as we learn more about REA-patterned problem solving.

For the present REACH prototype, we are aiming for these limited scope implementation objectives.

- Our first-order REA theory is being used during view modeling in only a forward engineering sense, as opposed to using the compromise heuristics to 'unearth' the full analysis of actual legacy accounting systems,
- We are concentrating initially on the easiest and most commonly occurring accounting transaction cycles (revenue, acquisitions, and payroll) as opposed to the more difficult ones (conversion, financing, etc.),
- Our reconstructive enterprise models are being used only during view integration and we are using just a few representative industry types,
- We are using just a limited set of implementation compromise heuristics for a limited set of cycles and delaying incorporation of others to later system versions,
- We are delaying the integration of methods knowledge (for both structured analysis and semantic database design) unless it is essential to prototype functioning.

In the future, we intend to remove many of the limitations described above. This is especially true with regard to the integrated use of the three domain knowledge types in a manner that spans all conceptual database-modeling phases. In our present design, we integrate the different types in simple pairwise fashion either in view modeling or in view integration but not in both. We also plan to integrate our system seamlessly with some structured analysis CASE tools and to use parts of some commercially available database design tools in support of our analysis and design consultations.

The REACH project is a very ambitious one.

The design or re-engineering of business accounting systems is one of the most common types of software engineering tasks, and there is a tendency to think that most of the problems in this arena have already been solved. This is decidedly not true, especially if one considers database environments where shared decision use of economic transaction data is a vastly undeveloped area. Most knowledge-based use of accounting data at present concentrates almost exclusively on account dollar analysis of the type illustrated in FSA (Mui and McCarthy, 1987). We believe that we have a methodology, a theory, and a tool to design events-based accounting systems that will far surpass the limited capabilities of computerized 'journal and ledger' implementations. The key to the automated support of such design is the integrated use of the knowledge types described in this paper.

Acknowledgements

Financial support for this paper was provided by the Department of Accounting at Michigan State University, by Ameritech, and by Arthur Andersen & Co. An earlier version of parts of this paper, prior to system implementation, was presented at The International Workshop on Expert Systems and Their Applications in Avignon, France, 1989.

References

Batini, C., Ceri, S. and Navathe, S.B., *Conceptual Database Design: An entity-relationship approach*, Benjamin/Cummings, Redwood City, CA, 1992.

Batini, C. and Ferrara, F.M., 'An integrated architecture for CASE systems', in *Proceedings of the Second International Workshop on Computer-Aided Software Engineering*, 1.7–1.12, Index Technology, Cambridge, MA, 1988.

Batini, C., Lenzerini, M. and Navathe, S.B., 'A comparative analysis of methodologies for database schema integration,' ACM Computing Surveys, **18**(4), 1986, 323–64.

Chen, P.P., 'The entity-relationship model: toward a unified view of data', *ACM Transactions on Database Systems*, **1**(1), 1976, 9–37.

DeMarco, T., *Structured Analysis and System Specification*, Prentice Hall, Englewood Cliffs, NJ, 1979.

Denna, E. and McCarthy, W.E., 'An events accounting foundation for DSS implementation', in Holsapple, C.W. and Whinston, A.B. (eds), *Decision Support Systems: Theory and Application*, Springer-Verlag, Berlin, 1987, 239–63.

Dogac, A., Yürüten, B., and Spaccapietra, S., 'A generalized expert system for database design', *IEEE Transactions on Software Engineering*, **15**(4), 1988, 479–91.

Fikes, R. and Kehler, T., 'The role of frame-based representation in reasoning', *Communications of the ACM*, **17**(3), 1985, 904–20.

Geerts, G., McCarthy, W. and Rockwell, S., 'Automated integration of enterprise accounting models throughout the systems life cycle', *International Journal of Intelligent Systems in Accounting, Finance and Management*, **5**, No. 3, September 1996, 113–28.

Gold Hill, *Goldworks III Reference Manual*, Gold Hill Computers, Cambridge, MA, 1993.

Ijiri, Y., *Theory of Accounting Measurement*, American Accounting Association, Sarasota, FL, 1975.

Lloyd-Williams, M. and Beynon-Davis, P., 'Expert systems for database design: a comparative review', *Artificial Intelligence Review*, **6**, 1992, 263–83.

Loucopoulos, P. and Harthoorn, C., 'A knowledge-based requirements engineering support environment', in *Proceedings of the Second International Workshop on Computer-Aided Software Engineering*, Index Technology, Cambridge, MA, 1988.

Loucopoulos, P. and Theodoulidis, B., 'Case—Methods and support tools', in Loucopoulos, P. and Zicari, R. (eds), *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development*, John Wiley & Sons, NY, 1992, 373–388.

Lum, V.S., Schkolnick, G.M., Jefferson, D., Su, S., Fry, J., Teorey, T. and Yao, B., *1978 New Orleans Data Base Design Workshop Report*, Research Report No. RJ2554, IBM Research Laboratories, San Jose, CA, 1979.

Mattessich, R., *Accounting and Analytical Methods*, Irwin, Homewood, IL, 1964.

McCarthy, W.E., 'The REA accounting model: a generalized framework for accounting systems in a shared data environment', *The Accounting Review*, **57**, 1982, 554–78.

McCarthy, W.E., Rockwell, S.R. and Armitage, H.M., 'A structured methodology for the design of accounting transaction systems in a shared data environment', in *Proceedings of the 1989 Conference of the Structured Techniques Association*, Structured Techniques Association, Chicago, IL, 1989.

McCarthy, W.E., Denna, E., Gal, G. and Rockwell, S., 'Expert systems and AI-based decision support in auditing: progress and perspectives', *International Journal of Intelligent Systems in Accounting, Finance and Management*, **1**, No. 1, 1992, 53–63.

McCarthy, W.E. and Rockwell, S.R., 'On the embedding of domain knowledge in automated software

engineering tools: the case of accounting', in *Proceedings of the Second International Workshop on Computer-Aided Software Engineering*, Index Technology, Cambridge, MA, 1988.
Mui, C. and McCarthy, W.E., 'FSA: Applying AI techniques to the familiarization phase of financial decision making', *IEEE Expert*, **2**(3), 1987, 33–41.
Pescow, J.K., (ed.), *The Encyclopedia of Accounting Systems*. Prentice Hall, Englewood Cliffs, NJ, 1976.
Plank, T. M. and Plank, L.R. (eds), *The Encyclopedia of Accounting Systems*, 2nd edition, Prentice Hall, Englewood Cliffs, NJ, 1994.
Reiner, D., Brown, G., Friedell, M., Lehman, J., McKee, A., Rheingans, P. and Rosenthal, A., 'A database designer's workbench', in *Proceedings of the Fifth International Conference on Entity-Relationship Approach*, North-Holland, Dijon, France, 1986.
Rockwell, S., *The Conceptual Modeling and Automated Use of Reconstructive Accounting Domain Knowledge*. PhD dissertation, Department of Accounting, Michigan State University, 1992.
Ryan, K., 'An experiment in capturing and classifying the software developer's expertise', in *Proceedings of the Second International Workshop on Computer-Aided Software Engineering*, Index Technology, Cambridge, MA, 1988.
Storey, V.C., 'A selective survey of the use of artificial intelligence for database design systems', *Data And Knowledge Engineering*, **11**(1), 1993, 61–102.
Storey, V.C., Chiang, R.H.L., Dey, D., Goldstein, R.C. and Sundaresan, S., 'Database design with common sense business reasoning and learning', *ACM Transactions on Database Systems*, **22**(4), 1997, 471–512.
Storey, V.C. and Goldstein, R.C., 'Knowledge-based approaches to database design', *MIS Quarterly*, **17**(1), 1993, 25–46.
Vasarhelyi, M.A. (ed.), *Artificial Intelligence in Accounting and Auditing*, Marcus Wiener, Princeton, NJ, 1995.
Yourdon, E., *Modern Structured Analysis*, Yourdon Press, Englewood Cliffs, NJ, 1991.