A DATABASE ARCHITECTURE FOR SUPPORTING BUSINESS TRANSACTIONS

Matthias Jarke

Jacob Shalev

Revised November 1983

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #51

GBA #83-27(CR)

To appear in Journal of Management Information Systems
Vol. 1, No. 1

A Database Architecture For Supporting Business Transactions

Abstracts

The central hypothesis of this paper is that database design and systems design in general can be simplified considerably by tailoring the design methods to a suitable range of applications. Domain-specific knowledge can be incorporated into a specialized database architecture that leaves the designer with the task to specify only the application-specific parts. Based on an analysis of business constraints, we propose such an architecture for the domain of business transaction processing.

The architecture offers several data and transaction management services, special-purpose sub-databases, and design checking rules to be used by the application designer. Two services, input management and audit and control services, are described in more detail.

Keywords:

transaction processing, semantic database modelling, database architecture, knowledge-based systems design methods

1.0 INTRODUCTION

Large-scale transaction processing has become the backbone of many business information systems. Therefore, it is of paramount importance that business transactions are processed in a safe, efficient, and traceable manner.

Often, transaction processing relies on the use of database management systems (DBMS) for storing and retrieving data. However, many generalized DBMS do not support this application very well. Unnecessary amounts of input have to be retyped in case of errors, processing must be repeated if output gets lost, and auditing facilities have to be hand-programmed instead of being part of the DBMS. It is our perception that one of the main reasons for these problems is a deficiency of dynamic and domain-specific concepts in current database management systems.

Traditionally, database research has focused on the essentially static view of a database as a collection of state descriptions. Only recently, a number of researchers have been trying to incorporate a more dynamic perspective into database systems, either by embedding the concept of history (a sequence of states) into database models [Ariav and Morgan 1982, Clifford and Warren 1983, De Antonellis and Zonta 1981], or by modelling change directly using transaction concepts [Borgida et al. 1982, Gray 1981, Rolland and Richard 1982].

Furthermore, systems analysis and design methods for business transaction processing systems seem to suffer from a lack of semantic knowledge about their application domain. Tens of thousands BTPS have been implemented, yet design methods display striking weaknesses:

1. They neither incorporate past experience nor do they draw upon a common base of knowledge. They require experienced people but do not support knowledge accumulation.

2. They do not use standard requirements, even for "standard" applications (like Accounts Payable, General Ledger, Personnel, etc.). Note, that "standards" here applies to the application itself and not to the methods of its specification (like data flow diagrams, structure charts, etc.).

3. They do not use pre-fabricated application-oriented components at the design level or at the software level.

4. They do not recognize standard operations (e.g. error checking) and thus may not use pre-fabricated components even if they exist.

The lack of these features forces the designer and the user to refine their design to a very detailed level, making it virtually impossible to cover all details and aspects of the system consistently. Even the structured methods (e.g., [DeMarco 1978]) do not directly address the above problems. In a way one could regard them as "syntactic", whereas we point out the lack of a "semantic" knowledge base, and of tools based on application knowledge.

The central idea of our approach is that such a knowledge base cannot be developed for information systems in general. It is necessary to focus on a generalized application domain (such as business transaction processing) to capture knowledge that is specific enough to really support the systems analysis process. To understand this point, consider how the knowledge domain is enriched if one zooms in from a requirements analysis of editors in general to one for word processors in an office environment: many necessary features of word processors (e.g., spelling correction, letter formatting) may be meaningless for editors in general.

In this paper, we outline a DBMS architecture that overcomes some of the limitations by introducing dynamic concepts and semantic knowledge about a generalized application domain, business transaction processing systems (BTPS). This semantic restriction allows much more specific design guidelines and supporting software systems to be used than in a general operations database having just a broad process concept such as described, e.g., in [Bradley 1978].

In the proposed architecture, the state-describing database is augmented by a transactions base, consisting of sub-databases for input, output, control, and audit of transactions, and of generalized services that allow the various sub-databases (and the human users) to communicate efficiently. In addition to these structural components, the architecture contains business rules derived from the specific purposes of business transaction processing; they serve as guidelines and checking procedures for the design of specific applications.

The paper is organized as follows. Section 2 defines the concept of business transaction and studies some domain-specific requirements from which business rules can be derived. Section 3 presents an overview of the proposed architecture. Two major components, input management and control are described in more detail. The conclusions report some preliminary experience and outline future research directions.

2.0 REQUIREMENTS FOR BUSINESS TRANSACTION PROCESSING

2.1 Business Transactions

At the operational level, a business is set up to carry out certain business transactions. An important property of the operational level is that there are usually a large number but only a small variety of business transactions. Speaking in programming language terms, one can define a small number of transaction types. Such a type definition will be called a business program to stress the fact that it is governed by the specific rules of business transaction processing to be detailed later. Essentially, the business program defines (planned) processes together with a script that defines the relationship among these processes. A planned process can be further refined into subprocesses; atomic subprocesses are called activities. The distinction between processes and activities is left to the discretion of the system designer. Processes of the same type may occur in multiple higher-level processes.

Each business transaction -- instantiation of the business program -- is composed of (actual) processes which can be further refined downto the level of (actual) activities. In contrast to transactions in the conventional database sense, business transactions may contain parallel processes, are long-lived and nested [Gray 1981]: there may be sub-transactions that have to commit before the end of the business transaction since an activity gives up control over an important resource that cannot be reclaimed without explicit counter-transactions if at all.

As an example, consider an Accounts Payable department that receives invoices, approves and pays them. In the invoice approval process, an approver enters the invoice data. It is checked by the computer against the purchase order (P.O.) data. If approved, the total-amount-approved for the P.O. is increased, as is the total-amount-approved for the vendor. Also, a payment voucher is prepared, and a record that will be sent to Headquarter's central computer.

This subprocess is part of the payment process that includes the activities of printing the check on the voucher's due date, and later on, the check reconciliation. The payment process is part of the overall business program set up to handle purchasing. Note, that once a check is out and paid, there is little chance to get the money back (= reset the payment sub-transaction) without major corrective action.

2.2 Related Research

The concepts of business transaction and business program are related to some recent work on semantic data models [Hammer and McLeod 1978], abstract data types [Borgida et al. 1982], transaction modelling [Rolland and Richard 1982], and data modelling in transaction-based decision support systems [Jarke 1982] which also stresses the importance of general transaction knowledge. However, our concept is more general in that it assumes the combined use of human and computerized processors, and it is more specialized in the sense that it incorporates knowledge about the requirements of operational level business systems.

Time-related concepts as referred to in the introduction can serve important purposes in a BTPS but are not yet sufficiently developed in practice. A history of states is a useful tool for time-stamp based concurrency control and for providing a description of previous states of the business. But a BTPS also needs a history of changes (what are they? who made them? when? why?).

The transaction concept offers consistency of mapping a single transition between two states of the real world. It also ensures atomicity and durability of the changes made to the database [Gray 1981]. However, it does not cover the fact that business transactions are a joint venture between human and computerized processors, or that they are long-lived and nested. Research in nested transactions is just in the initial stages [Ries and Smith 1982].

Some enterprise-level requirements analysis methodologies such as BIAIT [Burnstine 1979, Carlson 1979, Welke and Kumar 1980], and BICS [Kerner 1979, Zachman 1982] attempt to use prior knowledge to find out what information processing subsystems an enterprise may need. This is done by analyzing the types of orders the business handles and can be viewed as high-level business transaction analysis. The detailed systems analysis, however, charged with specifying each of the chosen information systems, uses syntactic tools such as data flow diagrams [DeMarco 1978], assembly line diagrams, and Warnier/Orr diagrams [Warnier 1981, Orr 1981]. The basic units of analysis on this level are data flows (structures) and data transformations. No attempt is made to exploit the transaction concept and the domain-specific knowledge of operational level systems.

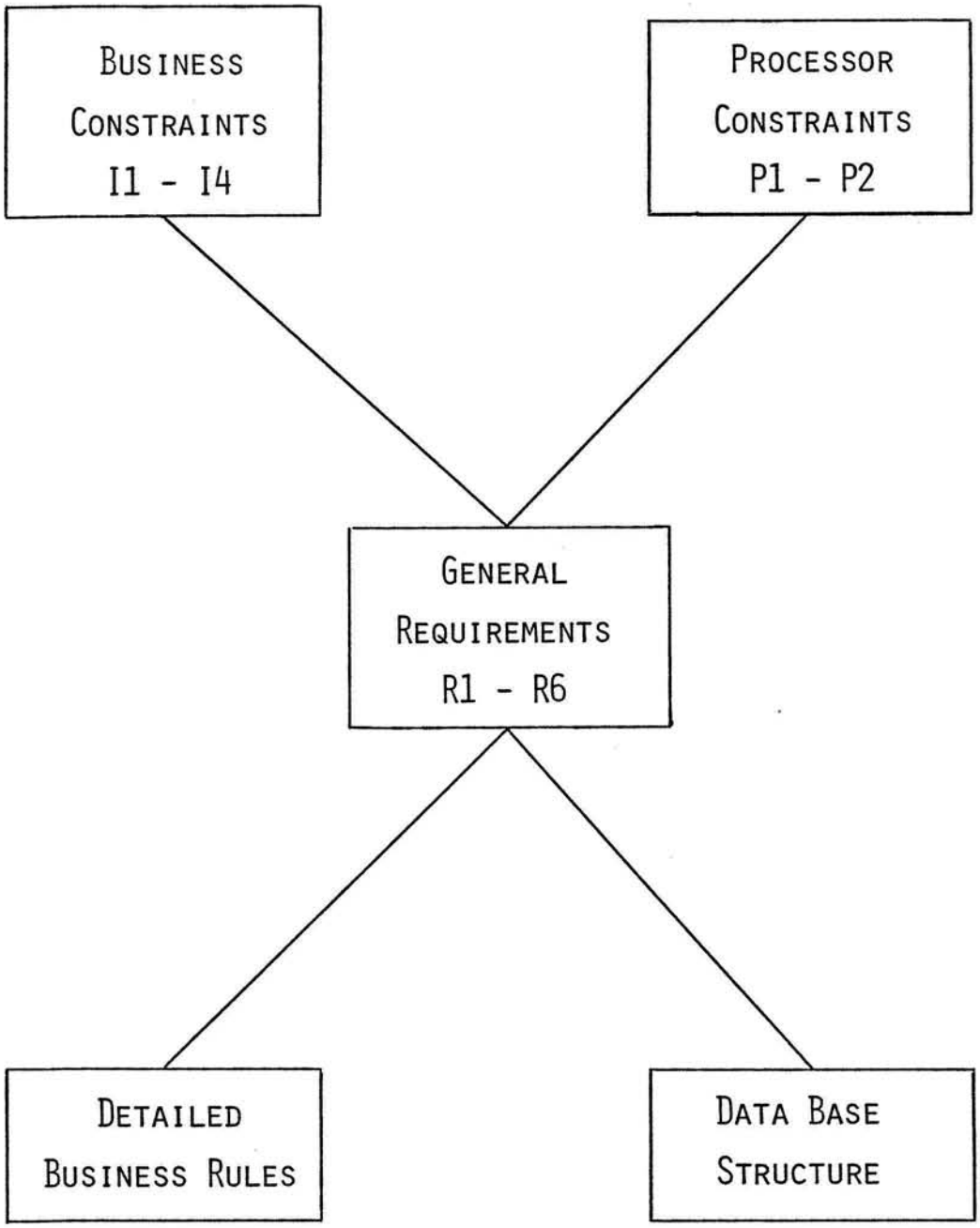


FIGURE 2-1: THE DERIVATION PROCESS OF BUSINESS REQU

2.3 Requirements Analysis For BTPS

A base of knowledge can be generated by analyzing common requirements of BTPS in a top-down procedure such as indicated in Figure 2-1. The underlying idea is that a BTPS is a tool for mass production of information which has to be efficient and precise (business constraints I1-I4) despite the presence of error-prone human and computer processors (processor constraints P1-P2). From these conflicting constraints, a set of general requirements (R1-R6) can be derived that each BTPS should satisfy (not necessarily other types of application systems, e.g., decision support systems).

***** INSERT FIGURE 2-1 ABOUT HERE *****

The requirements can be further refined to detailed rules to be used for checking a proposed design [Jarke and Shalev 1983]. Furthermore, an extended database architecture will be introduced that systematically enforces satisfaction of some of the requirements.

Similar to all "knowledge engineering" tasks, the derivation of requirements and detailed rules from business and processor constraints is not easily formalizable but rather represents a collection of aquired experience similar to the one used in an expert system [Clifford et al. 1983]. Figure 2-2 displays the main relationships presented in [Jarke and Shalev 1983]. In the sequel, a brief summary of the main business constraints, processor constraints, and general requirements will be given.

Business constraints:

I1. Quality of the transactions' outputs. In a BTPS environment, the business program design has to make sure that all necessary outputs are produced, and are correct and precise. This is more central to BTPS than, e.g., to decision support systems, not only because customer relations are at stake, but because serious errors can threaten the business' existence. Hence the commonly found business rules, signatures requirements, authorizations, and other quality control devices.

I2. Timeliness. A business transaction may be long lived, but has to be completed within a predetermined time period. The time period is either determined contractually (e.g. delivery dates, net 30 payment, etc.), or by laws and regulations (e.g. tax returns), or as a performance goal of the business (e.g. fast service). An overall transaction performance goal may not be sufficient. The business may elect to establish time constraints for critical processes. These need to be monitored, and the business program must contain elements of follow up and exception handling.

I3. Accountability. The business is held accountable for its activities by its clients, personnel, the law, and the shareholders. In the short term, the business will have to explain its actions regarding current transactions in process (for instance, when it pays less then the amount invoiced it will have to explain the deduction). Thus, the business program must be designed to provide the capability for answering outside questions. The long term accountability for the operational level business system is primarily manifested in reporting and auditability requirements. Also related to accountability are security and privacy requirements.

I4. Responsiveness. BTPS must be designed in a way that actively supports the accomodation to changing requirements. While the changes may be less rapid than in a decision support system the business program must still contain enough flexibility in itself to adapt to minor aberrations. In the long term, the supporting software tools must be powerful enough to support changes to the business program itself.

The above business issues are not the only constraints imposed on the business program. Once it is recognized that transactions are processed by two types of processors: people and computers, their specific strengths and weaknesses have to be taken into account.

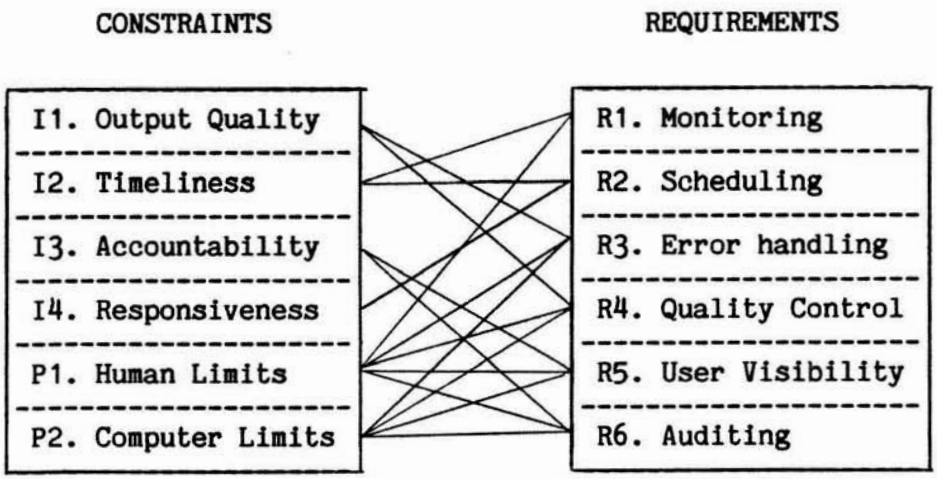


Figure 2-2: Deriving Requirements from Business and Processor Constraints

Processor constraints:

P1. The human processor. People make mistakes when processing documents. They err in performing decision rules and routine tasks. Their document storage and retrieval abilities are limited, causing lost and misplaced documents.

P2. The computer. Computers bring a new source of errors into business transactions. They also lack the (limited) integral quality control capacity of humans: common sense. Transformations of data into and out of the computer are an additional cause of errors and difficulties. The computer provides only that flexibility and data access that has been designed into it, thus inadequate design may severely limit users' control and data visibility.

***** INSERT FIGURE 2-2 ABOUT HERE *****

In their combination, the above constraints (I1-I4,P1-P2) lead to general requirements the design of any BTPS must resolve to compensate for these constraints, as indicated in Figure 2-2. For example, the limitations of human processors (P1) in a complex environment will endanger the satisfaction of the need for timeliness (I2) unless some specific action is taken to ensure it; this leads to the requirement of monitoring (R1) in any BTPS. Note once more, that this need is less urgent in decision support systems working typically with a single user without stringent time constraints.

Specific monitoring design checking rules would be aimed at satisfying the timeliness objective in the presence of complexity and processor imperfections. For example, the following monitoring questions should be answerable in a BTPS: in which process and activity is transaction X? what information is activity Y processing? what transactions do not meet time limits imposed on activity X, or on process Y, or on the whole program? what transaction or activity is in a special status (hold, urgent)?

General requirements:

R1. Monitoring is the ability to know for each transaction, in what process/activity it is and conversely, what a certain activity is doing. This must be compared against deadlines set by the business or outside constraints. Aspects of monitoring databases have been studied in [Buneman and Clemons 1979].

R2. Scheduling and control is the ability to alter the direction of flow, or the order in which transactions are processed. A business program lacking these features is inflexible. Since transactions are performed concurrently, input and output queues, and the need for their management as an integral part of the business program arise (e.g. can we recover a queue of input documents?). Problems of scheduling have been addressed both in the operational research and computer science literature; however, the application of this collected knowledge requires an appropriate systems environment.

R3. Error handling: Whereas in other system types an error may just prevent successful completion of a transaction, in a business transaction environment it may have additional adverse effects. The processes must therefore be designed to actively detect and eliminate errors, with an emphasis on effective error presentation and correction.

R4. Quality control takes into account the inability of the business program processors (P1-P2) to detect all errors. In high risk situations, quality control activities will check outputs, and may require compensating transaction types to be added to the business program.

R5. System visibility and user control: The business program should have tools that will answer at least the same user questions that could be answered in a manual system. Besides the well-known need for a user-visible data dictionary, similar devices are also required for the dynamic aspects of the system.

R6. Auditability is the ability to take a certain database state, or some output, and trace back. How was it arrived at? What activities modified it? Who did what, and when?

The next step in Figure 2-1 would be the derivation of detailed rules (see examples of monitoring questions, above). We skip this step here and proceed directly to the description of a database architecture for supporting the general requirements.

3.0 OVERVIEW OF THE DATABASE ARCHITECTURE

Conventional database design models data and transactions often independent of the information use outside the computer [Rolland and Richard 1982]. Design is typically a tiresome iterative process many details of which are repeated for each application of similar type.

In the previous section, an attempt was made to describe a knowledge structure of requirements for BTPS. Some of these take an application-specific form when actually used in design -- there are application-dependent answers to the design checking questions. However, a major portion is common to all BTPS. In this section, this common denominator will be exploited for developing an extended database architecture that allows the system designer to concentrate on details of the remaining application system. We thus propose an improved design process that will be comprised of two parts:

1. the use of a design environment of generalized business program services and sub-databases that will support the general requirements outlined in the previous section, and will be available for any business program.

2. the design of application-specific elements unique to each business program; here, business rules can be used only to evaluate the design.

In the remainder of this section, an overview of the design environment is given. Two major subsystems will be analyzed in more detail in subsequent sections.

One of the main problems of a BTPS is to get work done in the presence of errors. An input management service provides the capability to handle input documents without interrupting processing requiring unnecessary data re-entry. The documents are stored in an input database.

Output of a database transaction can be used in multiple different forms and may have to be reproduced later. Output management provides the service of maintaining and presenting output data using an output database. This sub-database can be seen as a generalization of the idea of storing computed relations for future reference in query optimization [Finkelshtein 1982].

Between input and output, transformation management roughly covers the functions of conventional database transaction execution and supervision, with a few functions added for modifying the status of the sub-databases when the main database has changed. In addition, there are sub-databases for the control and later audit of transactions. The control database offers the user system visibility and a limited amount of interrupt facilities through control services. The audit database permits ex-post tracing of transactions. Access services must be provided to all of the sub-databases with appropriate restrictions (e.g., no changes to the audit database).

The architecture is summarized in figure 3-1. In the subsequent sections, the designs of input management services and of the control database are investigated in more detail and the function of these services to support crucial business requirements is shown.

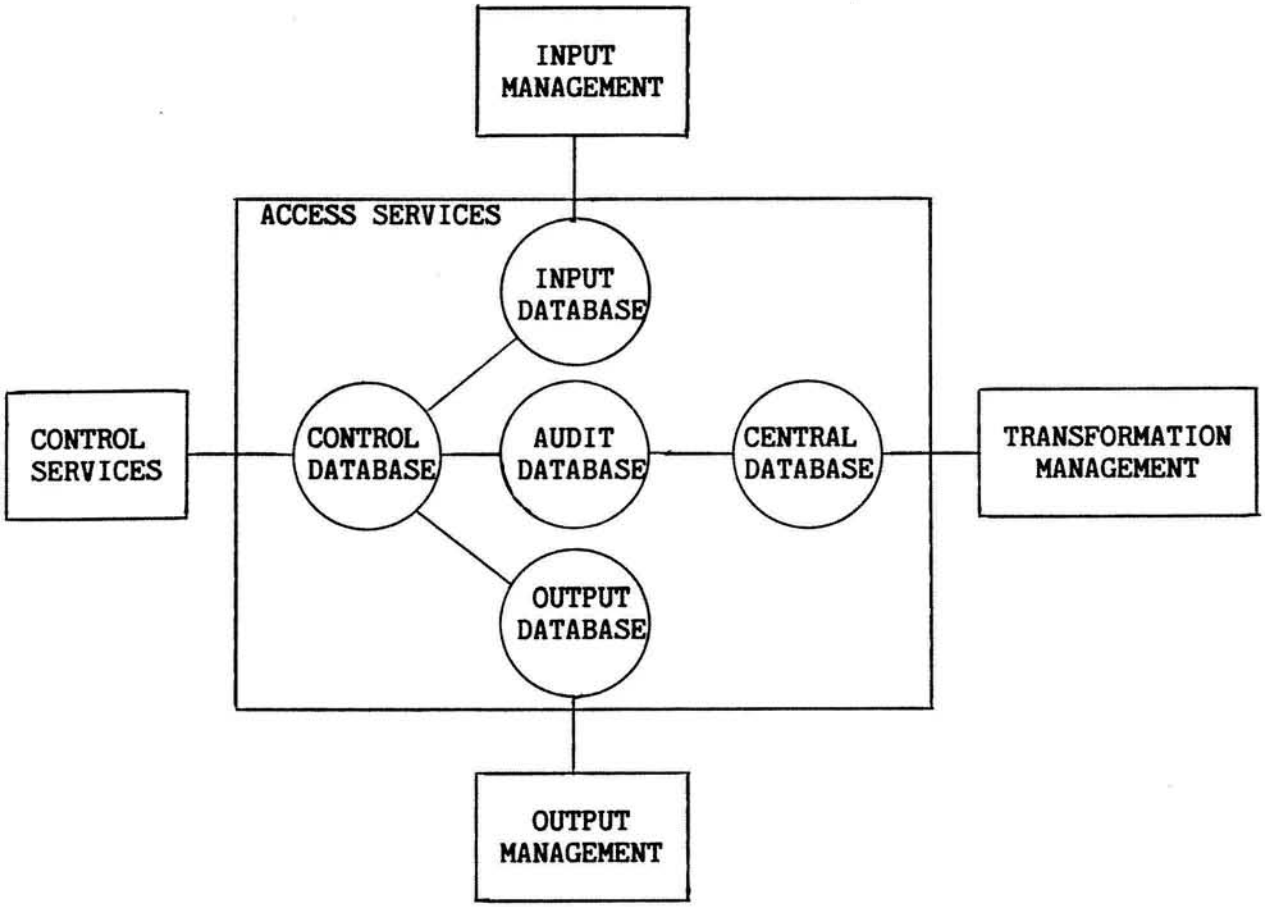


Figure 3-1: Transaction-Oriented Database Architecture

***** INSERT FIGURE 3-1 ABOUT HERE *****

4.0 INPUT DATABASE AND INPUT MANAGEMENT

4.1 Input Document And Error Checking

A BTPS input document can be defined as a hierarchy of record types. Purchase orders, invoices, receipts, checks, packing lists, and vouchers can all be described as hierarchical data structures, typically with few levels and record types. This model can be expanded by a representation of possible errors in a document. Our architecture would recognize four types of error checking.

1. identify: Check if this document is of the expected type. Otherwise it cannot be identified nor further processed by the system.
2. verify: check the attribute values against their data types and other domain restrictions (e.g., amount not greater than 10,000).
3. cross verify: check the relationship of attributes to other attributes in the same document (e.g., balance totals).
4. validate: check attributes against the database and update rules (e.g., referential integrity).

In the input document definition, a list of error codes extends each record type. A cross verify error type will be defined for the lowest common predecessor in the hierarchy of the attributes involved. Consequently, the place for identification error codes is in the root record type of the hierarchy. The root also contains a document number and a user identification.

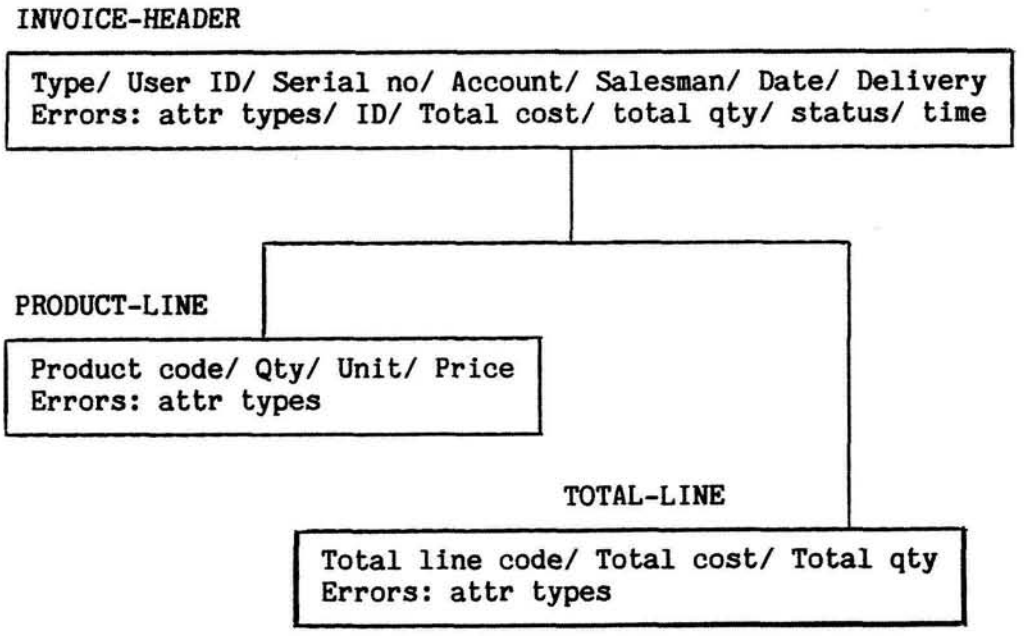


Figure 4-1: Invoice Data Structure -- Example of a Hierarchical Input Document Description with Attached Error Codes

***** INSERT FIGURE 4-1 ABOUT HERE *****

An example of the generalized input document data structure is provided in figure 4-1. Note, that the errors for total cost and quantity in the TOTAL-LINE record type are verify errors (e.g., data is not numeric) whereas the corresponding cross verify errors (computed total does not match the value given in TOTAL-LINE) are defined in the INVOICE-HEADER.

4.2 Input Document States

The management of input documents is aided by defining states of input documents, and by storing them along with time stamps. The states are stored in the root of the document or in the control database. The status of a document serves as a basis for deciding what should be done next with the document, as well as what should not be done with it. A document can be one of in the following states:

1. New document - The document was entered but not checked yet.
2. Modified document - The document was modified. Previous status is not relevant. No checking took place (after modification).
3. Verified document - Document identified, verified and cross verified successfully.
4. Verification error - Document failed verification. Errors codes are stored in the document.
5. Selected document - For update by the transformation processor.
6. Updated - The document did successfully update. It is retained for audit trail.
7. Update error - Update failed due to validation errors. Error codes are stored in the document.

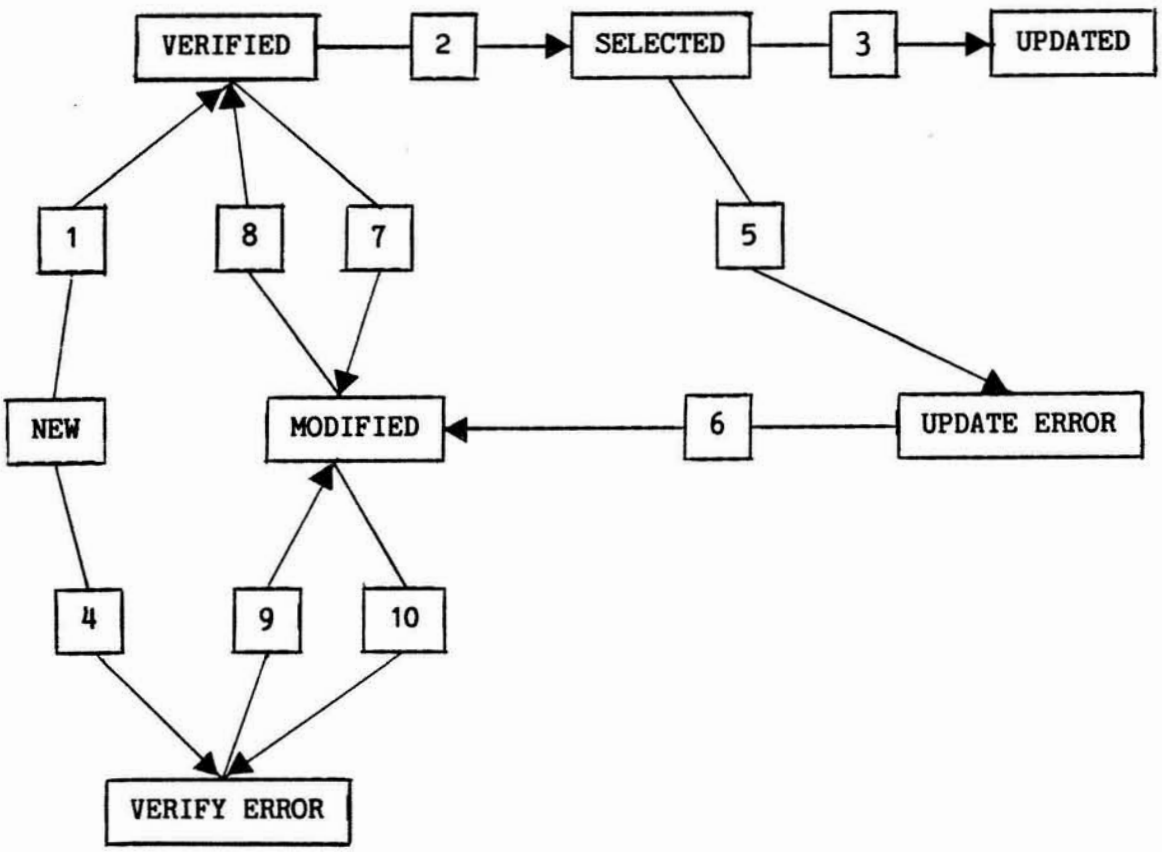


Figure 4-2: Transition Graph for Input Document States

Figure 4-2 describes the state transition graph. The arcs are labelled with numbers of explanation notes:

1) New document passes verification.
2) Verified document is selected for processing.
3) Selected document updates successfully.
4) New document fails verification.
5) Selected document fails update due to validation errors.
6) User modifies invalid document.
7) User modifies a verified document before it is selected for update.
8) Modified document passes verification.
9) User corrects a document that had verification errors.
10) Modified document fails verification.

***** INSERT FIGURE 4-2 ABOUT HERE *****

4.3 Input Management Services

We conclude this section with a summary of the services input management provides the user with for working on the input database.

An input document editor facilitates entry and modification of input documents. It performs identification, verification, and cross verification. The results (inputs and errors), are stored in the input database document structure. The editor may be batch, online, or it may be located at an intelligent remote unit.

Error reporting presents an erroneous document to the user. This takes the place of error messages distributed along program code and allows for standard error presentation in both batch and online environments.

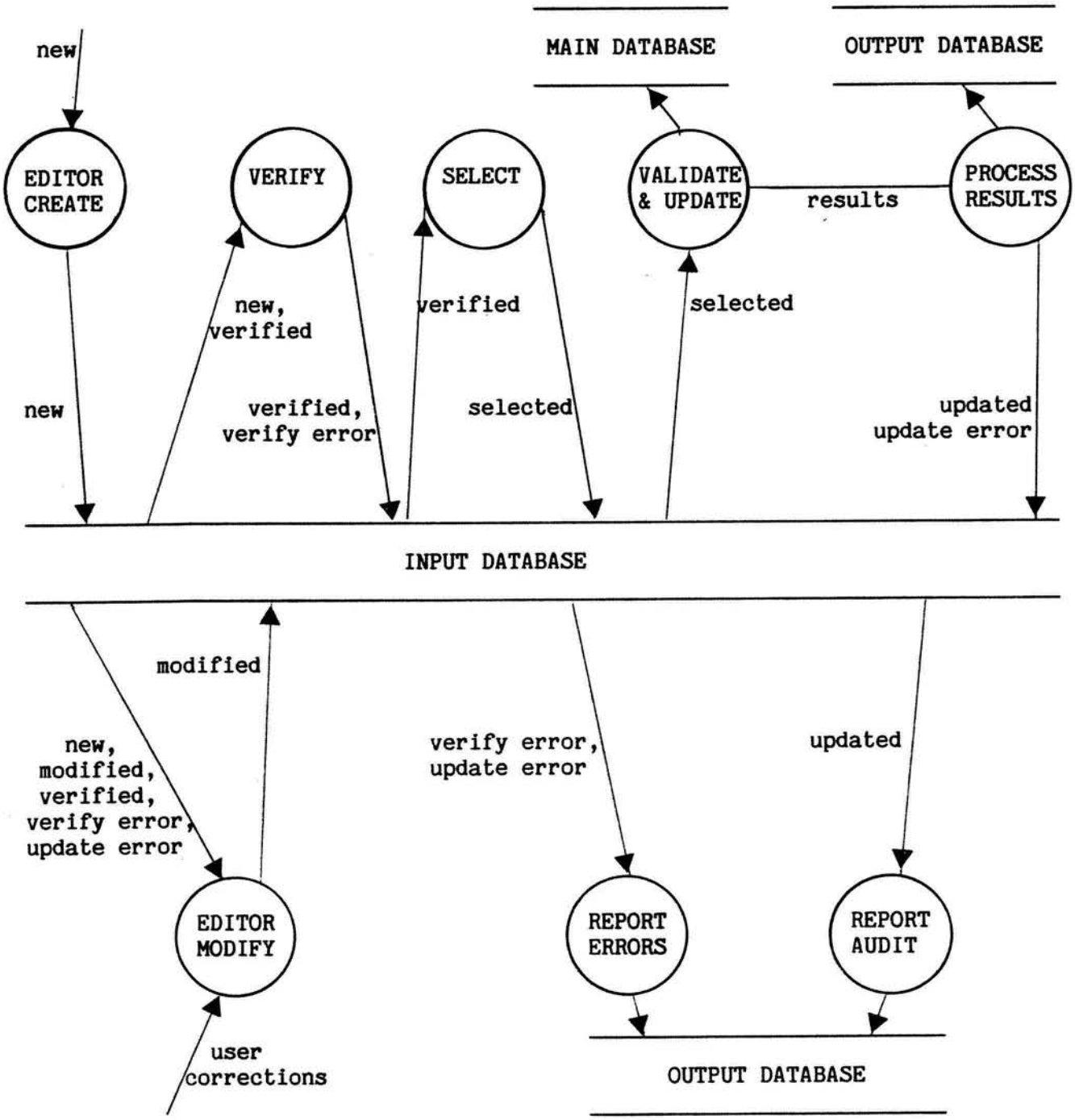


Figure 4-3: Prototype Data Flow Diagram for Input Management

Explanation of symbols:
capital letters - databases, document transformations
small letters - external activities, document states

Note that these services offer a bridge between office automation and data processing by allowing each discipline to receive and display the contents (and errors) of documents.

Two other services link input management to transformation management. The selection function chooses input documents for database update based on their status. The results function returns the results from transformation management that are relevant for input management, namely validate error messages and new document statuses.

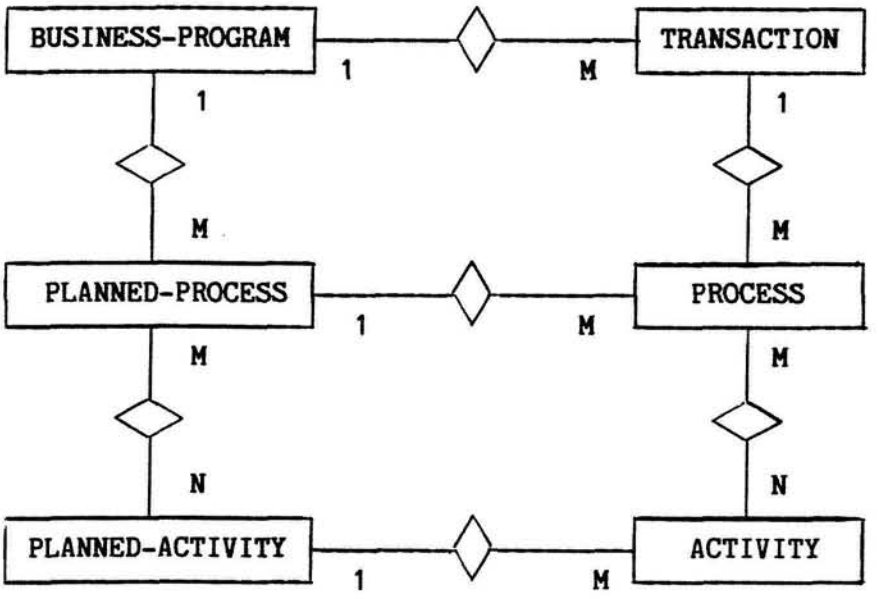
We can now tie each input management service to the related states. Each service has allowable input states and possible output states. Figure 4-3 summarizes this discussion by providing a prototype data flow diagram for input management services. This proposed state space can be further expanded to support input management in the various environments of batch, data entry, interactive and distributed input management.

***** INSERT FIGURE 4-3 ABOUT HERE *****

5.0 CONTROL DATABASE AND SERVICES

5.1 The Control Database

The control database contains data about the business program (with its planned processes and planned activities), and the business transactions (with their actual processes and activities). Figure 5-1 gives a partial entity relationship diagram.



Entity Attributes:

```
BUSINESS-PROGRAM (Name, Performance goals, Responsible user)
PLANNED-PROCESS  (Name, Performance goals, Responsible user)
PLANNED-ACTIVITY (Name, Performance goals, User, Predecessors, Successors)
TRANSACTION (<as BUSINESS-PROGRAM>, States, Timestamps, Priority, Flags)
PROCESS     (<as PLANNED-PROCESS>,  States, Timestamps, Priority, Flags)
ACTIVITY    (<as PLANNED-ACTIVITY>, States, Timestamps, Priority, Flags)
```

Figure 5-1: Control Database: Partial Entity-Relationship Diagram and Entity Attributes

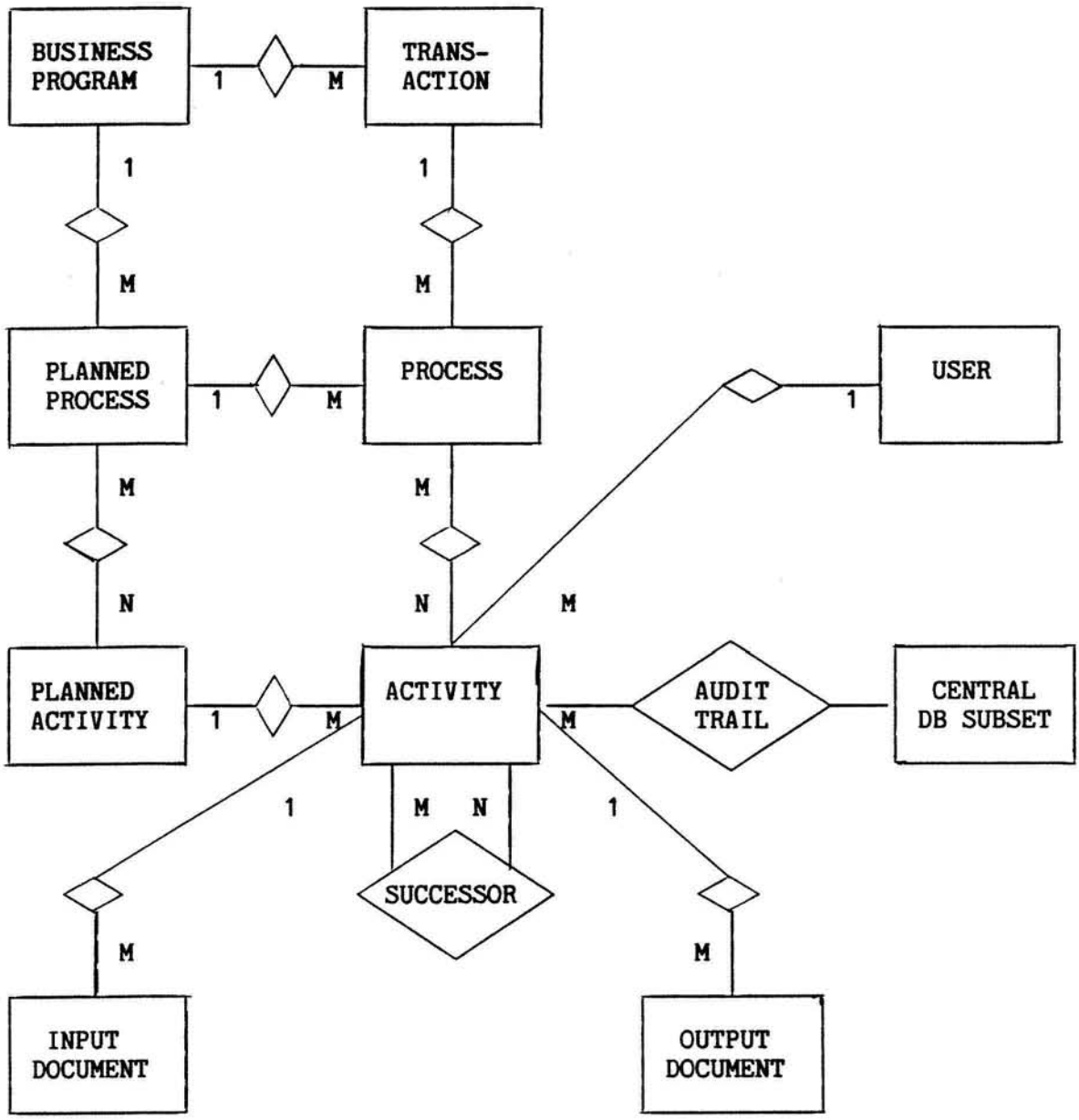


Figure 5-2: Control Database: Structure and Relationship to other Sub-Databases

***** INSERT FIGURE 5-1 ABOUT HERE *****

Note that a planned activity relates to its predecessors (whose completion enables it) and to its successors (the ones it enables). The transaction-control database records the actual occurences of the transactions, processes and activities. The states are defined per application and reflect measures of completion and exception situations. They are updated along with timestamps as the transactions go through execution.

We now extend the initial model of Figure 5-1 to a complete entity relationship model of the control database (see Figure 5-2). The activity entity becomes the focal point of the control database, as it is related to the input documents (in the input database) and the output documents (in the output data base) that took part in the execution of this activity instance. These relations are established and maintained at input entry (and edit), and at output generation time.

***** INSERT FIGURE 5-2 ABOUT HERE *****

Each activity instance is related to a "user" identified by a pair of names: one for the person responsible for the activity, and the other for the processor. In the case of a computer processor, the latter may be a computer identifier (we may have distributed processors) and a program name. Although it is not shown in Figure 5-2, one can expand the user entity into a model of the organization and the processor network.

5.2 Control Services

These services are necessary to satisfy the requirements of user visibility and control, monitoring and scheduling, and supporting quality control.

We first observe that the attributes of the business program database are repeated in the transaction control database. This allows inheriting the general plan as a default, but at the same time provides the flexibility of modifying the plan, by tailoring it for each transaction whenever necessary. We can thus handle exceptional situations like "rush", "hold", meet unexpected deadlines, while preserving the same control structure and promoting the use of uniform monitoring tools.

This model allows the assignment of activities to other than the normally planned processors (people or computer) without losing sight of who is doing what. This may be necessary for work load balancing or situations of unavailable processors. Entities may be assigned special flags to characterize special, application specific situations, and to alert the processors.

An activity will inherit the predecessors and successors from its corresponding planned activity. However, these relationships may be modified. The main reason for this feature is to support quality control and risk-reduction activities. Thus if certain conditions arise (e.g. an invoice for over $5,000), the normal activity sequence will be altered (e.g. the invoice is routed to an auditor for verification). These routing changes can be initiated by computer as well as by people, and they remain documented.

The control database can facilitate automatic scheduling of activities. People can get "action items" at their work stations, and computer programs can be initiated, as both input and output databases (which serve as queues) are available and known, as are the topology of the business program and the performance goals.

5.3 The Audit Trail Database

Our brief discussion of auditability requirements was business program oriented. Rather than the data oriented approach taken by logging and time-domain addressing, we concentrate on the dynamic aspects: how did we arrive at a certain value, what activity modified it, who did, when, what output was produced, etc.

We therefore propose a simple yet powerful audit trail database which relates some central database objects of interest to activity records in the control database. By telling the DBMS what entities and relations are to be tracked, and what activity every database alteration belongs to, the audit-trail can relate a modified data value to the record (in the control database) of the activity that modified it. These new relations comprise the audit trail database.

As shown in the previous subsection, the activity is related to its inputs and outputs so that a full picture of the business activities behind the evolution of a database state emerges.

6.0 CONCLUDING REMARKS

In this paper, we outlined a semantically enriched database architecture for business transaction processing systems that combines the ideas of dynamic (transaction-oriented) database management, and of domain-specific information systems structures found in some modern approaches to information requirements analysis. An early version of an important part of the proposed architecture, input management, has been implemented and is being used in the development and operation of defense BTPS in several countries.

For the systems developer, such an architecture provides a way to bridge the gap that still exists between high-level information requirements analysis and detailed systems design methods such as structured design and programming. Since many business-oriented services will be provided with the DBMS, the size of application programs can be expected to shrink considerably. Also, a set of detailed business rules can be derived from the general requirements described in this paper to check systems design on a high level (where the most serious errors occur!).

An informal test of our design methodology was conducted with a group of graduate students who were asked to evaluate a order entry system design proposed in a textbook. While they failed to detect any major problems, the application (by the same students) of BTPS design checking rules developed independently of that example revealed a number of grave omissions and errors in the design. A more formal evaluation of the method will be required once the design methodology is sufficiently developed and the database architecture implemented.

From this point, three research directions are pursued. First is the formalization, detailed design, and prototype implementation of the proposed database architecture. Second, a systems analysis and design procedure using the architecture is developed; a flexible structure is required that allows extensions of the knowledge base. Finally, we are researching the language definition and implementation of generalized access services that offer visibility not only of the static data and their histories, but also of the changes and transaction states.

Acknowledgments

Thanks are due to Ted Stohr for helpful discussions during the early phases of this work, and to the anonymous referees for their detailed comments that greatly helped improve the presentation.

References

1. Ariav, G., Morgan, H.L.: MDM: Handling the time dimension in generalized DBMS, Working Paper, Department of Decision Sciences, The Wharton School, University of Pennsylvania, May 1981.

2. Borgida, A., Mylopoulos, J., Wong, H.K.T.: Generalization as a basis for software specification, in M. Brodie, J. Mylopoulos, J.W. Schmidt (eds.): Perspectives on Conceptual Modelling, Springer 1983.

3. Bradley, J.: Operations data bases, Proceedings 4th VLDB Conference, Berlin 1978.

4. Buneman, O.P., Clemons, E.K.: Efficiently monitoring relational databases, ACM Transactions on Database Systems 4, 3 (1979).

5. Burnstine, D.: The theory behind BIAIT, Business Information Analysis and Integration Technique, BIAIT International Inc. 1979.

6. Carlson, W.: BIAIT - the new horizon, Database, spring 1979.

7. Clifford, J., Jarke, M., Vassiliou, Y.: A short introduction to expert systems, Database Engineering 6, 4 (1983).

8. Clifford, J., Warren, D.S.: Formal semantics for time in databases, ACM Transactions on Database Systems 8, 2 (1983).

9. De Antonellis, V., Zonta, B.: Modelling events in data base applications design, Proceedings 7th VLDB Conference, Cannes 1981.

10. DeMarco, T.: Structured Analysis and System Specification, Yourdon 1978.

11. Finkelshtein, S.: Common expression analysis in database applications, Proceedings ACM-SIGMOD Conference, Orlando 1982.

12. Gray, J.: The transaction concept: virtues and limitations, Proceedings 7th VLDB Conference, Cannes 1981.

13. Hammer, M., McLeod, D.: The Semantic Data Model: A modelling mechanism for database applications, Proceedings ACM-SIGMOD Conference, Austin 1978.

14. Jarke, M.: Developing decision support systems: a container management example, International Journal of Policy Analysis and Information Systems 6, 4 (1982).

15. Jarke, M., Shalev, J.: A knowledge-based approach to the analysis and design of business transaction processing systems, May 1983, NYU Working Paper Series, CRIS #53, GBA 83-52 (CR).

16. Kerner, D.: Business Information Characterization Study, Database, Spring 1979.

17. Orr, K.: Structured Requirements Definition, Orr&Associates 1981.

18. Ries, D.R., Smith, G.C.: Nested transactions in distributed systems, IEEE Transactions?on Software Engineering 8, 3 (1982).

19. Rolland, C., Richard, C.: Transaction modelling, Proceedings ACM-SIGMOD Conference, Orlando 1982.

20. Warnier, J.: Logical Construction of Systems, Van Nostrand Reinhold 1981.

21. Welke, R., Kumar, K.: An "ERA"-based analysis support system for BIAIT, ISRAM Working Paper WP-8010-2, McMaster Univ., Ontario, October 1980.

22. Zachman, J.: Business Systems Planning and Business Information Control Study: a comparison, IBM Systems Journal 21, 1 (1982).