Comput. & Ops. Res. Vol. 13, No. 4, pp. 427–436, 1986
Printed in Great Britain. All rights reserved

0305-0548/86 $3.00 + 0.00
Copyright © 1986 Pergamon Journals Ltd

A DECISION SUPPORT SYSTEM FOR DATABASE EVOLUTION USING DATA MODEL INDEPENDENT ARCHITECTURE

CHENG HSU*

Department of Statistical, Management and Information Sciences, School of Management, Rensselaer Polytechnic Institute, Troy, NY 12181, U.S.A.

Scope and Purpose—This paper presents methods for the control of large-scale databases—the repository of information pertaining to an enterprise using computers. In particular, the paper proposes a set of quantitative models for monitoring the performance of database systems when the changing nature of applications of the systems is explicitly considered. Using these models, the system's performance will be assessed constantly according to the criteria set forth for them. Also, decision rules will be derived to determine the optimal policies of control. The methods that are proposed in the paper are especially useful for operating settings where both the interrelationships of data and the uses envisioned for them are evolving rapidly. Some database constructs unavailable previously are developed to facilitate the implementation of such a decision framework for database systems.

Abstract—The existing corpus of scientific knowledge in the field of database systems (DBS) has been concerned mostly with such problems as database technologies and system development methodologies. Relatively few efforts have been devoted to the problem of adapting an ongoing DBS in a systematic fashion. Notwithstanding the lack of sufficient prior knowledge, this adaptation problem is critical in DBS management, since a DBS should really be conceived as an evolving structure, rather than a stable one-shot phenomenon. Toward this end, this paper proposes a reliability-based decision support framework for evolving a DBS systematically. Both user satisfaction with the DBS and the usage pattern of it are monitored on a real-time basis and used for controlling it adaptively. Chance-constrained models are proposed to characterize suitable decision rules for DBS evolution decisions ranging from file reorganization to DBS restructing. A four-schema architecture is also proposed for achieving data model independence in a DBS, whereby facilitating the control and evolution of DBS.

1. INTRODUCTION

Managing a database system (DBS) has been and will continue to be a critical subject for research in the information system (IS) field. Although the planning, analysis and design of a DBS can determine its effectiveness in operation, the backbone of DBS management is really the continual control of the system *after* its implementation—i.e. a DBS has to evolve with the host enterprise embedding it. Seldom a DBS can be constructed once and for all without nontrivial alterations. Reasons for this are, basically, due to uncertainty in environments facing the enterprise. Underscoring the uncertainty, include, for instance, the imperfect nature of information available when the system was designed and the unpredictable evolution of the host enterprise.

We consider below such a situation whereby helping to bring the database evolution problem into prominence. Suppose an accounting information system has been in operation successfully for some time. The database of the IS was designed in close resemblance (in the sense of, e.g. Date [1] and Ullman [2]) to the hierarchical organization of the enterprise with the accounting manager being the principal (highest priority) user of it. Over time, new occurrences of the existing record types have been added to the database and old ones deleted; new record types have been included while some old ones dropped due to changes in job design and organization; the principal usage pattern of the database has been shifted from record-at-a-time type of payroll processing in batch mode to set-at-a-time cost analysis and on-line *ad hoc* queries; new uses and users in other functional areas (e.g. marketing and production) and departments have been added to the service list of the DBS and so on. In short, the system's applications have evolved into such a situation where its original design is no longer

*Cheng Hsu is an Assistant Professor of Statistical, Management and Information Sciences at Rensselaer Polytechnic Institute, Troy, New York. He received his M.S. in industrial and systems engineering and his Ph.D. in management sciences from the Ohio State University. His articles have appeared in *Management Science*, *Production Cost and Engineering Economics* and several other publications. He is currently engaged in research on the theory and methodology for adaptive database systems control and manufacturing information systems.

amenable to the information characteristics of the host enterprise, resulting in unsatisfactory system performance.

This type of dynamic evolutionary change in the basic characteristics of a DBS simply is pervasive everyday, everywhere from universities and government agencies to industrial and business firms. Consequently, a critical need in the management of DBS is the appropriate control that can effect a DBS most efficiently with respect to the changes detected. Notwithstanding its importance, this control problem has received far less attention in the literature than the analysis and design of DBS. As a matter of fact, the latter has become so structured nowadays, thanks to the vast efforts devoted to it in the past few decades (see, e.g. [3–8]), that the real challenge facing an enterprise utilizing DBS is no longer constructing the systems properly, but controlling them appropriately.

This paper presents a theoretical framework* for database evolution through adaptive control and general systems theory which have not been previously applied to interpret, formulate and solve the DBS control problem. In particular, a decision support system (DSS) characterizing this framework is proposed; one that is capable of determining (i) *when* a DBS should be evolved, (ii) *what* components or aspects of it should be evolved and (iii) *how* to implement the evolution decisions; in the light of effecting the DBS performance both timely and efficiently. The control methodology developed in Hsu [10] and De and Hsu [11], that features system reliability and real-time users' feedback, is employed as the basis for the "when" and "what" tasks above. A new four-schema DBS architecture is also designed to achieve data model independence to facilitate the appropriate control of the DBS in question—an undertaking critical for the "how" task to be comprehensively accomplished. Suitable methods that are available in pertinent literatures will also be adopted and integrated in the DSS when implemented.

Section 2 delineates the differing levels of DBS control, corresponding to differing levels of evolutionary causes, *vis-à-vis* previously existing literature in this area. The DSS proposed is discussed in Section 3, while the new four-schema architecture is presented in Section 4 in the context of appropriate control for DBS evolution. Section 5 makes some concluding remarks.

2. A REVIEW OF DATABASE CONTROL

In general, changing conditions in a DBS operation will necessitate control measures taken by the database administer (DBA), or the information system manager for that matter, in any combinations of the following categories of actions.

Reorganize the (physical) database

Although the computing activities required for databases are automatically performed by the attendant data management system (DBMS), new records will generally result in overflow areas which have to be consolidated with the regular areas once in a while to maintain the design efficiency of the physical file structures employed. Since the decision here is really a trade-off between the cost of efficiency loss due to overflows and the reorganization cost, it is in essence similar to the well-known inventory replenishment problem. Many researchers, such as Yao [12], Batory [13] and Wiederhold [8], have developed decision rules and models for the problem under various conditions. A general limitation found in these works is their treatments of the random nature of record usage (e.g. assuming each record has either a fixed probability or an equal chance to be used); these treatments tend to be overly simplistic in dynamic settings of the real world. When the assumptions are somehow relaxed, the results will no longer hold [3].

Redesign the (physical) database

Data might be stored in such a way that the data most frequently used are also the easiest to be accessed (e.g. requiring the least "seek" movements of the hardware). Access keys of data or even file organizations might also be selected to optimize the computing efficiency for the intended usage or users. Such design methodologies are a common practice nowadays [7,14]; the problem, however, is to redesign the database according to changing usage patterns and preferred users. This adaptive

*Part of the framework has been implemented satisfactorily in a network DBS environment [9]. However, our discussion in this paper is focused only on theoretical aspects of the framework.

control is largely ignored in the literature, with the exception of a few, such as Hammer [5]. A limitation of Hammer's work, though, is that the proposal does not consider that redesign cannot be handled without also accounting for the need for schema alterations [7]. No specific decision rules are reported for differing usage patterns either. Nevertheless, the bottom line is that the usage statistics may well assist in schema analysis and design, thereby resulting in further evolution of the database system.

Fine-tune the database schemata

The three levels of database abstraction—storage schema, schema and subschema—reflect the status quo of the system and determine the way in which the database can be used. When they have to be modified to reflect changes in the reality, the alteration generally can be performed routinely through the DBMS, if the fundamental logical approach employed for developing the schema remains unchanged. Notwithstanding the DBMS capability, these alterations may well lead to a point at which the original logical approach is no longer optimal or even appropriate. Once again, the problem has to be addressed in a continual spectrum of causes and actions—see Sockut [15] for a framework characterizing this spectrum.

Restructure the database schemata

This corresponds to the situation where a new logical approach is necessitated by the changes in the reality. Usually, to employ a different approach for any of the three level views would result in the restructure of not only all three, but also the physical database and application programs. Researchers, such as Shneiderman and Thomas [16], Navathe [17] and Shu *et al.* [18], just to name a few, have paved the way for automatic conversion of schemata, files and programs from the source model to the target model under fairly general conditions. Although important, these works generally do not consider the possibility that data model independent architectures [19,20] may be implemented for the DBMS to simplify the need for restructuring. Changing the external model, for example, may not necessitate changing every construct in the system. The works of Date [21], Housel [6], etc., on providing the user with a unified interface facility capable of being coupled with all of the three basic data models (i.e. relational, network and hierarchical), together with the works of Chen [22], Hammer and McLead [23], etc., on permitting derivations of the three models from a single approach, have set the stage for simplifying the architectures needed to achieve data model independence. Parallel to this development are theoretical accomplishments concerning unifying the three models; exemplified by Jacobs on database logic [24] and Lien on model equivalence [25]. These works have made the operational and computing issues of data model independence even more promising. Evidently new designs exploiting these possibilities are needed to response to differing problems more selectively. But, just to reiterate the theme of this paper for quick reference, a more fundamental insufficiency of knowledge for the schema restructuring problem lies in the respect of making the decision. That is, given the availability of restructuring technologies, how one can determine the appropriateness of a schema, or a logical approach for that matter, with respect to others for an on-going organization so as to soundly decide when, what and how to restructure the schemata in question.

Effect user services

The DBA will also act on making services effective through appropriate documentation, user work shops and staff training, etc. This practice, *per se*, is fairly usual or even routine in most DBS management. The problem, again, is how to get specific and reliable signals, thereby allowing swift and precise response from the manager's end. A case in point is a problem often found in measuring user information satisfaction via conventional survey [26]; namely, users when interviewed tend to "aggregate" their experience with the system over both the scope and history of it, and come up with a mixed rather than specific assessment. Unless the DBA has sufficient personal contact with most users, which is often impossible, difficulties abound in narrowing down the signal "poor documentation" to any specific portions or aspects of the whole document system.

In summation, a DBS needs to be evolved as the host enterprise evolves, by appropriately *selecting* from a whole spectrum of control measures. This selection, in turn, calls for characterization and monitoring of DBS performance, regression of the performance to controllable causes and

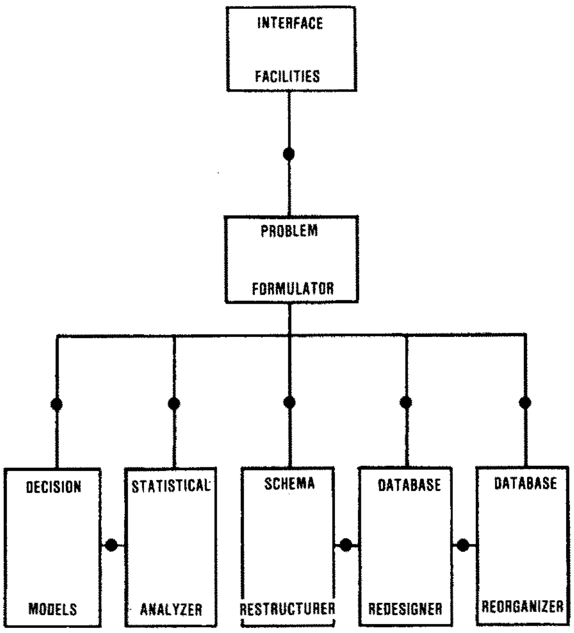

Fig. 1. A DSS architecture. ● Indicates inter-algorithm mapping/conversion.

optimization of control policies among all suitable candidates. These capabilities, along with those for the execution of differing control decisions, are furnished by the DSS for DBS evolution discussed next.

3. THE DECISION SUPPORT SYSTEM

We first propose a general framework for the DSS, then suggest methods to implement it. The general architecture of the DSS, shown in Fig. 1, features modules for (i) statistical analyses of system performance signals that would characterize the DBS performance and relate the performance to controllable causes, (ii) execution tools needed to implement the decisions and (iii) a problem formulator that integrates these modules and interacts with the decision makers through interface facilities. As such, the decision makers—IS managers or DBAs—may request information and analyses regarding DBS evolution from the DSS either on an *ad hoc* basis or via certain performance alerters incorporated in the statistical analyzer. Control decisions, which could be either furnished by the decision maker or suggested by action triggers [27] in the decision module, will be executed by the schema restructurer, database redesigner and database reorganizer as needed. The statistical analyzer, consisting of such tools as pattern recognition, quantile estimation and descriptive statistics, will result not only in depictions of the status quo, but also in assessments of the status quo regarding its acceptability with respect to system performance requirements. Such assessments give rise to performance alerters naturally. Decision models, similarly, will lead to (optional) decision rules capable of triggering DBS control actions on the basis of the performance signals detected (and, probably, analyzed).

An assumption underlying the DSS architecture is the existence of suitable performance signals and performance requirements. This assumption amounts to developing methods implementing this architecture; the methods follow next.

The fundamental performance measure of a DBS is, without a doubt, its productivity. Yet, practical questions abound: most of all, on how to define and measure the productivity. It is extremely difficult, if at all possible, to comprehensively quantity a DBS' states of operation in direct costs and benefits terms, not to mention assessing its indirect effects on the host enterprises. An alternative to this direct measurement is the use of surrogate measures such as user satisfaction in place of productivity (see, e.g. [28,29] for a discussion on this notion). Thus a real-time monitoring of this satisfaction through automatic surveys could provide feedback signals characterizing the DBS performance. These multifactor responses, in turn, could be analyzed to indicate the possible causes underlying the user dissatisfaction, if any. Actually, this is what we propose to do in the DSS for user feedback.

We further supplement this user satisfaction with signals generated automatically by the attendant DBMS from the actual use of a DBS; namely, various statistics regarding the usage patterns of both DBMS operators and physical database elements [9]. These statistics will provide clues revealing the computational nature of uses of the DBS. Altogether, these user-generated and system-generated feedback signals will serve as the foundations for relating the DBS behavior to suitable control variables. As a result, the basic decision tasks of "when to evolve" and "what to evolve" can be optimally determined accordingly.

The above notions are specified below; their developments can be found in Hsu [10].

User-generated signals

We first adopt an m-factor measure for user information satisfaction which can indicate the causes of the problem in terms of such performance attributes as response time and ease of use (see [10,28,29] for possible measures). The measure then is implemented in the interface facilities of the DBS to allow users to assess the system's performance on an individual jobs basis.

Further, the ith user response to a particular factor k for a job j, γ_{ijk}, is assumed to be subject to a white noise:

$$\varepsilon_{ijk} \sim N(0, \sigma_i^2). \tag{1}$$

That is, we take into account in this measure the undeterministic nature of human (quantitative) assessments, which are subject to various kinds of disturbances, including emotional, environmental and contextual sources. As such, we define below for the expanded real-time measure the user information satisfaction over a period of time containing n jobs of the user:

$$S_i = \frac{1}{n} \sum_{j=1}^{n} s_{ij}, \tag{2}$$

with

$$S_{ij} = \sum_{k=1}^{m} \gamma_{ijk} w_{ik}, \tag{3}$$

where

s_{ij} = computed overall satisfaction of user i concerning job j, and s_i is the user overall satisfaction during certain time period (e.g. since the system's last alteration),

γ_{ijk} = the user response (using the scales built in the measure, e.g. $-3, -2, \ldots, 3$) collected directly through the system's interface facilities,

w_{ik} = importance of factor k to user i (predetermined by the DBA and/or the users; or, this weighting factor can be suppressed from (3) for simplicity, per the DBA's discretion).

Users, therefore, will in essence be "surveyed" at the end of each session by the DBS—the "questionaire" will be prompted automatically to the user. Since questions pertaining to the m-factor measure can be organized into a hierarchical structure similar to the usual "help" commands, users could opt to respond at any degree of elaboration ranging from answering all m questions all the way to only giving an overall value, thereby enhancing the measures' feasibility in practice without compromising their basic validity. Kuo [9] describes such an on-line interface facility for users' real-time responses.

System-generated signals

User job requests are ultimately performed via the DBMS. Thus both the job's computing performance and its logical characteristics can be reflected in the DBMS's operation. A schema that is not natural to a job associating certain entities with others, for instance, may result in an excessively long series of query or data manipulation operators as well as computations (e.g. search and chaining).

Although the design of feedback must be system-specific, since it has to be tightly coupled with the DBMS employed (see Kuo [9] for a case study), certain categories of signals in principle would be needed for comprehensive control, in general, from the persective of computing efficiency:

(1) standard computing information for jobs, e.g. processing time, waiting time, DBMS time, queue size, etc.,
(2) standard data information, e.g. overflow, primary key and secondary key,
(3) frequency of data usage, overall as well as per job,
(4) frequency of DBMS operator usage, including the usual GET, WRITE, FIND and those for set-at-a-time processing,
(5) associations of data,
(6) user status, e.g. priority group, access path and security class.

These data about data and operations are themselves, once again, samples or sample statistics of the usage population for the entire enterprise during a period of time. Albeit system-specific, generic principles can be formulated for use in assessing particular logical views [30]. In a network environment, for example DBMS operators usually are designed after certain standard constructs (e.g. the DBTG model). They can also be grouped to mimic a classical sequential retrival or random access, among others. Thus decisions rules in terms of these standard operations will have general applicability transcending the particularity of individual DBMS.

System reliability decision rules

Both types of signals above will result in a time series that is informative for determining the suitable evolution measures. However, to determine when the evolution measures should be activated, we need a decision framework grounded in reliability theory due to the probabilistic nature of the signals. Succinctly, we propose that control decisions be characterized by chance constraints whose general format is

$$\Pr\{g(\gamma_{ijk|i\in I, j\in J, k\in K}) \geqslant b\} \geqslant \alpha, \tag{4}$$

where $g(\gamma_{ijk})$ is some satisfaction measure function regarding certain or all aspects or attributes of the system performance (e.g. ease of use for *ad hoc* queries) with respect to some user set I concerned, job set J performed and factor set K involved; b is the numerical satisfaction standard determined by the DBA; and α is the reliability requirement ($0 \leqslant \alpha \leqslant 1$). For example, when the entire user community and the whole system are concerned, the measure g will be the summation of S_i in equation (2) over every i. In this case, equation (4) says that the total satisfaction value must be at least equal to the standard, b, in 100α out of 100 cases in the intervening interval. When, on the other hand, only a particular user (e.g. the accounting manager) is concerned, the g is simply identical to (2) with the proper index i. Apparently any desirable aggregations over I, J and K are permissible for g.

Consequently, we are now able to derive a decision rule based on (4) in the spirit of Hsu [31], which in turn adopts the "zero-order decision rules" of chance-constrained programming [32] originated by Charnes and Cooper, as follows:

> "If the α-quantile of the satisfaction measure g is at least equal to the standard, then the system is effective and the status quo can persist; else adapt the system according to causes implied by both the user-generated and system-generated signals".

The quantiles (i.e. percentiles) needed in the above decision rule usually will be derived either from distribution laws or by using certain direct quantile estimation techniques available in the literature (e.g. Gorr and Hsu [33])*. In the latter case, both (4) and (5) can be implemented and monitored directly through the real-time signals, resulting in a fairly easy use of the system reliability rules.

*The chance-constrained models proposed in this section are concerned with chance contraints only. Other classes of models are also plausible (see [10,11]). To implement (5), the convolution distribution of g in (4) will be determined first, then its quantile function [e.g. the 0.975 quntile of normal distributions is: mean + 1.96 (standard deviation)]. This is the traditional approach. Alternatively, a linear model will be constructed for (4), and the quantile functions required will be estimated directly from the signals [33]. This approach is distribution-free and will usually result in mathematical models solvable by the linear programming methods.

Although the decision is primarily based on user perception of the system's performance, it is also logical to tune the system without user discontent, e.g. in a preventive maintenance manner. Prime cases in this regard may be database reorganization and file redesign. Since formal cost functions are commonly utilized by DBS for reorganization policies and even design alternatives under perceived usage patterns [12–14], the optimal policy and design need to be updated as the usage patterns evolve. The thresholds that determine whether or not the patterns, or the system usage in the host enterprise for that matter, have entered into a different regime should be, obviously, formulated and interpreted in the same system reliability context as discussed above.

In summary when the DSS characterized in Fig. 1 is implemented via the above approach, the DBS management will proceed basically in the following fashion. Feedback signals of both types will be generated and monitored constantly. When system performance requirements are (or are to be) violated according to pertinent chance constraints, the decision makers will be alerted by the system, and necessary analyses of the causes of problems will be undertaken. Decision models and decision rules will be involed, accordingly, to determine the optimal control policies at differing evolution levels (see Section 2). These decisions, finally, will be executed by the appropriate tools constructed in the system.

We now turn attention to the DBMS that is needed by such a DSS for, among other things, the generation of signals and the execution of DBS adaptations.

4. A FOUR-SCHEMA ARCHITECTURE

Up to now we have always implied that database adaptations can be achieved differentially, ranging from file reorganization all the way to schema restructuring; this will not be the case without proper DBS architectures. Consider, for example, a problem caused by users' changing views at the subschema level. When it comes to evolving the DBS, one would like to choose the action that is just comprehensive enough to remedy the causes—ideally, in this example, must to restructure (portions of) the subschema involved without necessitating other actions such as reorganizing the database. Unfortunately, the conventional DBS architectures (e.g. DBTG or ANSI/SPARC [20]) are all data model dependent: they can support only one data model. As such, restructuring any portions of the subschemata using a new data model will inevitably lead to overhauling the (entire) DBS.

Thus a data model independent architecture [19,20] is centrally important for fully materializing productivity improvements promised by the proposed DSS. If, for instance, the entire database has to be restructured so as to merely provide a more logical view of data to a particular user group, the whole idea may have to be foregone due to obvious cost deterrent. The improvement opportunity brought about by the feedback signals, in this case, cannot be seized.

We propose below such an architecture, based on a new four-schema design*. As shown in Fig. 2, a unified enterprise schema is employed to support differing logical approaches most suitable for particular users or user groups. Although general in itself, the enterprise schema may be best illustrated by drawing an analogy with the entity-relationship model (Chen [22]), which not only can be mapped into the usual data models, but can be directly implemented as well [34]. Likewise, the unified interface facilities of the architecture may be considered as similar to the unified database language (Date [21]). The statistics-gathering portions of the architecture are included to enhance the capacity of DBMS necessary for the proposed DSS. This part may be considered a statistical database system by itself, comprising the two types of signals discussed in Section 3. Finally, the intermodel mappings encompass both database technology and data model theory [18,24,25], but are sufficiently supported by the results in the literature.

In principle, any data model independent architectures can facilitate the DBS control. We note below two primary reasons for devising the four-schema approach at the presence of other data model independence designs. First, the use of an enterprise model has been shown in practice as invaluable for data modeling, regardless of the approach employed. To incorporate it into the DBMS capacity is

*The term "four schema" is used in the context where the conventional DBS architectures are referred to as "three schema". That is, one more level of abstraction is added to the usual subschema, schema and storage schema so as to explicitly account for the enterprise modeling. For the purpose of this paper, we will only qualitatively justify the design in terms of its conceptual soundness and theoretical feasibility in the light of DBS management.

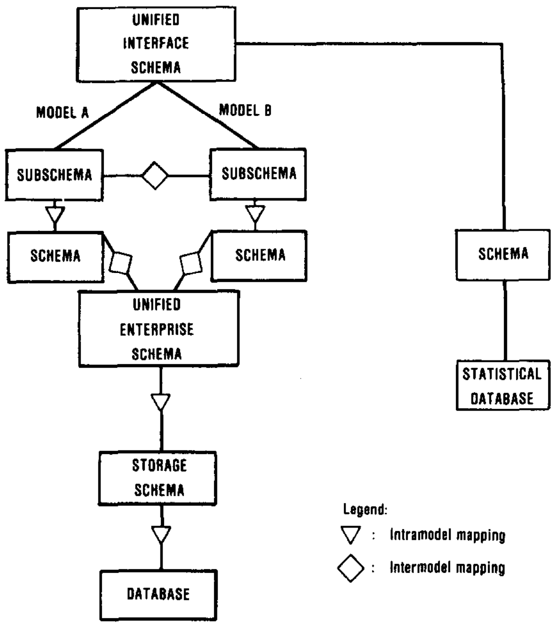

Fig. 2. Four schema model independence architecture.

both logical and beneficial. Second, a unified schema opens the possibility of optimally combining the best of all approaches natural to the users at the external level—e.g. ease of use and efficiency of processing. For instance, if the enterprise schema allows direct derivation from a storage schema similar to what a DBTG model would result in, and at the same time permits operations reasonably close to relational algebra and relational calculus, the amounts of intermodel mapping could be expected to be minimized in most of the applications.

This architecture can mitigate DBS changes against their affecting more components of the database than what are really implied. Specifically, appropriate responses are now available as summarized below.

(1) When only causes are file overflow and the like, reorganize the (physical) database.
(2) Usage patterns of the populated data have changed: restructure the storage schema, redesign the (physical) database and reorganize the files.
(3) Some of the users of a particular data model need a new view to their database: restructure the subschemata involved (e.g. modify the subschemata or even translate the (source) subschemata into new (target) ones if a new model is needed).
(4) A particular data model needs to be converted entirely: restructure both the schema and subschemata involved.
(5) Major organizational changes have taken place: restructure the unified schema and, accordingly, the entire database.

Note that now only the last category of changes will necessitate overhauling the entire DBS, while previously, with the conventional architectures, any of categories (3), (4) and (5) would result in this consequence.

5. CONCLUSIONS

In this paper we developed a DSS for DBS evolution based on the philosophy and methodology of both adaptive control and system reliability [31]. A four-schema architecture that promises data model independence was also proposed to synergistically collaborate with the DSS to achieve the full range of DBS management.

The DSS framework so developed represents an integration of pertinent but otherwise scattered techniques and a new application of them to the DBS control problem. More important, the formulation of this problem from the perspective of system reliability theory is unprecedented and promises new insights into the issues involved as demonstrated in this paper. As a matter of fact, an analogy may be drawn between what the decision framework adopted herein promises to contribute to the management of DBS and what the statistical quality control has been doing to the management of production systems since its inception decades ago. The significance of scientific systematic control of DBS becomes increasingly important when the database field evolves into its maturity. Works, such as the proposed framework, are indeed overdue.

Actual implementations of ideas pertaining to the DSS are currently being undertaken through both field tests and laboratory experiments. A major objective of the positive investigations is to develop robust decision rules of control for representive logical data models, as well as for particular DBS environments. Future research along this line will eventually enable the development of self-adaptive DBS's using the "intelligence" of such rules. The implementation and empirical study of the four-schema architecture is another subject for future research, whose significance and relevance are supported by, but not limited to, the adaptation of DBS. The architecture in its own right represents a new approach to achieving the increasingly imperative task of unifying data models at the level of IS practice.

REFERENCES

1. C. J. Date, *An Introduction to Database Systems*, 3rd edition. Addison-Wesley, Reading, Mass. (1981).
2. J. D. Ullman, *Principles of Database Systems*, 2nd edition. Computer Science Press (1982).
3. S. Christodoulakis, Implications of certain assumptions in database performance evaluation. *ACM Trans. Database Syst.* **9**, 163–186 (1984).
4. P. De and A. Sen, A new methodology for database requirement analysis. *MIS Q.* In press.
5. M. Hammer, Self-adaptive automatic database design. *1977 NCC Proc. AFIPS*, Vol. 46, pp. 123–129 (1977).
6. B. Housel, QUUEST: a high level query language for network, hierarchical, relational databases. In *Improving Database Usability and Responsiveness* (Edited by P. Scheuermann), pp. 95–119. Academic Press, New York (1982).
7. S. B. Navathe and J. P. Fry, Restructuring for large databases: three levels of abstraction. *ACM Trans. Database Syst.* **1**, 138–158 (1976).
8. G. Wiederhold, *Database Design*, 2nd edition. McGraw-Hill, New York (1983).
9. B. M. Kuo, Automating user feedback for evolving information systems. Unpublished Master's Project, Rensselaer Polytechnic Institute, Troy, N.Y. (1984).
10. C. Hsu, Evolving information systems through users' feedback: a database theoretic approach. Working paper, Series 36-84-P2. School of Management, Rensselaer Polytechnic Institute, Troy, N.Y. (1984).
11. P. De and C. Hsu, Information systems evolution through users' feedback. *Proc. 18th HICSS*, pp. 661–669 (1985).
12. S. B. Yao, A dynamic database reorganization algorithm. *ACM Trans. Database Syst.* **1**, 159–174 (1976).
13. D. S. Batory, Optimal file designs and reorganization points. *ACM Trans. Database Syst.* **7**, 60–81 (1982).
14. D. Ferrari, G. Serazzi and A. Zeigner, *Measurement and Tuning of Computer Systems*. Prentice-Hall, Englewood Cliffs, N.J. (1983).
15. G. H. Sockut, A framework for logical-level changes within database systems. *Comput. IEEE Comput. Soc.* **18**, No. 5, 9–27 (1985).
16. B. Schneiderman and G. Thomas, An architecture for automatic relational database system conversion. *ACM Trans. Database Syst.* **7**, 235–257 (1982).
17. S. B. Navathe, Schema analysis for database restructuring. *ACM Trans. Database Syst.* **5**, 157–184 (1980).
18. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh and V. T. Lum, EXPRESS: a data extraction processing, and restructuring system. *ACM Trans. Database Syst.* **2**, 134–176 (1977).
19. J. A. Larson, H. R. Johnson and T. B. Wilson, Data indepencence categorizations for the data base computer. *Proceedings of the 8th International Conference on System Science*, pp. 221–230 (1979).
20. D. C. Trichritzis and A. Klug, The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. *Inform. Syst.* **3**, 173–192 (1978).
21. C. J. Date, An introduction to the unified database language (UDL). *Proceedings of the 6th International Conference on Very Large Data Bases*, pp. 15–32 (1980).
22. P. P-S. Chen, The entitity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.* **1**, 9–36 (1976).
23. M. Hammer and D. McLeod, Database description with SDM: a semantic database model. *ACM Trans. Database Syst.* **6**, 351–386 (1981).
24. B. Jacobs, On database logic. *JACM* **29**, 310–332 (1982).
25. Y. E. Lien, On the equivalence of database models. *JACM* **29**, 333–362 (1982).
26. D. Deese, Experiences measure user satisfaction. *Proceedings of the Computer Measurement Group of ACM* (1979).
27. S. K. Chang (Ed.), Office information system design. *Management and Office Information Systems*. Plenum Press, New York (1984).
28. J. B. Bailey and S. W. Pearson, Development of a tool for measuring and analyzing computer user satisfaction. *Mgmt Sci.* **29**, 530–545 (1983).
29. B. Ives, M. Olson and J. Baroudi, The measurement of user information satisfaction. *CACM* **26**, 785–793 (1983).
30. W. Effelsberg and M. E. S. Loomis, Logical internal, and physical reference behavior in CODASYL database systems. *ACM Trans. Database Syst.* **9**, 187–213 (1984).
31. C. Hsu, Management of chance-constrained systems using time series analysis. Unpublished Ph.D. dissertation, Ohio State University, Columbus, Ohio (1983).

32. A. Charnes and W. W. Cooper, Deterministic equivalents for optimizing and satisficing under chance constraints. *Opns Res.* **11**, 18–39 (1983).
33. W. L. Gorr and C. Hsu, An adaptive filtering procedure for estimating regression quantiles. *Mgmt Sci.* **31**, No. 8 (1985).
34. C. Hsu, Structured database systems analysis and design through entitiy-relationship approach. *Proceedings of the 4th International Conference on Entitity-Relationship Approach.* IEEE Computer Society, pp. 56–63 (1985).
35. D. Hsiao, Data base computers. *Adv. Comput.* **19** (1981).
36. M. Schkolnick, A survey of physical database design methodology and techniques. *Proceedings of the Fourth International Conference on Very Large Data Bases*, pp. 474–487 (1978).