A KNOWLEDGE-BASED APPROACH TO THE ANALYSIS AND DESIGN OF BUSINESS TRANSACTION PROCESSING SYSTEMS

Matthias Jarke
and
Jacob Shalev

May 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 #53

GBA #83-52(CR)

A KNOWLEDGE-BASED APPROACH TO THE ANALYSIS AND DESIGN OF BUSINESS TRANSACTION PROCESSING SYSTEMS

Abstract

In this paper, we propose a new approach to the analysis and design of management information systems. While previous methods were either syntactic in nature (structured analysis and design, problem statement languages) or focused on the user-analyst interaction (user-controlled design, prototyping), our method - while compatible with both - additionally incorporates the experiential knowledge gained from the thousands of systems in operation today.

This goal is achieved through providing the systems analyst with a business systems architecture (BSA) consisting of two parts: a set of rules representing the knowledge about a generalized application domain, and a domain-specific database architecture that enforces these rules. The rules can also be used to check the design of existing systems and to guide the analysis process for new systems.

The paper describes a BSA for transaction processing systems. The approach can be applied in a similar way to other application domains such as decision support systems.

1.0 INTRODUCTION

Three main problem areas are commonly encountered in system development: (a) Time and cost overruns. (b) The resulting system does not satisfy the requirements. Even if the desired outputs are produced, the user interface is often not acceptable. (c) The system is difficult and costly to maintain.

Although from a management viewpoint these are serious problems in themselves, the system developer views them as symptoms of underlying problems in the systems development process. It is well-known that 40% of the development effort and up to 80% of the error handling effort are spent in the systems analysis and design phase of the life cycle [Alberts 1976, Boehm 1973, 1976, Brooks 1975]. Yet existing methods often fail to improve significantly the quality of these critical steps.

In this paper, we analyze some of the underlying reasons for this failure and propose a new method that adds the systematic use of semantic knowledge about a generalized application domain - in this paper: business transaction processing systems (BTPS) - to the systems analysis phase. This knowledge is derived with relative ease from the available experience with the development of the thousands of systems in operation today, yet none of the available methods exploits it systematically for the development of new systems.

We express the knowledge about BTPS in a business systems architecture (BSA), consisting of a set of rules and of a specialized database architecture based on these rules. The rules can be used to

evaluate an existing design as well as to guide the analysis process by "asking the right questions".

The BSA database architecture is described in a companion paper [Jarke and Shalev 1983]. An important part of that architecture, input management, has been implemented and is being used in the development and operation of defense BTPS in several countries.

A further informal test of our methodology was conducted with a group of advanced students who were asked to evaluate a design proposed in a textbook. While they failed to detect any major problems, the application of our rules revealed a number of grave omissions and errors in the design. Nevertheless, no claim can be made that the rules are complete. Therefore, a flexible structure is required that allows extensions of the knowledge base.

The paper is organized as follows. Section 2 gives a theoretical treatment of the inherent difficulties of systems analysis and design. Section 3 analyzes the shortcomings of current methods with respect to these difficulties and relates our approach to work done in other areas, mainly abstraction mechanisms developed in artificial intelligence, database, and programming language research.

Section 4 describes the derivation process of domain-specific knowledge about BTPS to be used in our method. As an example, two requirements, monitoring and systems visibility/ user control, are refined to more detailed rules. Finally, section 5 summarizes the application of knowledge in our architecture and the expected impact on the systems analysis and design process.

2.0 SYSTEMS ANALYSIS AND DESIGN -- A TRANSFORMATION PROBLEM

When trying to develop an information processing system (IPS), two domains are given: the enterprise (the "real world") and the computer environment. The IPS developer introduces new domains by developing models of the IPS to be constructed.

The life-cycle approach to systems analysis and design is aimed at breaking down the lengthy and expensive system development process into manageable phases. Each phase has specified objectives and results in the production of necessary system documents and products. The output of each phase becomes the input to the following phase [DeMarco 1978].

The IPS construction can be seen as successively solving a series of transformation problems that optimize (or satisfice) an objective function subject to constraints. At each phase, the solution of the problem generates constraints for the following one:

(A) The Information Requirements Analysis Problem:

Objective: Attain the enterprise's goals (profit, market share)

Constraints: Resources, environment imposed, etc.

Domain of decision variables: Technology, people, tasks, organizational structure, IPS requirements.

Note that this formulation views the IPS as a component in the overall structuring of an enterprise and is thus compatible with the Organizational Behavior approach of viewing the introduction of an IPS as a process of organizational change.

(B) The System Analysis Problem:

Objectives: Produce an understandable, buildable and maintainable system specification.

Constraints: The information requirements and eventually the rest of the enterprise domain variables.

Domain of decision variables: Data flows, data structures, processes and their data transformation specifications.

(C) The System Design Problem:

Objectives: Produce a maintainable, error free design, that maximizes cohesion and minimizes coupling.

Constraints: Data flows, data structures, transformation specs, physical computer environment.

Domain of decision variables: System structure, program modular structure, module specifications, file structure, data structures.

(D) The System Construction Problem (programming):

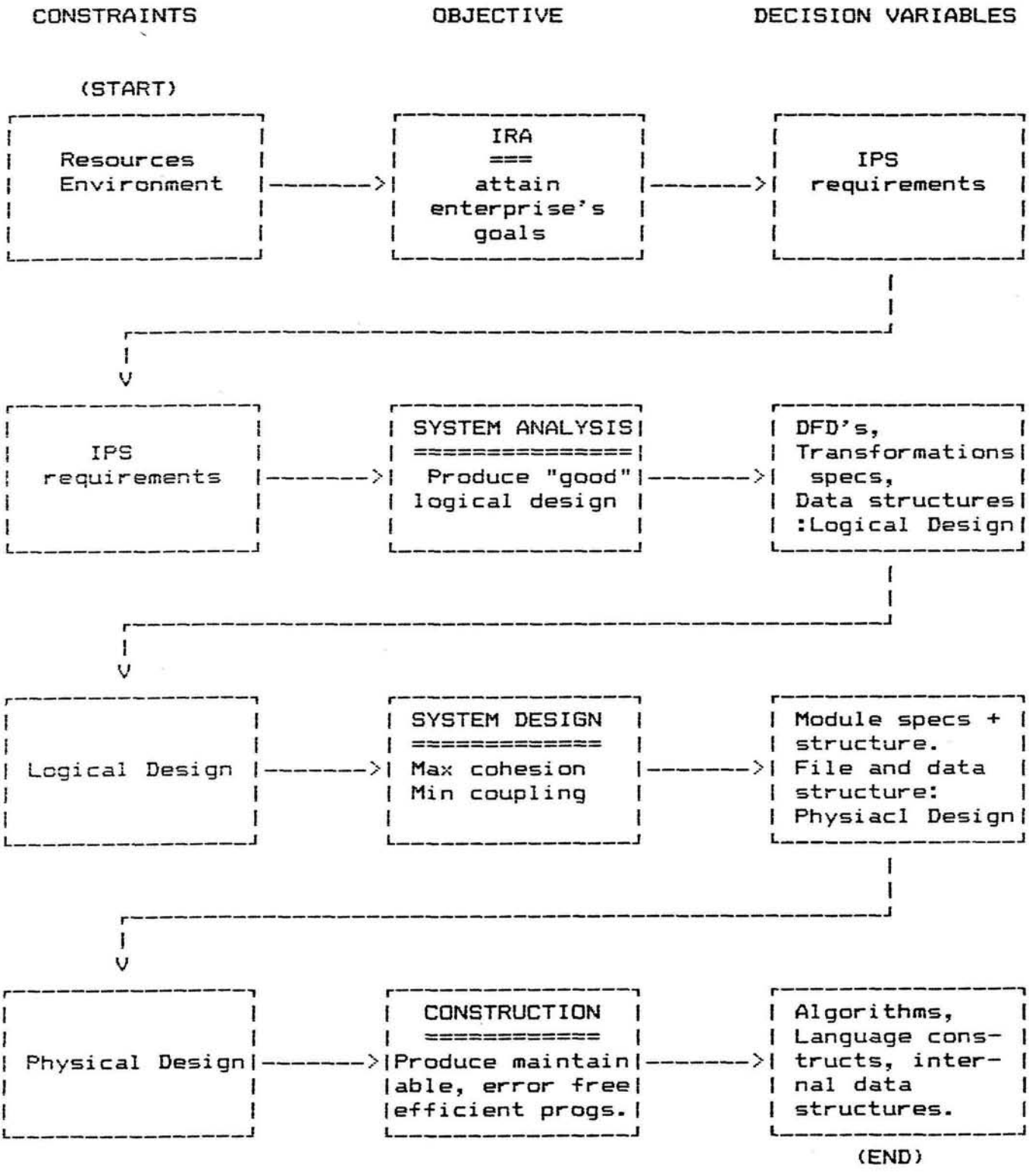
Objectives: Produce a maintainable, error free and efficient system.

Constraints: System structure, program modular structure, module specifications, file structure, data structures, available programming tools and the computer environment.

Domain of decision variables: program constructs (if.. then.. else.., sequence, iteration, etc.), internal data structures, algorithms.

Figure 2-1 summarizes the above description. The IPS developer is faced with a large combined gap - from the enterprise's real variables to the end result of programming language constructs and program data structures. He tries to cross the gap by iteratively solving the above problems. An error in an earlier phase may have profound implications for the later phases.

Figure 2-1: The Transformation Problems



The development time of a large IPS is long. User requirements change over time, at a rate that may be faster than that of development. These changes do not only have to go through the same transformations but may seriously impact the current solution at each phase. Thus the inherent difficulty of the development process becomes almost obvious. We need ways to speed up the process and increase its flexibility and "tolerance" for change.

3.0 REVIEW OF SYSTEMS ANALYSIS APPROACHES

In this section, we review the solutions of three groups of methods concerned with systems development (life cycle methods, prototyping, abstraction mechanisms), and contrast them with our approach.

3.1 Life Cycle Methods

The MIS literature provides us with life cycle oriented methods. We will focus on the structured analysis methods [DeMarco 1978, Ross 1977, Orr 1977;1981, Yourdon and Constantine 1978, Gane and Sarson 1979, Warnier 1981].

Some methods designed for information requirements analysis extend into system analysis. BSP [IBM 1981], BIAIT [Carlson 1979, Burnstine 1979;1980] and BICS [Kerner 1979, Zachman 1982] seek to perform an enterprise level analysis of the information needs, and to facilitate a smooth transfer to the detailed levels of subsystem analysis and data base design.

Another group of methods are the problem specification languages (PSL). The early PSLs of the 50's and 60's [Young and Kent 1958, Grindley 1956] were motivated by the need to accurately capture user requirements, linking all the components in a comprehensive way.

PSLs served as the foundation upon which a very ambitious approach was attempted - ISDOS. This method tried to automate the overall system development process from a problem specification in PSL through specification analysis (PSA) and physical design (SODA) to code generation [Teichroew 1970, Teichroew and Sayani 1971, Nunamaker 1971, Nunamaker and Konsynski 1976].

The expectations that ISDOS will become the system development method did not materialize [Couger et al. 1982]. However, a less ambitious method, PLEXSYS, emerged from the ISDOS approach that supports the development effort by providing a workbench environment for system development [Nunamaker and Konsynski 1982].

Like any other system development methodology, the life cycle approach cannot avoid the basic transformation problem: from the enterprise's objectives to a set of working machine language programs. However, additionally the life cycle phases have been criticized for being too difficult to be actually carried out with acceptable quality, and for taking too much time [Freeman 1980, McCracken 1980, Martin 1982].

3.2 Prototyping

The failure of the life cycle methods to significantly reduce the development time and assure acceptable system quality, prompted a call to abandon them in favor of prototyping. Rather than developing a large IPS in phases, prototyping calls for implementing a sequence of systems, each time refining the current "coarse" version in accordance with user feedback. The approach shortens the time needed to produce initial versions by relying on high level languages (HLLs) and data base management systems [McCracken 1980, Zmud 1980, Martin 1982].

If the performance of the last prototype is poor, or if the application will be run frequently, the heavy time consumers will be rewritten in a more efficient language, or, if that is not possible, the system serves as the specification for the target application to be developed using conventional languages and DBMS.

Our perception is that this approach makes a number of hidden assumptions. We summarize them in the following list to explain our belief that prototyping (at least alone) may not be suitable for building large backbone BTPS.

1. The application is built upwards and never has to restart. This assumption is very optimistic. Design decisions made when only a small portion of the system requirements are known may have to be severely modified requiring extensive reprogramming.

2. Users use the prototype long enough to provide good feedback. This assumption may not be realistic for a BTPS environment. Data may be output of another subsystem not yet built, the user may be busy doing his regular work. The effort of producing the necessary variety of situations so as to make user sessions realistic may approach that of detailed design, but now we also have to program and very probably modify. The effort involved may inadvertently limit the scope of the analysis.

3. The difference in the end may only be that of efficiency. This assumption hides the fact that current HLLs use different environments from common BTPS. An incompatible environment is a serious disadvantage: 40% of package buyers indicated that environment compatibility has a major influence on choosing a package (Datamation software rating, March 1983).

4. The application can serve as a specification for conventional program development. However remember that we do not have a documented statement of the requirements, no DFDs and no data dictionary. Program conversion experience shows that a working program is a very poor specification tool.

5. Prototyping presents a process of improved user requirements elicitation. The above study found that an overwhelming 87% out of 2387 users based their decision to buy a package heavily on the features and functions provided. It is not clear whether prototyping produces more features and functions than a systematic analysis.

3.3 The Knowledge-Based Approach

We will not concentrate on the sometimes unsatisfactory results of the above methods but rather on the inadequate inputs to the process. Tens of thousands BTPS have been implemented, and still, when we examine the phases of the methods we make the following observations:

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.). In standards here we mean standards that apply 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. The current methods do not ensure reproducible designs. Rather we find "ad-hoc" designs that are internally inconsistent. Designs are even less consistent across (sub)systems leading to extensive debugging and modifications.

Even the structured methods 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. Application generators can be seen as an ad-hoc answer to these issues but lack an underlying theory.

The central idea of our approach is that such a knowledge base cannot be developed for IPS in general. It is necessary to focus on a specific 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, the reader should consider for a moment 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) are meaningless for editors in general.

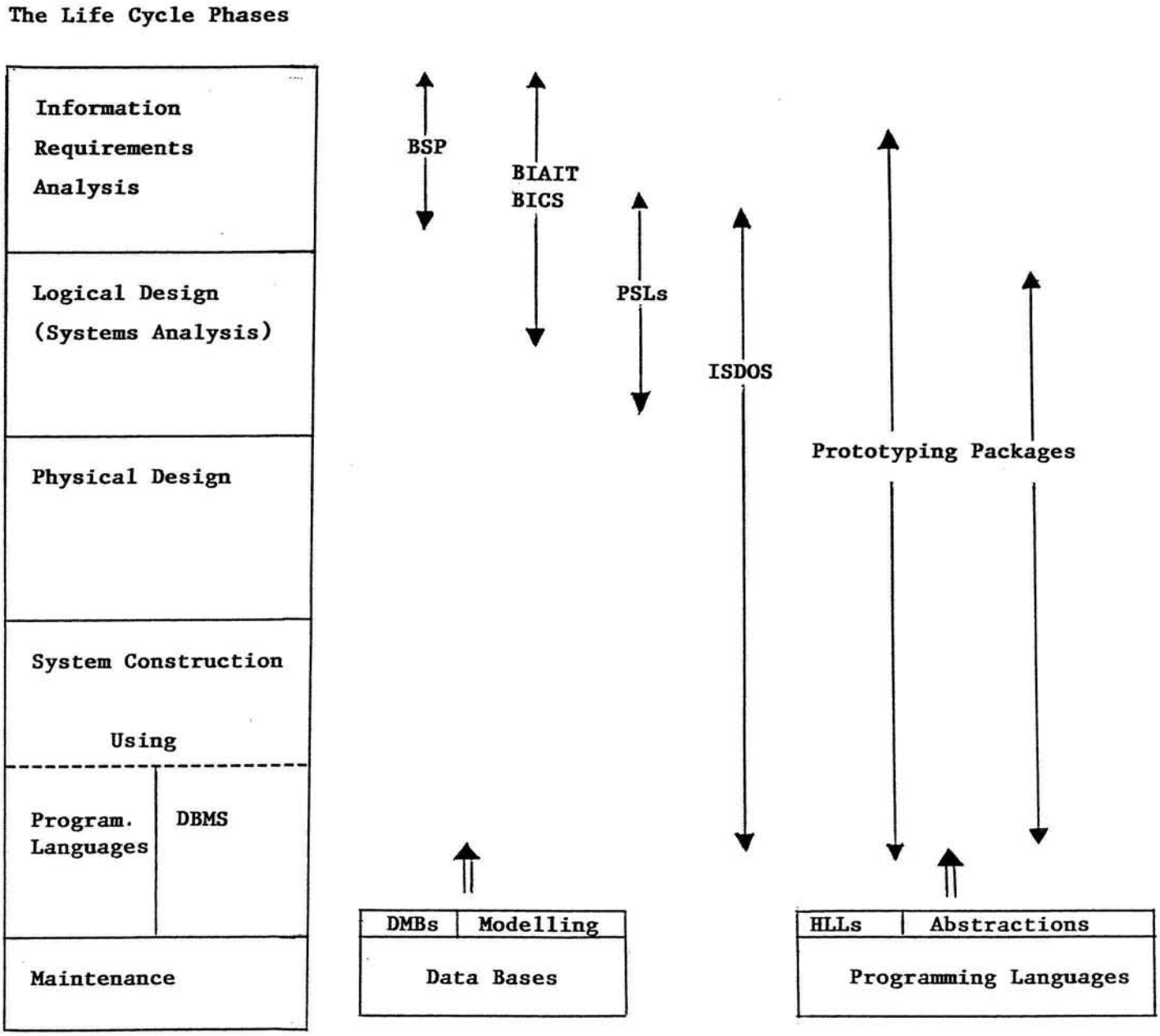
3.4 Relationship To Other Disciplines

Fig. 3-1 summarizes the existing methods and shows which life cycle phases are covered by each methodology. The bottom of the figure introduces two other disciplines, data base management systems (DBMS) and programming languages (PL). These disciplines are mainly concerned with tools for the system construction phase and not for analysis. However, the development of high level programming languages (HLL) and of powerful data manipulation languages (DML) actually changes the physical environment that the analyst sees.

The main advances in the PL discipline that will impact system analysis are abstraction mechanisms [Shaw and Wulf 1977, Schmidt and Mall 1983]. Abstractions support a methodology in which programs are developed by means of problem decomposition. At each stage of the program development, lower level abstractions are used to implement the current abstraction, thus isolating use from implementation. Abstractions seem to be a vehicle that can move programming languages from a mere construction tool to the earlier phase of analysis.

A similar path has been taken by the database researchers who have developed conceptual modelling that encompasses both data modelling and behavior modelling [Brodie and Zilles 1980, Schmidt and Mall 1983]. In the BTPS context, the concept of database transactions as a behavior modelling construct is of particular importance [Gray 1981, Rolland and Richard 1982]. A recent attempt to integrate these concepts into a complete system for IPS specification and implementation through compilation into a database programming language [Schmidt et al. 1982] is TAXIS [Mylopoulos et al. 1978].

FIGURE 3-1: System Development Methods



4.0 KNOWLEDGE STRUCTURE FOR BTPS

In this section, a domain-specific knowledge structure for BTPS is developed. First, the fundamental concept of a business program governing the business transactions is introduced. Within this framework, the specific constraints and requirements of transaction-level processing are derived. The requirements are further refined into specific rules to be considered in order to arrive at a satisfactory BTPS design. For space reasons, only two requirement areas can be discussed in detail.

4.1 The Concept Of Business Transactions.

At the operational level, a business is set up to carry out certain business transactions. The special property of this level is that there is 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. Essentially, the business program defines a script together with the associated planned processes. A planned process can be further refined into a collection of planned activities. Activities of the same type may occur in multiple process definitions.

Each instance of a business program is called a business transaction. Corresponding to the components of a business program, a business transaction is composed of (actual) processes which can be further described by (actual) activities. In contrast to database

transactions, business transactions are long-lived (take weeks rather than seconds) and nested (contain major subtransactions) [Gray 1981].

As an example, consider an Accounts Payable department that receives invoices, approves and pays them. In the invoice approval activity, 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 activity 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.

4.2 Business Constraints And Requirements

A business program must be designed as to achieve the business goal (performing the transaction) while complying with certain constraints and practices. We have developed a hierarchy of business constraints using the process sketched in Figure 4-1. We identify these constraints and practices and show their implications for the design of detailed rules in two areas.

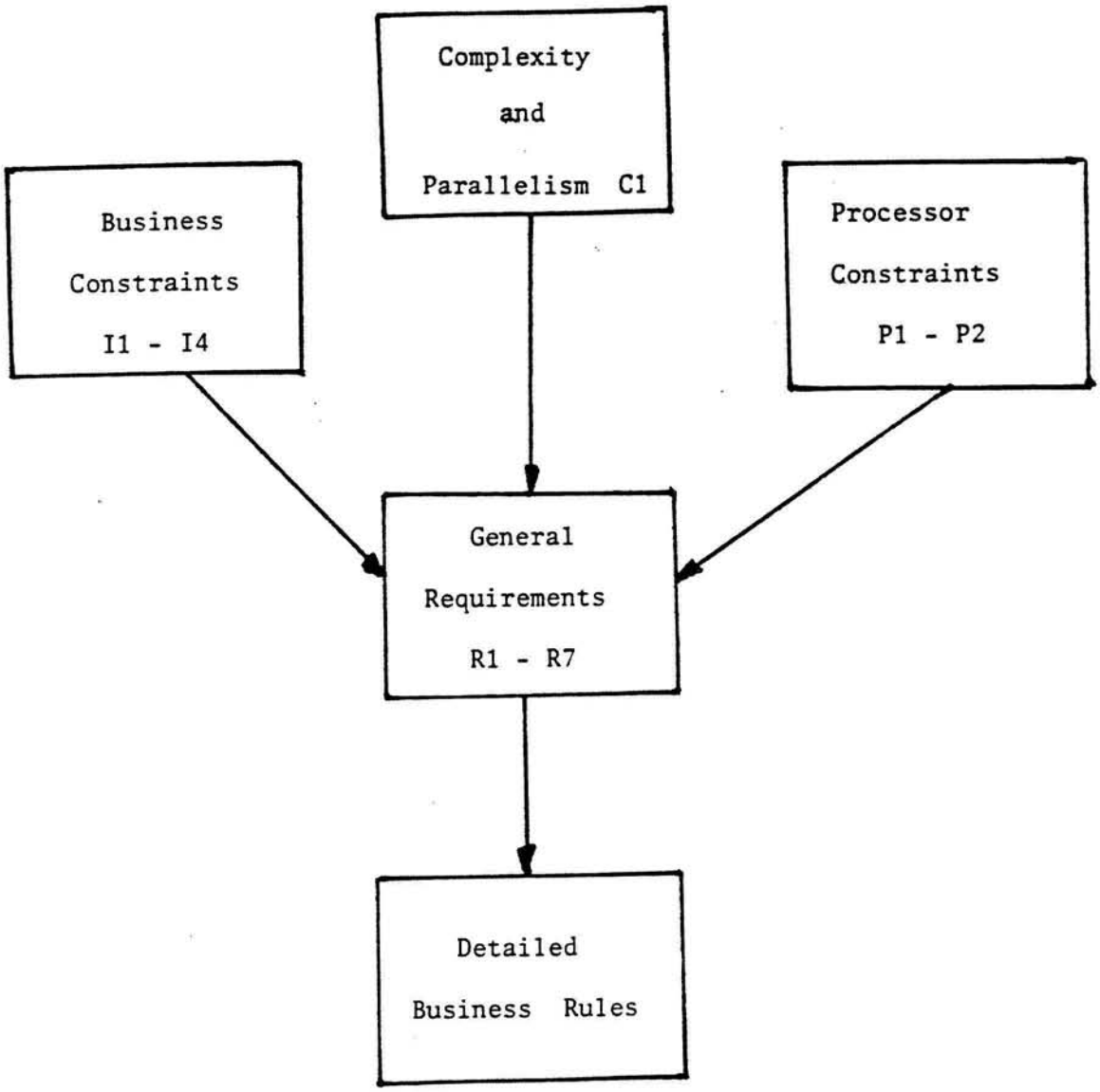


Figure 4-1 : The Derivation Process of Business Requirements and Rules

I1. Quality of the program's outputs. The business program has to make sure that all necessary outputs are produced, and are correct and precise. This is an issue of major importance not only because customer relations are at stake, but because serious errors can threaten the business' existence. Hence the commonly found business rules, signature requirements, authorizations and other business quality control devices.

I2. Timeliness. A 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). The business may also elect to establish time constraints for processes and critical activities.

I3. Accountability. The business is held accountable for its activities by its clients, the law, and its shareholders. We can distinguish between short term (current transactions) and long term (reporting and audit) accountability.

I4. Responsiveness. We define responsiveness as the business' willingness to accommodate a changing environment. In the short run, the business will accept and act on requests for change in a current transaction. Long run changes will induce the business to change the business program itself.

The above business issues are not the only constraints imposed on the business program. Besides satisfying constraints originating from the environment, such as limits on the use of external resources, we must also note that two types of processors perform the activities of the business program: people (the human processor) and computers (the machine processor).

P1. The human processor. People make mistakes when processing documents. They err in performing decision rules and routing 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 the business program. They lack the integral (limited) quality control capacity of humans: common sense. Transformations of data into and out of the computer are required. This man-machine interface is 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 user control and data visibility.

The complexity and parallelism of the business program (constraint C1), coupled with the above constraints (I1-I4,P1-P2) identify requirements that the design of every BTPS must resolve. We briefly describe a set of general requirements that a good business program must satisfy and then discuss two of them in more detail.

R1. Monitoring is the ability to know for each transaction, in what process/activity it is and conversely, what is a certain activity doing. This must be compared against deadlines set by the business or outside constraints.

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.

R3. Queue management: As the activities are performed in parallel, it is probable that activity B will not be ready to immediately process the results (output) of activity A. This gives rise to input and output queues, and the need for their management as an integral part of the business program (e.g. can we recover a queue of input documents?).

R4. Error handling: Whereas in other system types an error may just prevent successful completion, in a business environment it may have additional adverse effects. The business processes must therefore be designed to actively detect and eliminate errors, with an emphasis on effective error presentation and correction.

R5. Quality control takes into account the inability of the business program processors (P1-P2) to detect all In high risk situations, quality control activities will check outputs, and may require compensating transactions and amendment capabilities to be added to the business program.

R6. 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.

R7. 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?

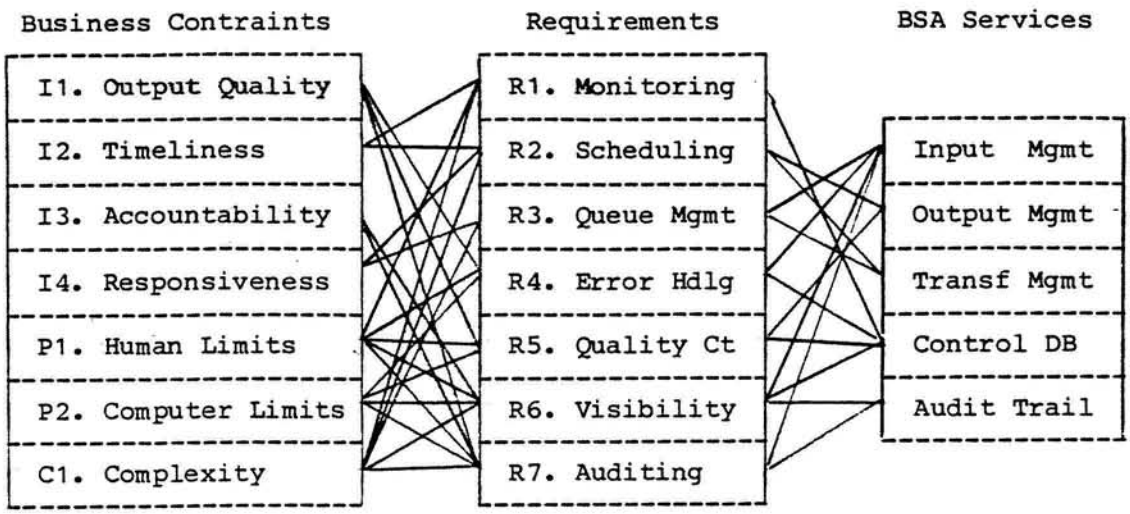


Figure 4-2: Derivation of Business Requirements and BSA Services

The left half of figure 4-2 describes which of the business constraints discussed previously give rise to each of these requirements. The right side says how, in turn, the requirements lead to the business architecture services to be described in section 5. For space reasons, we cannot discuss all these relationships in detail. However, two examples (R1 and R6) may illustrate the process. They also indicate how the general requirements can be broken down into more detailed rules to be used in the design process. Since the first use of such rules is typically design checking, the rules are mostly stated as questions any BTPS design should have an answer for.

4.3 Monitoring

Consider the issue of timeliness (I2). The business transaction has to be completed within a predetermined time period. We view that as an objective of the business program. The question that arises is: If we design the business program to just do the data transformations

of the transaction, will the transaction actually complete in time? Stated differently, do we need to add some components to the design of the business program so as to make sure that we attain the timeliness objective?

The complexity of the business program (C1) and the limitations of the human processor (P1) provide an obvious answer. The business program is a complex network of activities that are performed asynchronously and in parallel and it is therefore highly probable that a transaction will not complete on time in such an environment. This probability is even higher if we consider the slowness and the propensity to err of the network's human processors.

We therefore conclude that monitoring capabilities must be designed into the business program to ensure the attainment of the timeliness objective, or in other words: the relationship of I2, P1 and C1 yields the general requirement of monitoring (R1). Specific monitoring rules are aimed at satisfying the timeliness objective in the presence of complexity and processor imperfections. The following monitoring questions should be answerable in any well-designed BTPS:

1. In which process and activity is transaction X?
2. What is activity Y processing?
3. What transactions do not meet time limits imposed on activity X, or on process Y, or on the whole program?
4. What transaction or activity is in a special status (hold, urgent)?

4.4 System Visibility And User Control

Consider the issue of accountability (I3). The business has to be able to answer outside questions and explain its actions while the transactions are in progress or later. We also know from the need to be responsive (I4), that some questions may be connected with requests for change. We again view accountability (I3) as an objective and ask if it can be attained without specifically designing for it.

When we relate this objective to the attributes of the computer processor (P2), we note that one of the most elementary ways of providing accountability: reviewing all documents and paperwork related to the request, is possible only if designed for. The limited storage and retrieval abilities of humans (P1), and the multitude of activities and transaction instances (C1) add to the inherent difficulty of attaining the accountability objective. We thus derive the need to actively design the business program for accountability by providing tools for system visibility and the following user action which we term as user control.

To achieve system visibility and user control, a business program should be designed so as to enable answering the following user questions and requests:

1. What actions did I request (submit)?
2. Was an action successful? If successful:
    1. What were the results of the action?
    2. I want to amend the results.

3. I want to reproduce the output (without re-applying the action!).

If unsuccessful:

1. Where was I wrong?

2. I want to correct my request.

3. How was a certain result generated?

The last question may seem similar to the auditability requirement (R7). In a sense we may think of it as a short-term "local" auditability - tracing the recent actions to effect the result of interest. This rule supports the user's immediate control capability.

5.0 APPLYING THE KNOWLEDGE TO IMPROVE BTPS DEVELOPMENT

The BSA provides a three-faceted knowledge base: (a) the business program model, (b) the general rules and requirements, and (c) the design architecture. In this section, we outline the application of this knowledge to improved systems development.

5.1 A Database Architecture For BTPS

Some requirements take application-specific forms when actually used. The others provide a common denominator for all BTPS. A specialized database architecture described in [Jarke and Shalev 1983] provides an envelope of business program services to support those requirements, leaving the developer only with the design and implementation of an application-specific kernel.

The constructs of this architecture include general services that can be mapped to pre-fabricated data flow diagrams or parametrized software modules along with their supporting data structures (or data bases). We briefly summarize the components.

An input management service provides the capability to handle erroneous input documents without interrupting processing or requiring unnecessary data re-entry. For this purpose, all documents are stored in an input database.

Output of database transactions can be used in multiple different forms and may have to be reproduced later (without repeating all the processing). Output management provides the service of maintaining and presenting output data using an output database.

Between input and output, transformation management roughly covers the functions of conventional database transaction execution and supervision, with functions added for modifying the status of the input, output, and control 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 through access services, and a limited amount of interrupt facilities through control services. The audit database permits ex-post tracing of transactions.

Taken together, these standard components enforce the use of many of the business requirements and rules as indicated in figure 4-2. For example, one of the main purposes of the control database is to

satisfy the requirements presented in sections 4.3 and 4.4. The services can be described on the design level by standardized data flow diagrams, or they can be implemented as parameterized software packages. In both cases, they relieve the application designer from the task of reinventing standard operations.

5.2 Impact On The Structured Analysis And Design Process.

The knowledge-based approach can enhance both the life-cycle methods and the prototyping approaches. In this subsection, we briefly analyze the impact of the approach when added to the structured life cycle as described in sections 2 and 3.1. The knowledge structure outlined in section 4 can be used in several phases of the life cycle.

The general model of a business program aids in the earlier phases of detailed requirements analysis to structure the problem at hand. Additionally, also the rules and requirements can be used as issues for discussion and analysis within the project group, and as an aid in eliciting user requirements and priorities.

In later stages, the BSA provides pre-fabricated components of the overall logical design to which the analyst adds the application-specific kernel. Furthermore, requirements and rules can be used to evaluate a proposed systems design as demonstrated in the informal experiment mentioned in section 1.

Analysis and design are not atomic phases. They are broken down into more than ten interrelated activities [DeMarco 1978, ch. 2], and one can study the BSA's impact on each. A complete discussion is beyond the scope of this paper but will be a logical next step in our work.

6.0 CONCLUSION

The BSA was derived as a result of focusing on a subset of IPS, BTPS. It provides tools that improve the detailed requirements definition and logical design of a BTPS. It then proceeds to provide standard designs that address themselves to the common denominator of most BTPS, assuring acceptable quality and lower implementation costs.

Considering the future, it is conceivable that BSA-based software packages can provide "tailored environments" (for a certain machine, DBMS and on-line monitor) for BTPS implementation. The business will then be able to tailor an application to its specific needs while still enjoying many of the benefits of packages.

Another obvious extension of our approach is to incorporate part of the BTPS knowledge structure into an artificial intelligence-based expert system [Nau 1983] that supports design checking and is linked to information about the special-purpose database structure. The problem of linking expert systems to databases has been studied in [Jarke and Vassiliou 1983]. However, it should be noted that a major additional effort would be required to encode the knowledge captured in our rules in a useful computerized form.

Finally, the portability of our approach beyond the domain of BTPS will be investigated. For example, in [Jarke 1982] we show the usefulness of time concepts, input and output management mechanisms in an operational level decision support system. On the other hand, the purpose of these services is quite different from those in BTPS, and in general, many BTPS requirements (e.g., timeliness, accountability) do not play a major role in other system types while others (e.g., flexible user interfaces) may be more important in those than in BTPS.

References

1. D.S.Alberts: The Economics of Software Quality Assurance, Proceedings National Computer Conference, 1976.

2. B.W.Boehm: Software and Its Impact: A Quantitative Assessment, Datamation, May 1973.

3. B.W.Boehm: Software Engineering, IEEE Transactions on Computer, December 1976, 225-240.

4. M.L.Brodie, S.N.Zilles (Eds.): Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 1980.

5. F.F.Brooks: The Mythical Man-Month: Essays on Software Engineering, Reading,MA: Addison-Wesley, 1975.

6. D.Burnstine: The Theory Behind BIAIT, Business Information Analysis and Integration Technique, BIAIT International Inc. 1979.

7. W.Carlson: BIAIT - The New Horizon, Database, Spring 1979.

8. D.J.Couger: Evolution of Business System Analysis Techniques, Computing Surveys, September 1973.

9. D.J.Couger, R.W.Knapp: System Analysis Techniques, New York,NY: Wiley, 1974.

10. D.J.Couger, M.A.Colter, R.W.Knapp: Advanced System Development / Feasibility Techniques, New York, NY: Wiley, 1982.

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

12. P.Freeman: Why Johnny Can't Analyze, Conference System Analysis and Design: A Foundation for the 1980's.

13. C.Gane, T.Sarson: Structured Systems Analysis: Tools and Techniques", Englewood Cliffs,NJ: Prentice-Hall,1979.

14. J.Gray: The Transaction Concept: Virtues and Limitations, Proc. 7th VLDB Conf., Cannes 1981, 144-154.

15. C.B.Grindley: SYSTEMATICS - A Nonprogramming Language for Designing and Specifying Commercial Systems for Computers, Computer Journal, 1956, 124-128.

16. IBM Corp.: Business Systems Planning - Information Systems Guide, Application Manual,IBM, GE20-0527, July 1981.

17. M.Jarke: Developing Decision Support Systems: A Container Management Example, Int. Journal of Policy Analysis and Information Systems 6 (1982), 351-372.

18. M.Jarke, J.Shalev: A Database Architecture for Supporting Business Transactions, NYU Working Paper Series CRIS#51, GBA 83-27 (CR), March 1983, submitted for publication.

19. M.Jarke, Y.Vassiliou: Coupling Expert Systems with Database Management Systems, NYU Symposium on Artificial Intelligence Applications for Business, New York, May 1983.

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

21. J.Martin: Application Development Without Programmers, Englewood Cliffs, NJ: Prentice-Hall, 1982.

22. D.D.McCracken: A Maverick Approach to Systems Analysis and Design, Conference System Analysis and Design: A Foundation for the 1980's.

23. J.Mylopoulos, P.A.Bernstein, H.K.T.Wong: A Preliminary Specification of TAXIS: A Language for Designing Interactive Information Systems, Technical Report, CCA-78-02, January 1978, Computer Corporation of America.

24. D.Nau: Expert Computer Systems, Computer, February 1983, 63-85.

25. J.F.Nunamaker: A Methodology for the Design and Optimization of Information Processing Systems, Proceedings AFIPS Conference, Vol. 38, May 1971, 283-293.

26. J.F.Nunamaker, B.R.Konsynski: Computer-Aided Analysis and Design of Information Systems", CACM, Vol. 19,No. 12, December 1976.

27. J.F.Nunamaker, B.R.Konsynski: Plexsys: A Systems Development System, in Couger et al. (eds.): Advanced System Development / Feasibility Techniques, New York,NY: Wiley, 1982.

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

29. C.Rolland, C.Richard: Transaction Modelling, Proc. ACM-SIGMOD Conf., Orlando 1982, 265-275.

30. D.T.Ross: Structured Analysis: A Language for Communicating Ideas, IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.

31. J.W.Schmidt, M.Mall, J.Koch, M.Jarke, "Database Programming Languages", Proceedings Database User Interface Workshop, Philadelphia, October 1982.

32. J.W.Schmidt, M.Mall: Abstraction Mechanisms for Database Programming, Proc. ACM SIGPLAN Conference, San Francisco, June 1983.

33. M.Shaw, W.A.Wulf: Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators, CACM, Vol. 20, No. 8, August 1977, 553-564.

34. D.Teichroew: Problem Statement Languages in MIS, Proceedings, International Symposium of BIFOA, Cologne, July 1970, 253-270.

35. D.Teichroew, H.Sayani: Automation of System Building, Datamation, August 1971, 25-30.

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

37. R.Welke, K.Kumar: An "ERA"-Based Analysis Support System for BIAIT, ISRAM Working Paper WP-8010-2, McMaster Univ., Ontario, October 1980.

38. J.W.Young, H.K.Kent: Abstract Formulation of Data Processing Problems, Journal of Industrial Engineering, Nov. 1958, 471-479.

39. E.Yourdon, L.L.Constantine: Structured Design, New York, NY: Yourdon Press, 1978.

40. J.Zachman: Business Systems Planning and Business Information Control Study: A Comparison, IBM Systems Journal 21, 1 (1982).

41. R.W.Zmud: Management of Large Software Development Efforts, MIS Quarterly, June 1980.