DSS DESIGN--A SYSTEMIC VIEW OF DECISION SUPPORT

Gad Ariav
Michael Ginzberg

October 1984

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

Working Paper Series

CRIS #84

GBA #84-81(CR)

Table of Contents

Abstract

Current DSS research is rather fragmentary, and typically myopic--it centers either on the decision situation which DSS support, or on DSS tools or generators. In this paper we adopt a comprehensive view of DSS emphasizing their systemic nature. This entails identifying the links among the five aspects that classically characterize a system:

1. the environment, i.e., decision situations and access patterns;

2. the function (within this environment), i.e., types and levels of decision support;

3. the functional components that make it up, i.e., dialog, data, and model management;

4. the arrangement, i.e., the linkages among the components and the assignment of functions to modules; and

5. the resources consumed, i.e., hardware, software, human skills, and data.

The systemic view provides a concrete framework for the effective design of DSS, and serves as a basis for accumulating DSS research results.

1. Introduction

DSS studies typically concentrate on either (1) the nature of decision situations and the type of services provided by a DSS (e.g., [Gorry 71], [Little 70]), (2) DSS components and the tools and technologies which are needed to provide them (e.g, [Bonczek 82]), or (3) the processes of DSS design, implementation, and use (e.g., [Keen 80], [Moore 80]).

Effective integration of these diverse elements is missing, and this hampers efforts to develop a solid basis for DSS design. Research that focuses solely on components or resources can offer only limited help in the design of DSS. For instance, recent books on DSS design (e.g., [Bennett 83], [Sprague 82], and [Bonczek 81]) have proposed a basic set of DSS components, but rarely explicated the necessary contingent relationships between this structural aspect of the system and, for example, the type of services it is expected to provide.

The premise of the systemic view of DSS is that understanding these systems requires the simultaneous consideration of the five system aspects, i.e., environment, role, components, arrangement of components, and resources required to support the system [Churchman 68]. A meaningful DSS design must explicitly link all these aspects so that characteristics of the system's environment and role will be reflected in its components and their arrangement.

The name DSS implies that the objects we are discussing are indeed

systems, yet this perspective has been lost in much of the DSS literature. The purpose of this paper is to present a comprehensive view of DSS, using the systemic framework as an organizing concept. No new terminology is introduced. Rather, the paper attempts to integrate the disparate perspectives found in the DSS literature into a consistent and coherent body of knowledge. This integrated view indeed reveals new insights.

Section 2 defines the classical aspects of a system. Sections 3 through 7 apply these definitions to DSS, explicating the meaning of each aspect in the DSS context. Section 8 examines more closely the relationships among the systemic aspects of DSS, particularly the ways role and environment determine components, arrangement, and resources. Finally, section 9 considers the implications of a systemic view for DSS design, design research, and curricula.

2. The Aspects of a System

The fundamental premise of systems theory (or the systems approach) is that systems, regardless of their specific context, share a common set of aspects or elements [Churchman 68]. The systems view, however, is an abstract model, as systems exist only in the mind of the beholder. Well designed man-made systems are constructed to resemble that view, as it provides a methodical justification for a set of components and their arrangement.

Systems thought emphasizes the need to take an holistic view in

order to answer why an object is structured as it is, or how it should be structured. Thus, the system study starts from without, identifying the environment in which the object exists and the way it impacts that environment, i.e., its role. Only after these external aspects of the system have been studied does it make sense to consider the internal aspects--the components and their arrangement. To maintain simplicity and to facilitate comprehension, only first level decomposition of the system is typically outlined, and components are presented without internal operational details (i.e., as "black boxes"). Each could eventually be a subject for further systemic study, revealing its internal structure.

Following the study of these four system aspects, resources can be knowledgeably selected and allocated. Resources are differentiated from system components, and should be discussed last, in order to ensure that the functional composition of the system is not unduely biased by the perceived availability of resources. That is, design should proceed from an "ideal" system to a critical review of resource availability, and then to a feasible system [Ackoff 81].

Briefly, the system elements are:

* The environment is the set of entities and conditions outside of the system boundary which affect or are affected by the system. The entities in the environment may be affected by the system, but are not controlled by it; that is, the coupling between the system and the environment is looser than that among the system components.

* The role, function, or objective of a system represents its

intended impact on its environment. It specifies what services the system is supposed to deliver and what its goals are. It also provides the basis for evaluating the system, and thus should be specified in terms which are amenable to measurement.

* The components of the system are the identifiable elements within the system boundary. Typically, these components represent the functional building blocks of the system. Two common bases for component definition are specialization of labor--the ability to perform effectively a particular, necessary task--and specialization by environmental segment--the ability to interface with a particular aspect of the environment. Obviously, neither of these two bases can be considered until the environment and system role have been specified.

* Arrangement concerns the links among the system components and between them and the environmental elements. The fundamental concern in arranging components is the balance between coordination and autonomy [Emery 69]. Generally, it is preferable to have the minimum of interdependence among components which still allows the system as a whole to serve its function. The two key dimensions of arrangement are (1) the configuration or layout of the links among components, and (2) the nature of these links.

* System resources are the elements which are used or consumed in building and operating the system. Like the environment, resources exist outside of the system boundary. Resources are differentiated from the environment in that they are at least partially controllable--if not in total quantity available, at least in the mix which will be used. Resources may include people, raw materials, capital, tools and techniques, etc.

The following sections identify the five system aspects in the DSS context.

3. DSS Environment

The description of the environment should be design-relevant; it should highlight only those aspects that have, or should have, an impact on the system structure. There seem to be at least two important dimensions to the DSS environment: the task type and the pattern of access. Task type has been discussed in the DSS literature since the earliest days, but we suggest a somewhat broader range of task aspects. Specifically, we characterize tasks by their structurability, level, decision process phase, and functional area. Access pattern includes the interaction mode, user community, and relationship to other computer-based systems.

The task characteristic most frequently associated with DSS is the degree to which the decision maker can apply predefined rules and procedures, i.e., task structurability [Ginzberg 82]. Gorry and Morton [Gorry 71] used task structure as a key concept for defining the appropriate environment for DSS. More recent work has suggested that there is no inherent structure to the task itself, only structure as perceived by individuals [Moore 80]. We should think instead about task structurability, the possibility of bringing structure to bear on a task, which is dependent upon both the individual performing the task and the task itself.

A second key task characteristic is its level--operational control, management control, or strategic planning [Anthony 65]. This characteristic, too, has been discussed in much of the "classic" DSS

literature (e.g., [Gorry 71]), with the suggestion often made that DSS were only appropriate at higher levels. More recent DSS literature has suggested that DSS may be appropriate at any level (e.g., [Ginzberg 82], [Sprague 82], [Gorry 83]).

The third important task characteristic is the decision process phase--intelligence, design, or choice [Simon 60]. Most DSS have been used at the choice (i.e., solution selection) or, to a lesser extent, intelligence (i.e., problem definition) stages of the decision making process. They are also applicable to the design (i.e., alternative generation) stage, and the pattern of use to date reflects the difficulty of designing systems to support the "design."

The final task characteristic to be considered is its functional area, e.g., finance, marketing, production. The demands and constraints which different functional areas place on a DSS may be as significant as those resulting from any of the other task dimensions.

The second critical dimension of the DSS environment is the access pattern. Like task type, this too has several aspects, each of which limit or constrain the DSS. First is the pattern of user interaction. The early DSS literature assumed that use would always be highly interactive; i.e., the decision maker would interact with the DSS as he/she made a decision (e.g., [Gorry 71]). While some DSS are used in this manner, it has become clear that at least as many others are not (e.g., [Keen 76], [Ginzberg 82]). User interaction with a DSS can range from true on-line dialog to intermittent batch use.

The second aspect of the access pattern is the user community, i.e., who uses the DSS. This can range from one person to many people. Expertise in the system (e.g., as a result of frequent use), computer usage in general, or the problem area can vary from almost none to extensive. A user may be the ultimate decision maker, an interested party (to the decision), or a chauffeur who supports the 'real' user. Another type of DSS user defined by [Alter 80] is the 'feeder,' a person who uses the system solely to provide it with data it requires. One dimension of the user community which has been discussed in the DSS literature, but which we choose to omit, is cognitive style. We agree with Huber's [Huber 83] assessment that not enough is known about the impact of cognitive style to make design relevant prescriptions.

The final aspect of the access pattern is the relationship to 'neighboring' information systems. A DSS may interface with other computer-based systems to acquire its source data or dispose of its output (data, models, suggestions, instructions). Most early DSS were entirely free-standing and had no direct interactions with any other systems. This pattern began to change as DSS which required large operational databases were developed (e.g., [Berger 77], [Laning 82]), and is likely to change further as more complex DSS which may draw on common data and model bases are developed. Each system in such a network might require data produced by other systems as well as produce data which those other systems use. Clearly, this type of interaction among DSS implies a substantially different environment from that faced by stand alone DSS.

4. DSS Role and Function

The objective or purpose of DSS has long been defined as support of a decision making process. There are, however, many ways to support decision making, and each has implications for DSS design. For example, Alter [Alter 80] identifies a range of DSS types which are characterized by the type of support they provide to users: raw data, general analysis capabilities, simple representational (e.g., accounting) models, causal models, solution suggestions, and solution selections. One should note the hierarchical relationship among these levels of decision support; each level contains and adds upon the previous levels.

Another, somewhat simpler, way to view support is by the degree to which it is generalized vs. particularized. DSS can provide support which is tailored to a specific problem, or even a specific individual's view of a particular problem. At the other extreme, DSS can provide general analytic capabilities which will support multiple decision makers in multiple, though related, problem contexts. There are, of course, many possibilities between these extremes.

These variations in support concern its content. Equally important are variations in the process. Most DSS literature views the support provided as enhancement of individual cognitive processing capabilities (e.g., [Ginzberg 83]), or the facilitation of learning (e.g., [Keen 83]). Other ways to support decision making include support of communication or coordination among parties involved in the decision process, and control or influence over the outcome of a decision process (see, e.g., [Ginzberg 82]).

5. DSS Components

Components should not be confused with modules, formal sub-systems or parts, but rather represent a functional breakdown of the system. The actual assignment of system functions to, say, software modules is mainly a question of arrangement.

DSS design literature seems to agree about three major functions, tasks, or conceptual components necessary to fulfill the DSS's role, namely: (see Figure 5-1)

* the management of the dialog between the user and the system;
* the management of data; and
* the management of models.

The justification for each of these components can readily be seen if we consider the DSS environment and role. The most pervasive and fundamental aspect of the DSS Environment is people, and the Dialog Management component embodies the specialized functionality necessary to handle the system's interaction with its human users. The Data Management component reflects another fundamental aspect of the role of DSS--all levels of decision support are based on access to a set of data. The need for a Model Management functionality is derived from the nature of the task to which DSS are applied, that is, tasks which are only partially structurable and consequently require the manipulation of underspecified, evolving models.

There is a less agreement about the specific functional content of

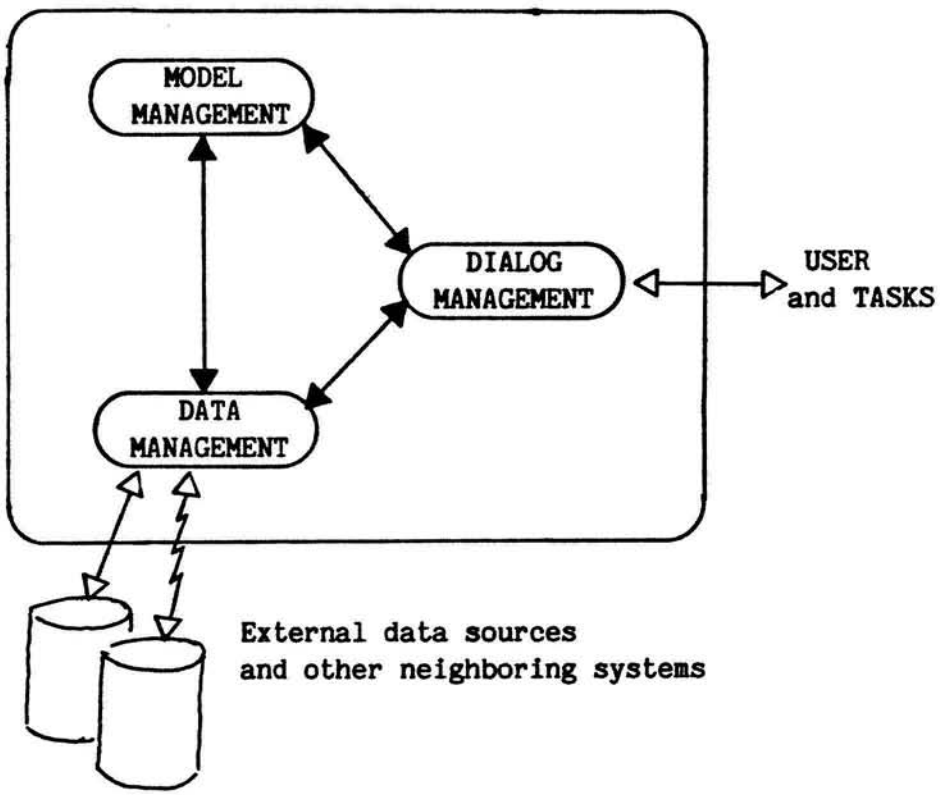

Figure 5-1: DSS Design--Major components

these components (e.g. [Sprague 82], and [Bonczek 82]). In this section we outline our view, which is closer to that of the former, while incorporating some features of the latter. In Section 8 we examine major relationships between the various components and other aspects of the system, especially environmental conditions or constraints.

The dialog between the user and the system provides the framework in which outputs are presented, which in turn defines the context for subsequent inputs. This suggests three dialog management functions, namely

1. a user interface to handle the syntactic aspects of the interaction (e.g., the devices, the physical view, and the style of interaction);

2. a dialog control function to determine the basic semantics of DSS interactions, and maintain the interaction context, which could range between strictly system defined or loosely "user-driven"; and

3. a request transforming function to provide the necessary (two-way) translations between users' vocabulary and the specific modeling and data access repertoire of operations.

The management of data, i.e., the ability to store, retrieve, and manipulate data is fundamental to any service that a DSS provides. This component maintains the factual basis (including possible links and associations) of the DSS. The specific functions required for the management of data in a DSS are

1. a database management system (DBMS) to provide a high-level access mechanism to data in the database,

2. a data directory to maintain the database's data definitions, and further description of the types and sources of data in the system,

3. a query facility to interpret requests for data (from either of the other major components), determine how these requests could be filled (consulting, possibly, the data directory), formulate the detailed and DBMS-specific requests for data, and finally return the results back to the issuer of the original request, and

4. a staging and extraction function for the access to external sources of (especially historic) data, and connecting the DSS with its possibly multiple external data sources.

The mechanism for explicit management of models and modeling activity is what sets DSS apart from more traditional information

processing systems. The ability to invoke, run, change, combine, and inspect models is a key capability in DSS and therefore a core service. Any support beyond direct access to raw data invokes the application of a model. In particular, inferential retrieval of data from the database [Bonczek 82] is achieved through a model driven process. The state of development of Model Management functionality lags far behind the other two components, but is an active research area (see e.g., [Miller 83], [Konsynski 82]).

The ideal model management facility would provide:

1. a model-base management system (MBMS) to generate, retrieve, update parameters, and restructure models, including a "model directory" to maintain information about available models;

2. model execution to control the actual running of the model, and to link models together when the need for integration arises;

3. a modeling command processor to accept and interpret modeling instructions as they flow out of the dialog component, and internally route them to the MBMS or the model execution function; and

4. a database interface to retrieve data items from the database for running models, and eventually store model outputs in the database for further processing, perusal, or as input to other models.

6. Arrangement of DSS Components

The systemic view of DSS suggests that architecture (i.e., component arrangement), like components, has to be justified in the broader context of environment and role. Moreover, architecture cannot be independent of the available resources. For instance, some DBMS

already contain a dialog management function; therefore, the explicit interaction between such separate components is eliminated.

In general the linkages among the DSS components, the nature of these linkages, and especially the justification for these linkages in terms of environment and role have been treated in the literature in an extremely limited fashion. For instance, [Sprague 82] introduces four generic architectures: the "Network" and "Bridge" for multi-DSS situations, and the "Sandwich" and "Tower" where access to data is only possible through a model. The discussion of these architectures completely fails to embed them in the overall context of the DSS; the appropriateness of each arrangement in various decision situations is never explicated. Advantages and drawbacks of the four architectures are related solely to ease of construction and the internally oriented concerns of system engineering. In this section we discuss two examples of DSS architecture and their environmental determinants. Further discussion of the links between environment, role and architecture is in Section 8 below.

The "Sandwich" arrangement is depicted in Figure 6-1. Note that it differs from the generic structure by the absence of any direct interface between Dialog Management and Data Management. The mediating role of the model implies the existence of such a model (or a number of models). This architecture is, therefore, relevant in decision situations which have been structured.

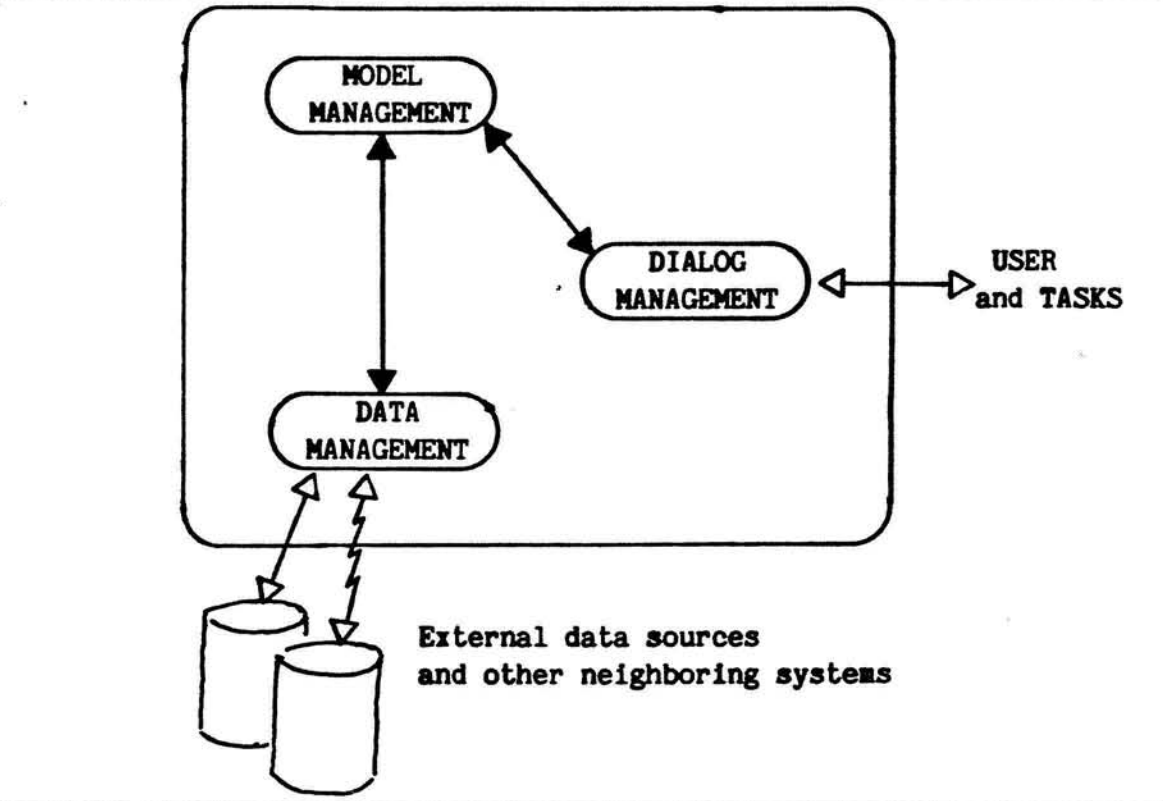

Figure 6-1: DSS Design--A "Sandwich" arrangement

None of the suggested architectures addresses itself directly to situations where models either do not yet exist, or where the relationships between the data and the models are very complex or underspecified. We refer to such situations as "Exploratory", and propose an alternative architecture for it (Figure 6-2). This arrangement gives the user maximum control over both data and models, and requires that any desired linkage between data and models will be maintained "manually".

The "Bridge" and "Network" architectures described in [Sprague 82] are merely specific implementations of the generic DSS structure of

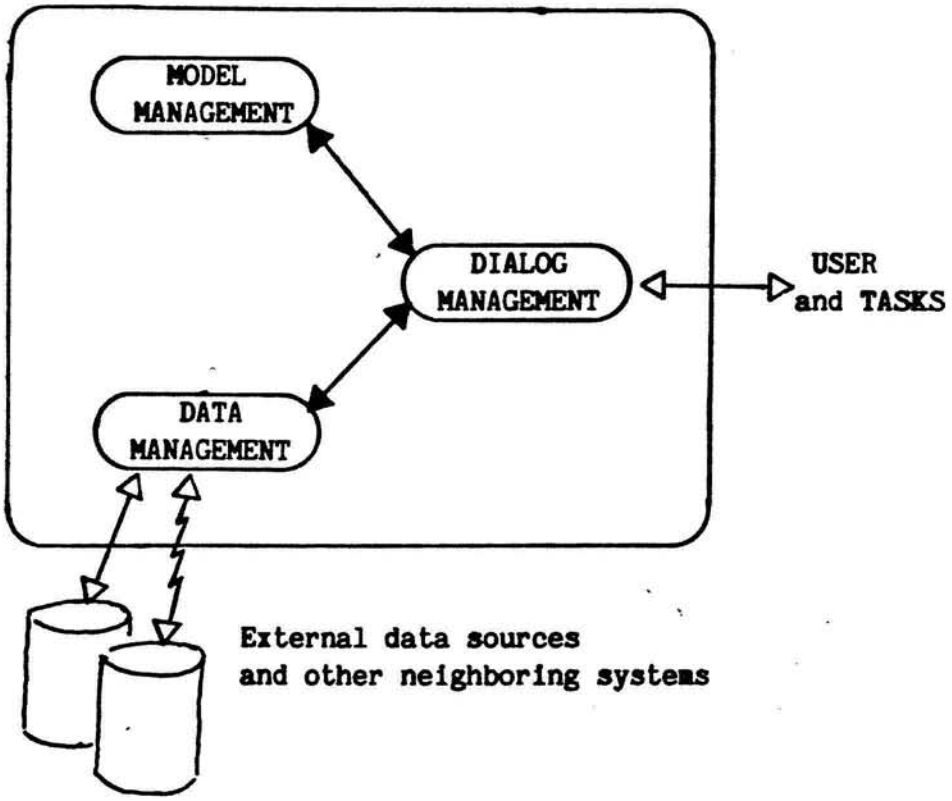

Figure 6-2: DSS Design--An "Exploratory" arrangement

Figure 5-1. Treating these as distinct architectures confuses the notion of components (needed functionality) and resources (the way functionality is provided).

7. DSS Resources

It is only once the DSS has been designed, i.e., the components and their "ideal" arrangement selected, that resources should be considered. The key questions are (1) how can the proposed DSS best be realized; i.e., how "close" to that ideal can a feasible system come? and (2) which resources should be employed to build and support the DSS? The resources available for DSS fall into four major categories: hardware, software, people, and data.

The hardware resources typically include processors (e.g., mainframe, minis, and micros), terminals (e.g., hard copy, CRT, graphics and plotters), storage media, and communication networks. None of these resources is unique to DSS, and all are applicable to other systems as well. Some of the specific linkages between DSS role and hardware configuration are discussed in Section 8.

The greatest range of resources, particularly of resources developed specifically for DSS, is in the software area. We can identify four major classes of software resources:

1. General purpose programming languages
2. DSS tools (e.g., DBMS, modeling languages, dialog managers, and 'arrangement' packages)
3. DSS generators
4. Generalized DSS

Ultimately, all DSS software is built upon general purpose programming languages, and any DSS can be built directly using only such software tools. General purpose programming languages, even very high level languages like APL, provide only limited leverage for the development of DSS, and we shall not discuss them any further.

DSS Tools are single-function building blocks which can be used to construct DSS; that is, they address only one of the major DSS components/functions. Thus, four key types of DSS tools are database management systems (DBMS), model management systems, dialog management

systems, and arrangement packages. Arrangement packages (often referred to as "software environments" or "windowing packages" [BusinessWeek 83]), address the interfaces among the other sub-systems but not the content of any of them.

DSS Generators are, in essence, a collection of DSS tools. That is, they are software packages which address all three DSS functionalities (at least to some extent), and which can be used to construct Specific DSS, systems tailored to specific problems or problem situations. A typical example of a DSS generator is a modeling package (such as EXPRESS, EMPIRE, or IFPS) which includes some data management capabilities as well as dialog/presentation facilities.

The next category of DSS software resources is the Generalized DSS. Generalized DSS provide support for a class of problems. For instance, a generalized DSS for scheduling and assignment is conceivable, in which Professors could be assigned to classes, operators to shifts, and specialists to projects. All these instances share a core of concerns, and more importantly, a model base. Generalized DSS fall between DSS Generators and Specific DSS in terms of generality and the amount of effort required to apply them to specific problems. They provide capabilities which can be applied directly to a decision problem, unlike DSS Generators which are only used to develop specific DSS. Their generality, however, means that some of the linkages and transitions which would be built into a specific DSS must be performed manually. Generalized DSS are becoming an increasingly popular mechanism for

providing decision support (note the surge of offerings in the PERT/CPM systems for micro computers), as more and more people want decision support tools, but specifically tailored DSS either are not readily available or are too expensive to provide in resource constrained situations.

The People resource for a DSS is different (conceptually) from the people aspect of the system's environment. This resource provides one of the clearest examples of the substitutability of alternative resources in the design of a DSS. It is almost always necessary to acquire data from other systems for the DSS. This staging mechanism can be a hardware bridge, a hardware/software bridge, or a human bridge. The issue is to recognize that there are many roles in developing and operating a DSS which can, but need not, be played by humans; the choice should depend on the availability and relative cost of both human and other resources.

Data resources include the various sources of data available to the DSS. This includes internal, operational data as well as externally available data. The latter may be from generally accessible databases or the result of special studies.

8. DSS Design--Linking the System Aspects

In this section some of the major contingencies between a DSS's internal structure and its role, environmental conditions, and resource availability are explicated. A more exhaustive identification and

validation of such linkages is clearly needed, but is beyond the scope of this paper.

Task structurability directly impacts the model component of the DSS. Models can be maintained in the MBMS in variety of forms, ranging from subroutines (where the model-base equals a software library), to more abstract and manipulable forms, which equates models to data [Konsynski 80]. Where tasks are highly structurable, procedural model specification (e.g., as subroutines) is appropriate, while low structurability suggests a more flexible, declarative form of model definition (e.g., production rules).

Low structurability also suggests that the Model execution function should interact directly with the dialog component, both for eliciting user provided parameters and for handling user interventions (e.g., intermittent perusal of variable values). A related issue is the design of the dialog control function: "system-prompted" dialog fits situations with relatively high levels of structure, while completely "user-driven" sessions are appropriate where no predefined sequence of activities can be specified.

One obvious impact of task level is on the needed data resources and facilities for accessing them, i.e., on the staging function in the data management component. Operational control tasks are likely to require access to current operational data. The DSS, then, must provide an on-line linkage to data files maintained by operational systems.

Strategic planning tasks, on the other hand, may require access to external data sources, but probably not on a continuously on-line basis. These DSS must include a mechanism for remote access and selective extracting of data from extra-organizational sources.

Task level may interact with the type of support to impact the selection of hardware resources. For example, to support senior managers in a strategic decision situation (e.g., new product selection), a stand alone microcomputer based DSS may be very appropriate; little data is needed, the data does not come from the operational stream, and the system is meant (primarily) to provide cognitive support to individual managers. On the other hand, providing support for real-time operational decisions (e.g., production control) in a complex production environment would best be accomplished on a mainframe based DSS with good data communications capabilities. This type of support requires a substantial quantity of quite volatile, operational data, and the primary purpose is to assure coordination among the multiple parties to the decision.

The decision process stage can impact the need for a directory. The Data Directory in a DSS provides the basis for answering questions about the availability of data items, their sources, and their exact meanings. As such it is most important in systems which support the intelligence phase of the decision making process, where data exploration is a paramount concern.

The functional area determines the set of verbs and objects useful in the specific situation of problem solving. Both the request transformer and the dialog control should reflect this user vocabulary.

The desired interaction mode has obvious implications for both resources and the arrangement of components. On-line interaction, for instance, limits the selection of employable resources, and calls for tighter inter-module linkages (i.e., immediate exchange of messages is necessary), while batch oriented design can use data files on secondary storage as a possible medium for message exchange.

The design of the user interface should be determined by the nature of the user community, e.g., proficiency of users, frequency of use. This should be the basis for selecting the interaction style which the interface will accomodate (e.g., menu-driven or Q/A-driven dialog).

Neighboring systems have the greatest effect on the Staging function--its structure should directly reflect the nature of the data sources in the DSS's environment. For instance, if remote databases are included, some data communication facility is necessary. Likewise, the selection of a data model (i.e., the data structures in the database and valid operations on it), is contingent upon the structure of the available external data.

As noted earlier, data related services underlie all DSS roles. Nevertheless, DSS roles differ in the extent of model intensity they imply. Consequently, the arrangement of DSS components should be

contingent upon the model intensity of the service to be provided by the system. For instance, the Sandwich architecture is a model centered architecture which is appropriate for directing the user's decision making process. It is clearly more applicable to the higher levels in Alter's hierarchy of support types where models are more dominant. The Exploratory architecture--as its name suggests--is appropriate for those situations where little imposed structure is possible (or desirable), the lower levels of the support hierarchy.

9. Discussion: Implications of the Systemic View

In this section we sketch out some implications for DSS research, development, and educational curricula which are suggested by the Systemic View. The fact that a number of interesting issues are indeed raised demonstrates the usefulness of this approach.

Perhaps the clearest implication of the systemic view for DSS design is the necessity to take an "outside-in" approach. As is articulated in Section 8 above, the selection of components and their arrangement (the "inside") must follow from an understanding of the environment and role (the "outside"). Recent books on DSS design provide only limited guidance for relating environmental conditions to the specific system components or their arrangement.

The focus of early DSS literature (e.g., [Gorry 71]) on the system's environment and role was critical. These elements are central to explaining why DSS differ, and probably will be built differently,

from traditional information systems. More recent DSS literature has focused on resources, components, and architecture (i.e., arrangement) [Bonczek 81], [Sprague 82]. It is only when these elements are considered in the context of environment and role that true understanding and insight can be developed. A change is needed in DSS research strategy, from "coming up with designs" to explicit derivation and justification of components and architectures. Such a change does not imply that study of single system elements should be avoided, only that in any study we must first understand the context.

A key direction for future DSS research is gaining a better understanding of the range of DSS environments and roles. While the practical use of DSS has permeated a substantial array of decision environments--from labor negotiations to military combat situations, the major share of DSS research to date has focused on a limited set of environmental conditions (typically, the intelligence and choice phases of organizationally isolated, financial decisions). Section 8 of this paper is a first step towards an explicit DSS design theory in which design is contingent on comprehensive notions of environment and role. Further work is needed, however, to develop a design-relevant taxonomy of DSS environments, identifying key environmental characteristics and the constraints they place on the choice of DSS components, their arrangement, and the resources required.

One environmental characteristic which is likely to be increasingly important for future DSS is the other systems with which the DSS must

interact. There has been only a preliminary attempt to examine DSS that interface directly with other computer-based systems (e.g., [Sprague 82]. The trend towards interconnected systems is, however, quite clear (e.g., [Zmud 83]), and many DSS in the future will be required to interact with other DSS, traditional information systems, etc. Research is needed to understand both the range of potential interaction patterns and the requirements and constraints they will place on DSS design and development.

The systemic view sharpens the distinction between system components and resources. It suggests a methodology for evaluating resources, and a basis for making educated decisions about the consumption of these resources. Given the popular marketing strategy of labeling every product a DSS, it is important that a more critical approach be adopted by potential users. Such an approach is not meant to discredit one product or another but rather to identify the capabilities of each and to serve as a guideline for the selection of resources. In Figure 9-1 we list some popular software resources for micro-computer based DSS, and classify them into the levels presented above[1].

There is substantial variety of software resources, and Figure 9-1 represents just a sample of what is available. At the tool level, there

---

[1]The "General Purpose Programming Languages" level has been omitted since there is nothing at this level unique to either DSS or micro-computers.

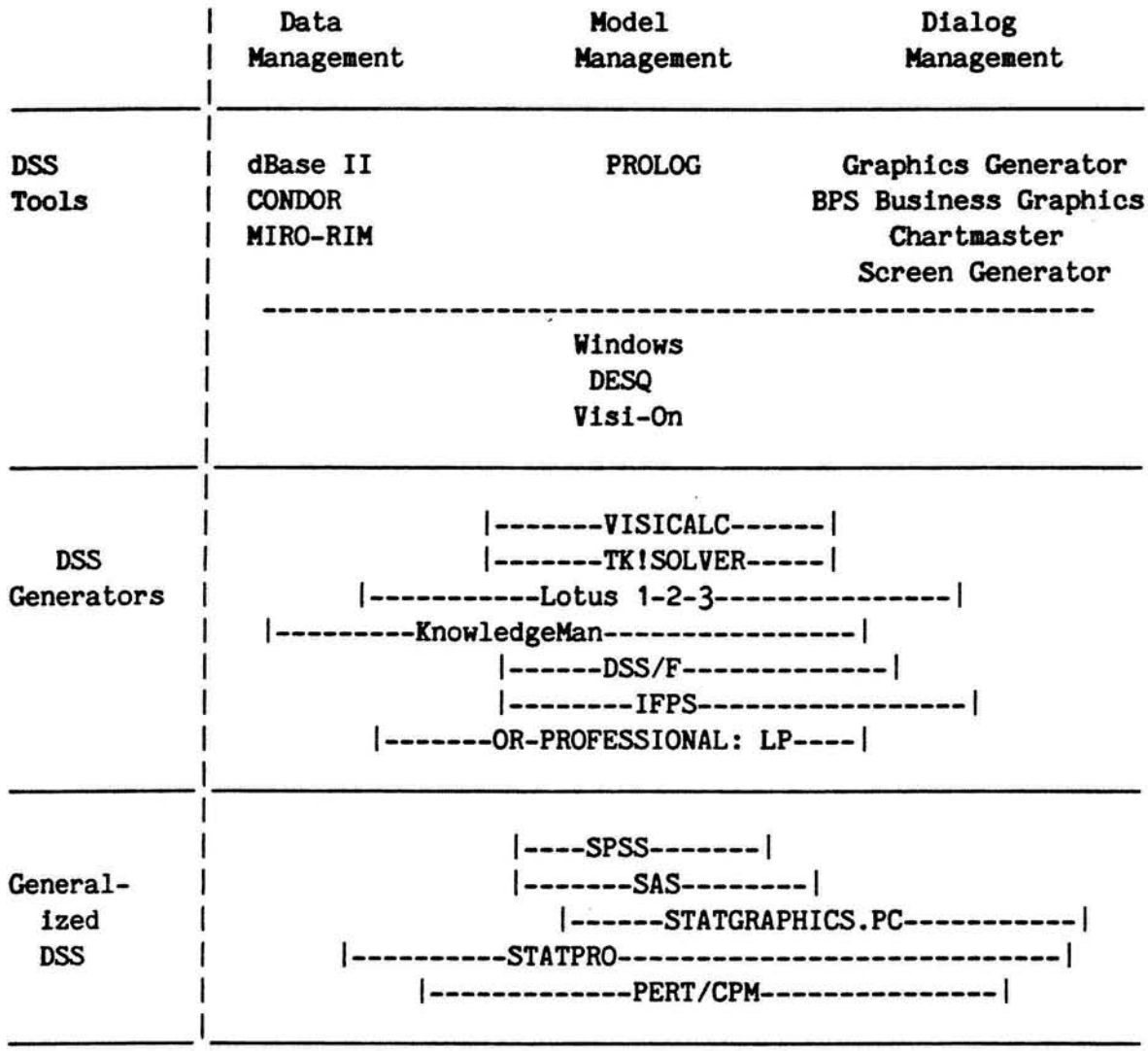

Figure 9-1: DSS Software Resources for MicroComputers

are a number of data management as well as dialog/presentation management packages. There has been relatively little work on developing tools which address model management alone, though perhaps languages like PROLOG fit this description. One emerging tool type is the 'arrangement package' which links the other components together. Microsoft's Windows, Visicorp's VisiOn, Quarterdeck's DESQ, as well as the LISA operating system all fall into this category.

DSS Generators are used to build models. Consequently, all have model management capabilities (though they vary in sophistication), and all address to some extent the data management and dialog management functions. VisiCalc, TK!SOLVER, and DSS/F represent three levels of (increasing) model management sophistication; each also provides minimal data and dialog management facilities. Lotus 1-2-3 provides model management facilities at about the same level as VisiCalc, but more extensive data and dialog management. KnowledgeMan provides even more sophisticated data management, while providing roughly the same level of model and presentation management as VisiCalc.

Generalized DSS are developed around some broadly applicable model (e.g., PERT, statistical analysis). Unlike DSS Generators their implicit models can be applied directly to the user's problem without having to first construct a formal representational model. Like DSS Generators they provide some data management and dialog management facilities. STATPRO, for example, provides some data management and extensive dialog management (including excellent graphics). STATGRAPHICS.PC provides equally good dialog management, but virtually no data management, relying instead on the capabilities of its host language, APL, to provide these facilities.

This scheme demonstrates the use of the systemic view as a resource evaluation methodology. The software resources at each level represent differing degrees of comprehensiveness and integration. Choice among them would depend, at least in part, on the availability of complementary design and development skills.

The recognition that resources are only one aspect of the system suggests that a modification in the focus of DSS curriculum is needed. Too many DSS courses are built around a specific tool or tools (e.g., IFPS). This focus does not provide students with an understanding of DSS nor a basis for being educated consumers of DSS tools. As an alternative, we have used the systemic view as an organizing framework for a DSS course. In it the analysis of the decision situation is the basis for the development of an "idealized" system design, against which available resources are assessed and critically examined. Resources (i.e., available packages or generators) enter the curriculum only at its final stages, and the focus at that point is evaluating the trade-offs between functionality and ease of construction--the core of any decision concerning the employment of resources.

Most DSS literature so far has emphasized the differences between DSS and other, more traditional computer-based systems. The Systemic View helps us to see the similarities. Once we adopt this perspective it becomes apparent that DSS differ from traditional systems in degree, not in fundamental structure. For instance, large scale DSS and 'institutional DSS' are very similar to classical MIS both in terms of environment (e.g., anonymous user) and the constraints this places on design (e.g., run-time efficiency). This suggests that there is much that DSS researchers and developers can learn from prior experience with other types of computer-based systems, and that a cumulative experience <u>across</u> types of information systems is both possible and desirable.

This recognition that DSS differ from other systems only in degree also implies that evolving types of computer-based systems might be similar to DSS and should not be treated as totally novel. A case in point is Expert System (ES). We have observed that some ES developers are treating these systems as a completely new phenomenon, and are struggling afresh through the classical problems of IS/DSS development [O'Conner 84]. The systemic view makes it clear that ES bear many similarities to DSS. Their environments and roles are quite similar, and it is only a change in components and resources which differentiates them. It is therefore inappropriate to discard the (very relevant) experience with DSS.

10. Conclusion

One may argue that the systemic view has been embedded in DSS all along. While this may be true, this view had receded too far from the surface. Adopting the systemic approach means giving explicit emphasis to the relationships among the system aspects, a subject which has received only limited attention both from practical and theoretical view points, even though it is the essence of design activity.

[Ginzberg 82] commented on the migration of DSS definitions from environment/role focus to component/arrangement focus. The systemic view responds to that 'tension' and brings the two under a single comprehensive framework. Of course, much research is still needed to further clarify, complete, and validate the proposed systemic framework and examine its value for DSS research and practice.

References

[Ackoff 81] Ackoff, R.L.
Creating The Corporate Future.
John Wiley and Sons, New York, N.Y., 1981.

[Alter 80] Alter, S.
Decision Support Systems: Current Practices and Continuing Challenges.
Addison-Wesley, Reading, Mass., 1980.

[Anthony 65] Anthony, R.N.
Planning and Control Systems: A Framework for Analysis.
Harvard University GSBA, Studies in Management Control, Cambridge, Mass., 1965.

[Bennett 83] Bennett, J.L. (editor).
Building Decision Support Systems.
Addison-Wesley Publishing Co., Reading, Mass., 1983.

[Berger 77] Berger, P. and Edelman, F.
IRIS: A Transaction-Based DSS for Human Resources Management.
Database 8(3):22-29, 1977.

[Bonczek 81] Bonczek, R.H., Holsapple, C.W., and Whinston, A.B.
Foundations of Decision Support Systems.
Academic Press, New York, 1981.

[Bonczek 82] Bonczek, R.H., Holsapple, C.W., and Whinston, A.B.
The Evolution from MIS to DSS: Extension of Data Management to Model Management.
In M.J. Ginzberg, W.R. Reitman, and E.A. Stohr (editors), Decision Support Systems, pages 61-78. North-Holland, Amsterdam, 1982.

[BusinessWeek 83] BusinessWeek.
A Fierce Battle Brews Over the Simplest Software Yet.
BusinessWeek :114-115, November 21, 1983.

[Churchman 68] Churchman, C.W.
The System Approach.
Dell Publishing Co., Inc., New York, 1968.

[Emery 69] Emery, J.C.
Organizational Planning and Control Systems: Theory and Technology.
Macmillan Publishing Co., Inc., New York, N.Y., 1969.

[Ginzberg 82] Ginzberg, M.J., and Stohr, E.A.
Decision Support Systems: Issues and Perspectives.
In M.J. Ginzberg, W.R. Reitman, and E.A. Stohr (editors), Decision Support Systems, pages 9-32. North-Holland, Amsterdam, 1982.

[Ginzberg 83] Ginzberg, M.J.
DSS Success: Measurement and Facilitation.
In C.W. Holsapple and A.B. Whinston (editors), Data-Base Management: Theory and Applications, pages 367-387. D. Reidel, Dordrecht, The Netherlands, 1983.

[Gorry 71] Gorry, G.A., and Scott-Morton, M.S.
A Framework for Management Information Systems.
Sloan Management Review 13(1):55-70, Winter, 1971.

[Gorry 83] Gorry, G.A., and Krumland, R.B.
Artificial Intelligence Research and Decision Support Systems.
In J.L. Bennett (editor), Building Decision Support Systems, pages 205-219. Addison-Wesley Publishing Co., Reading, Mass., 1983.

[Huber 83] Huber, G.P.
Cognitive Style and DSS Designs: Much Ado About Nothing?
Management Science 29(5):567-579, May, 1983.

[Keen 76] Keen, P.G.W.
'Interactive' Computer Systems for Managers: A Modest Proposal.
Sloan Management Review :1-17, Fall, 1976.

[Keen 80] Keen, P.G.W.
Adaptive Design for Decision Support Systems.
Data Base 12(1/2):31-40, Fall, 1980.

[Keen 83] Keen, P.G.W, and Gambino, T.J.
Building A Decision Support System: The Mythical Man-Month Revisited.
In J.L. Bennett (editor), Building Decision Support Systems, pages 133-172. Addison-Wesley Publishing Co., Reading, Mass., 1983.

[Konsynski 80] Konsynski, B.
On the Structure of a Generalized Model Management Systems.
In Proc. of the 14th Annual Hawaii International Conference on System Sciences, pages 19-31. Western Periodicals, North Hollywood, CA, January, 1980.

[Konsynski 82] Konsynski, B., and Dolk, D.R.
Knowledge Abstractions in Model Management.
In Proceedings of DSS-82, pages 19-31. June, 1982.

[Laning 82] Laning, L.J., Walla, G.O., and Airaghi, L.S.
A DSS Oversight--Historical Databases.
In G.W. Dickson (editor), DSS-82 Transactions, pages 87-95. June, 1982.

[Little 70] Little, J.D.C.
Models and Managers: The Concept of Decision Calculus.
Management Science 16(8):B466-B485, April, 1970.

[Miller 83] Miller, L.W., and Katz, N.
Model Management Systems to Support Policy Analysis.
Technical Report DS-WP 82-11-01, Decision Sciences Dept., Univ. of Penn., April, 1983.

[Moore 80] Moore, J.H., and Chang, M.G.
Design of Decision Support Systems.
Data Base 12(1/2):8-14, Fall, 1980.

[O'Conner 84] O'Conner, D.E.
Using Expert Systems to manage Change and Complexity in Manufacturing.
In W.R. Reitman (editor), Artificial Intelligence Applications for Business. Ablex, Norwood, NJ, 1984.

[Simon 60] Simon, H.A.
The New Science of Management Decision.
Harper & Row, Publishers, New York, 1960.

[Sprague 82] Sprague, R.H., Jr., and Carlson, E.D.
Building Effective Decision Support Systems.
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

[Zmud 83] Zmud, R.W.
Large-Scale Interconnected Information Systems: Design Considerations for Promoting Organization Adaptation and Organizational Adaptability.
Technical Report, School of Bus. Ad., U. of N. Carolina, Chapel Hill, NC, September, 1983.
Presented at the working conference on Large-Scale Interconnected Systems, Athens, OH, October 10-11, 1983.