Machine Intelligence and Autonomy for Aerospace Systems

Edited by
Ewald Heer and Henry Lum

Progress in Astronautics and Aeronautics

Martin Summerfield
Series Editor-in-Chief

Volume 115

American Institute of Aeronautics and Astronautics, Inc.
Washington, D.C.

Library of Congress Cataloging in Publication Data

Machine intelligence and autonomy for aerospace systems
edited by Ewald Heer, Henry Lum.
(Progress in astronautics and aeronautics; v.115)
Includes index.
1. Space stations—Automation. 2. Artificial intelligence. 3. Robotics. I. Heer, Ewald. II. Lum, Henry, 1936- III. Series.
[TL507.P75 vol. 115 [TL797] 88-37336
629.1 s—dc19
[629.47'028'563]
ISBN 0-930403-48-7

Chapter 8

A Design Methodology for nowledge Base Management Systems

Walter Truszkowski*
4 *Goddard Space Flight Center, Greenbelt, Maryland*
and
B. Silverman,† H. Hexmoor,‡ and R. Rastogi‡
Intellitek, Inc., Rockville, Maryland

Introduction

er addresses major issues involved in the development and use e Base Management Systems (KBMS), particularly a develop- and associated design methodology, the context being auto- l-control systems for near-Earth scientific spacecraft typical of ıal control centers at NASA Goddard Space Flight Center tification of the major ground-support functions, their inter- and the data and information of their domains and co- provide a real-world operational backdrop. A major premise apter: a need both to minimize effort (that could be wasted in prototyping) and avoid prototyping a large-scale KBMS into tions treat, in order, an assessment of the GSFC domain and MS applications; the KBMS paradigm and the KBMS devel- el; the KBMS design methodology under development; an plying the methodology on an already completed large-scale ype; and future directions.

Ground-Control System and KBMS Requirements

cecraft missions require coordinating many functions in tions. As the NASA missions become larger, more sophisti- ore complex in decision-making, reliance on efficient and and information handling becomes ever more critical,[1] and e believe, need for new ground-system architectures involving hine intelligence and autonomy. A brief overview of a generic ol system similar to the ones in use at GSFC will give an idea x environment for current research and experiments.

ıta Systems Technology Division/Code 520.
, Professor, Artificial Intelligence, at George Washington University.
ligence Programmer.

The Fig. 1 block diagram depicts the NASA end-to-end data and information system: from components involved during the capture of data by a spacecraft sensor to its conversion into information and use by principal investigators and others—a complicated and intensive data-transformation.

Functional Components of a Mission Ground-Control System

Figure 2 presents a model of a mission ground-control system with typical functions,[2] as follows: a) mission planning, b) resource scheduling, c) command management, d) command transmission, e) data capture and preprocessing, f) data analysis, g) monitoring and control, h) data archiving, i) data distribution, and j) simulation and predictions. Some, like monitoring and control, while being data-intensive, are also highly information-intensive, involving a great deal of interaction on the part of the mission support personnel.

Data/Function Interoperability

Figure 3 partially illustrates the complexity of the overall ground control by indicating the use of some of the many interacting data sets. For example, data sets D4 and D5, which are time-reference and sensor-orientation data, respectively, are utilized by the function F2, i.e., calibrate and locate observations in order to produce/modify the data set D3, the sensor-calibration data. The figure also illustrates the fact that some functions depend upon others and that some ground-control objectives can be realized only by a coordination of several system functions.

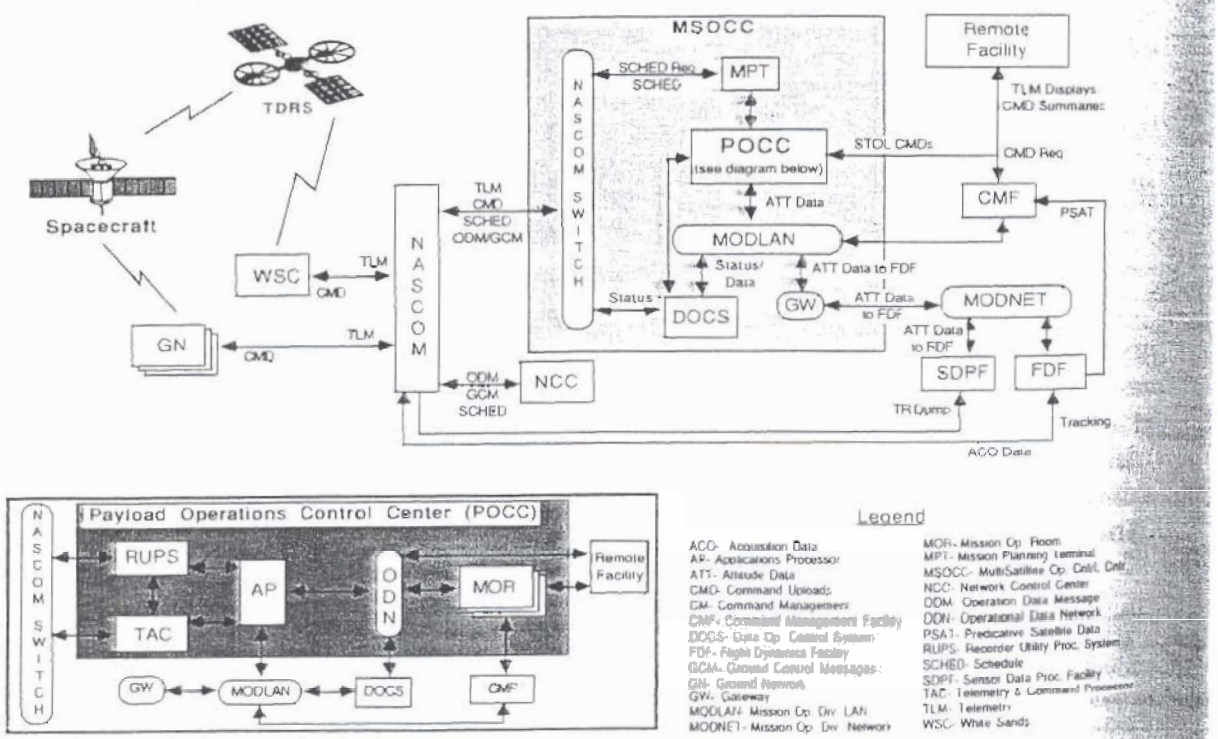

Fig. 1 NASA end-to-end data/information system.

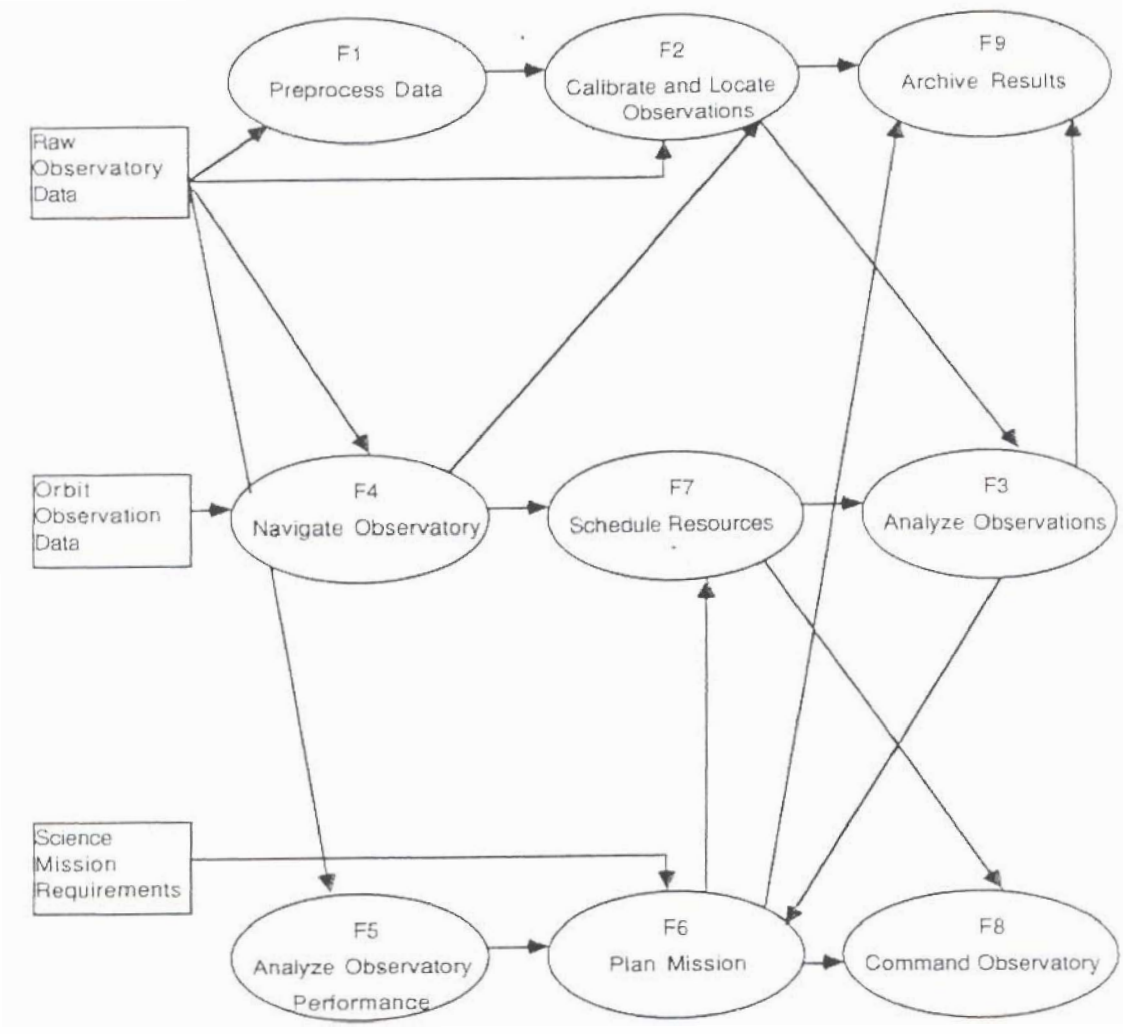

Fig. 2 Generic ground-system functional model (top-level).

Human Tasks, KBMS Issues

Many ground-operations pesonnel are involved in supervisory control involving complex situations and the controller's subjective preferences. Some tasks may involve initiating a restoration of normal system state, or identifying alternative functional paths to compensate for the effects of a change, or identifying patterns of critical variables related to safety. Thought processes in such supervisory control can be described in terms of data, models, and strategies:[3] data as mental representations of system state; models as system anatomy or functional structure; strategies as the unified exercise of sets of models, data, and critical process rules.

Figure 4 shows operators and analysts to be in feedback loops with various mission control center processes. There are successive levels of detail in data and information handling by humans at the corresponding levels of feedback loops of Fig. 4. The operators monitor the state of the system and keep it within specified limits. Although skilled, the operators stick to well-learned activities about their domain, When a problem goes beyond their scope, the operators usually turn it over to system analysts at

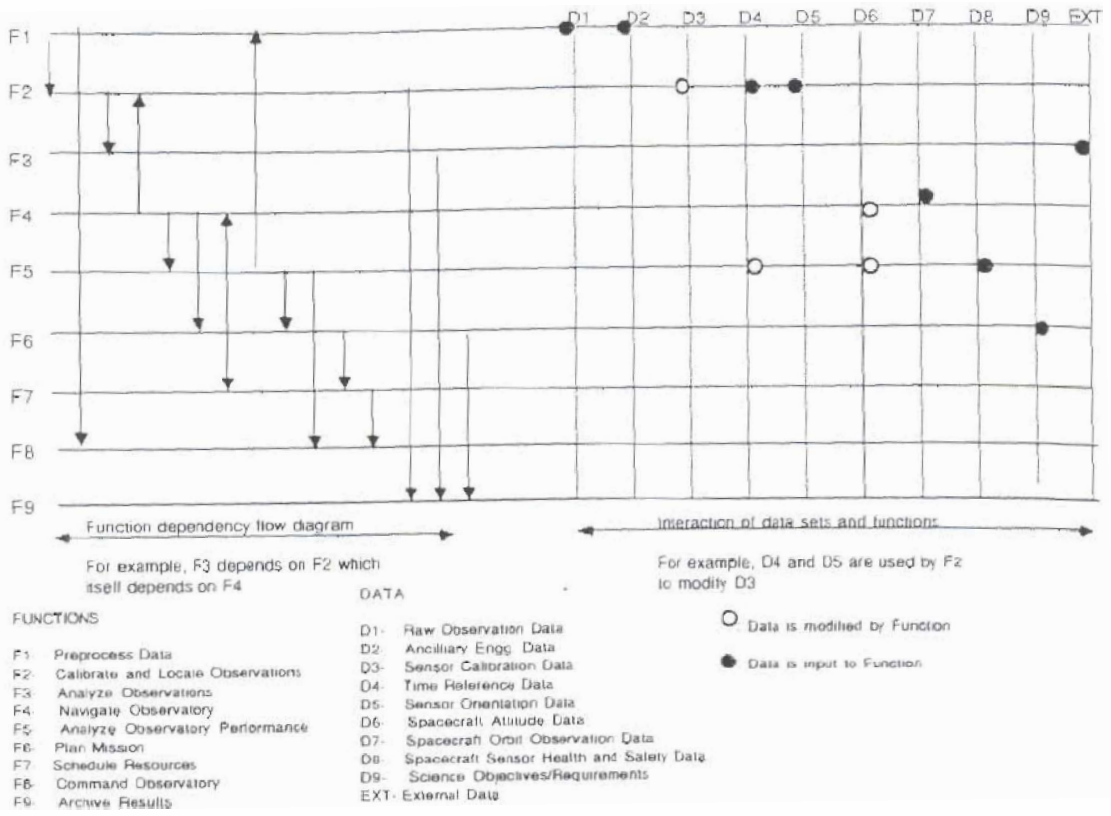

Fig. 3 Function interdependency via data sets.

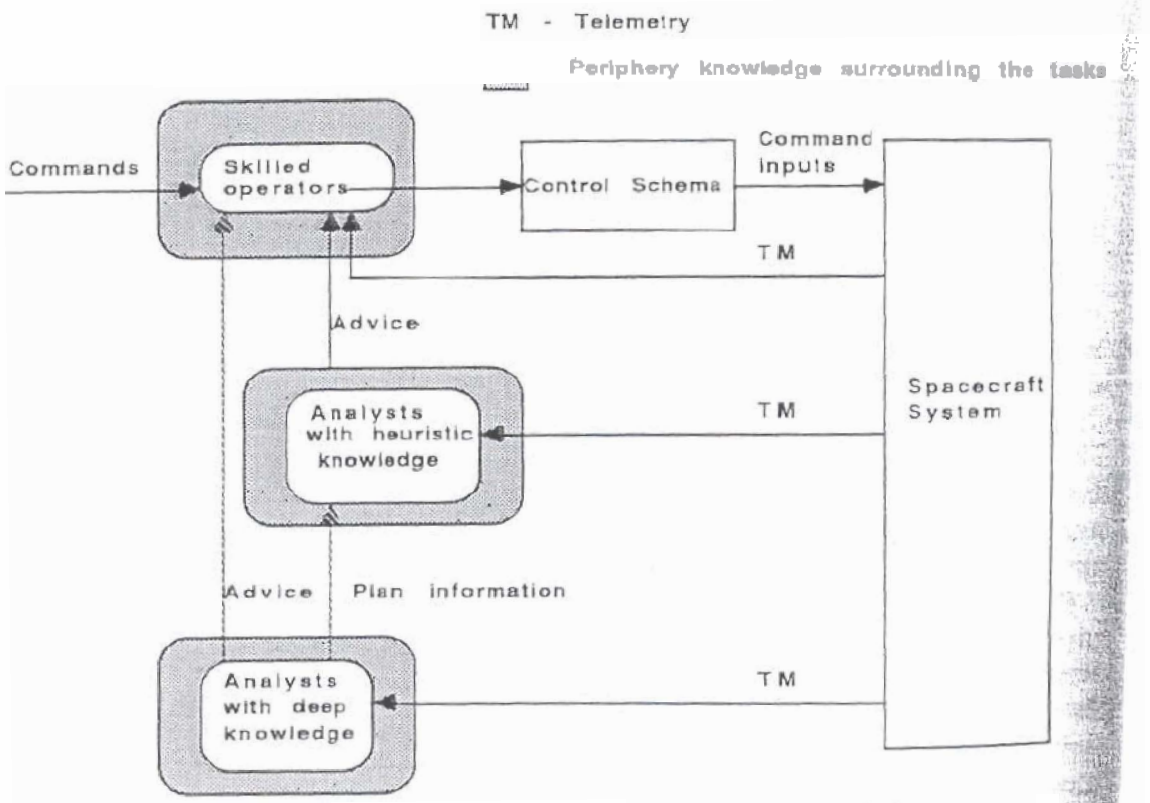

Fig. 4 Human operators and analysts in feedback loops.

the next outer feedback loop of Fig. 4. The system analysts generally use heuristics and some design knowledge for problem solving. For a detailed understanding of the problem, the system analysts refer to deep design knowledge contained in manuals and other technical literature, as well as immediately bringing to bear their experience.

The KBMS should tier detailed information about system goals and states (parameters) under observation. The detailed information about parameters should be invisible to the operators and should be intended for the analysts and personnel with more knowledge of the system. This implies that the KBMS computing modules should have access to and be specialized to deal with only certain types of information. It also implies that the knowledge sources should be organized into hierarchies and levels of abstraction for sequestering information and partitioning computing elements. Of course, to enable a common goal and mutual coordination, control schemes need centralization to some extent, and must be sensitive to inputs from all locally intelligent modules or computing elements. Table 1 presents high-level KBMS requirements suggested by the operational environment.

Table 1 High level summary of KBMS requirements, as suggested by KBMS Study

GSFC Ground Control System Requirements	KBMS Requirements
1. Attributes like Data Rates, Avg. Volume, and Response Time to perform the function	KB Size and Processing Time issues.
2. Data Sets are shared by functional elements	KB Sharing, Cooperative Expert Systems
3. Certain data is meaningful during time windows	KB archiving and regeneration
4. Use of previous learning experiences	Storage of past KBMS sessions, automated knowledge elicitation, acquisition, and learning
5. Interaction of functional elements	KB partitioning, multiple usage of knowledge, multiple representation schemes
6. Involvement of Human Supervisory Positions	Cognitive Protocols and related AI perspectives, hierarchial organization of knowledge sources, information hiding and partitioning of computing elements
7. Presence of mutual goals	Cnetralized control schemes, opportunistic problem solving

Knowledge Base Management Systems

Consider this brief stepwise description of space-mission data processing:

1) Data relating to a spacecraft subsystem power level is extracted from a telemetry datastream.

2) The data presented to the appropriate software system component (function).

3) That component executes the required bit-management, which may include decommutations, reversals, concatenations, etc.

4) The data are converted to engineering units, compared to limits, and the data and comparison results displayed to the operator.

5) The operator views the displayed data.

6) The operator compares his "mental model" with the pattern of displayed values and, if necessary, tries to reconcile differences. This may require commands.

Steps 1–4 of this scenario are called data-intensive since they deal only with the syntactic and structural aspects of data and its representations. Steps 5 and 6 are called information-intensive since they involve the execution of various cognitive processes by an operator who applies

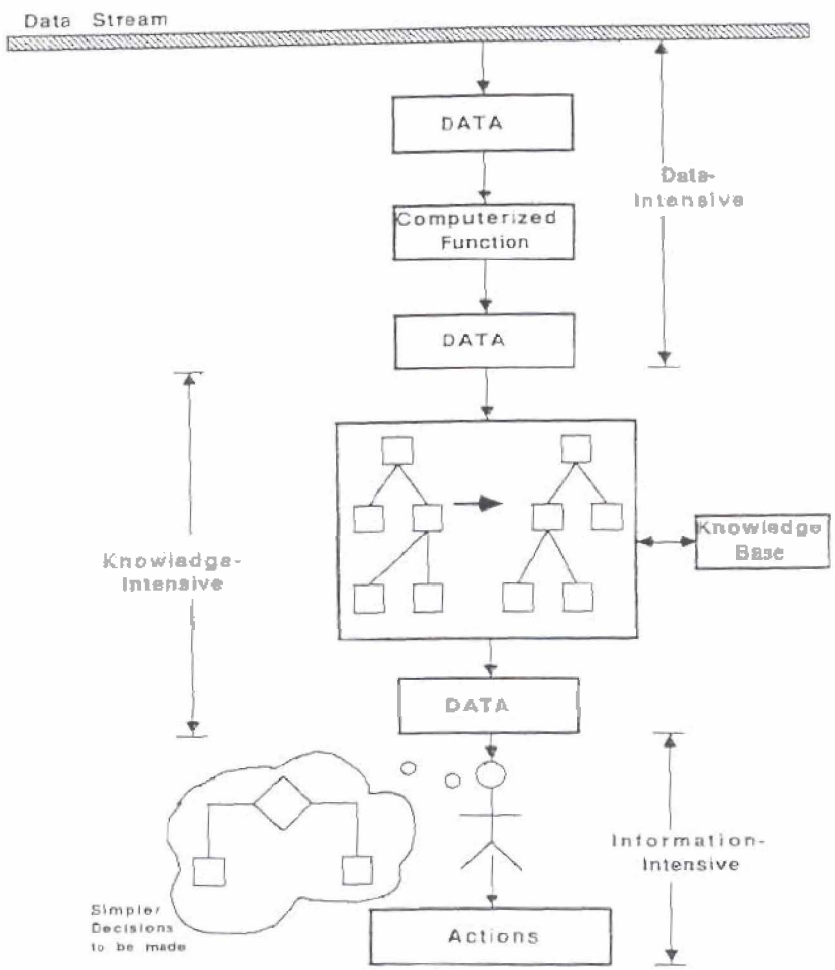

Fig. 5 Computer sharing of knowledge-based and model-based reasoning.

knowledge and models of system behavior along with decision-making based on logical reasoning.

This scenario entails two levels, or perspectives, at which the system functions and their interfaces/interactions may be categorized and studied: the data level and the information level. The scenario highlights the fact that the non-data-level activities are highly manual and could conceivably force the operator to deal with quite complicated and perhaps error-prone and costly lines of reasoning. One approach to alleviating the severe dependence on human inferencing and decision-making would move some of the knowledge and model-based reasoning into the computer system. Figure 5 depicts one possible view of this alternative: the introduction of a knowledge-intensive level that offloads from the operator some preliminary analysis of system behavior. Results of this analysis are presented to the operator for final decisions. The so-called KBMS comprises software at the knowledge-intensive level.

Figure 6 charts subsystems that a large-scale KBMS generally requires. Not every KBMS needs to have all these subsystems; and the quality and level of sophistication per subsystem are soley determined by the problem at hand.

The Knowledge Base Interface supports access to the knowledge base (KB) that contains the explicit domain-specific knowledge—facts, rules, and other information units and inter-information relations. The user interface contains the screen, graphics, and menu routines as well as command, interpretation, and query language interfaces to assist the user in communicating with the system. Through knowledge-engineering "aids" the

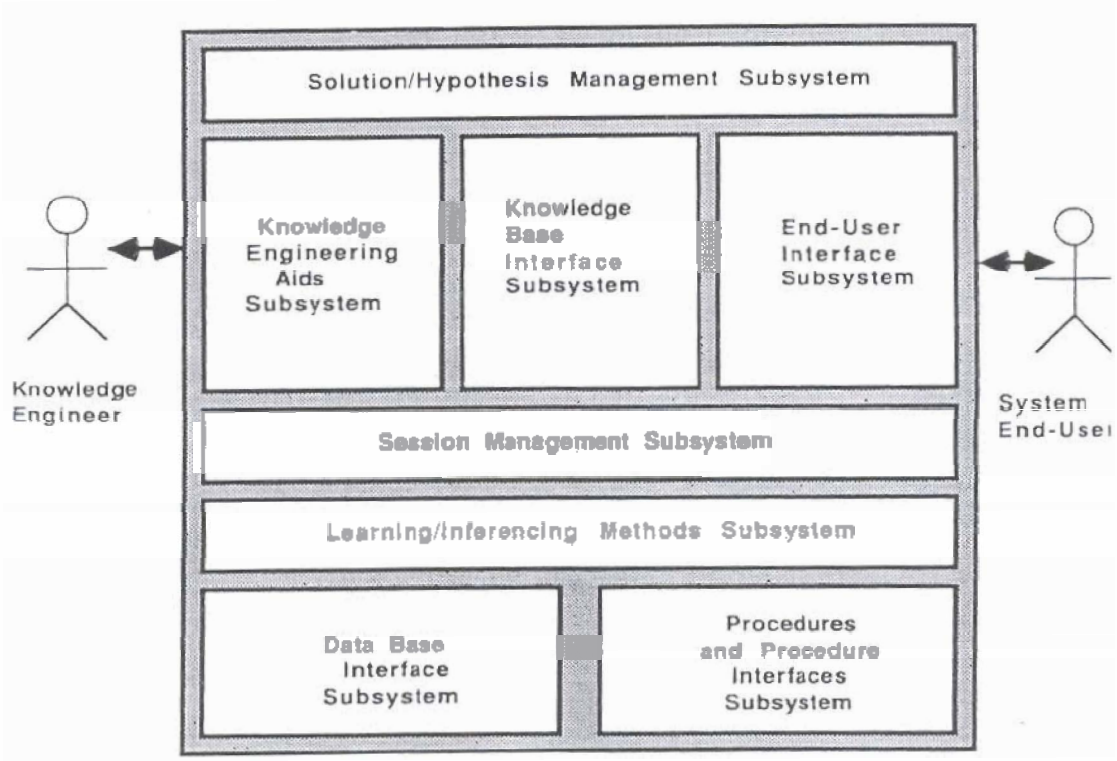

Fig. 6 Subsystems in a KBMS

engineer enters the domain-specific knowledge, defines an end-user interface, and instructs the system about problem-solving strategies to be used by the learning/inferencing and solution/hypothesis management subsystems.

"Session management" comprises utility routines used to aid in handling the outputs of each session held with the KBMS.

Learning/inferencing methods are libraries of routines normally called upon to serve as techniques for traversing the solution space (e.g., chaining techniques, pattern matching, and mathematical calculations).

Solution/hypothesis management evaluates the state of the solution formation and helps the system converge on acceptable solutions.

In environments with large amounts of data/knowledge handling, a KBMS will retrieve data/knowledge and use information-processing procedures from other data/knowledge handling systems. For these functions, two additional subsystems are provided: an interface to allow access to database systems and an interface to use readily available procedures, software, and models from the problem domain.

KBMS Development Model

KBMS design and prototyping usually combine in a single step called "rapid prototyping," because knowledge acquisition and system building interact inseparably and KBMS development necessitates evolutionary system improvement. So design of a KBMS starts with selecting some basic structural concept and then transitions to a more suitable, focused design as the architectural and operational requirements become clearer. This approach can be undisciplined, and the transition phase become lengthy, costly, and inefficient. Shortcomings incurred by the indiscriminate use of rapid prototyping can be reduced by recourse to a KBMS development model as a guide to architecture. Such a model aims to gain a broader understanding of the problem domain and the impact of requirements.

Current research has identified, to date, three major perspectives from which to determine required KBMS functionality:[4] knowledge (world view), system engineering (user view), and symbol (machine view) perspective.

We highlight a project manager's point of view, a top-down system's view of overall development. This differs form the view of a KBMS designer (technical manager), who often gets deeply involved with the problem domain in search of ideal architectures and environments. Although using a similar problem domain and field requirements, the designer will be more intimately involved with the information and will be updating his opinions as more details are revealed.

As his chief objective, the project manager wants to partition the knowledge-engineering (KE) development into stages and tasks within phased development levels. Figure 7 shows a matrix of KBMS work levels against stages and tasks.

Knowledge perspective (world view) encompasses a functional breakdown of the system to be built, including requirements and specifications of the desired behavior of the system. If this perspective is incorrect, so will be the resulting KBMS. Two KE levels define the world view. The first entails a complete description of what is to be built and how the system must

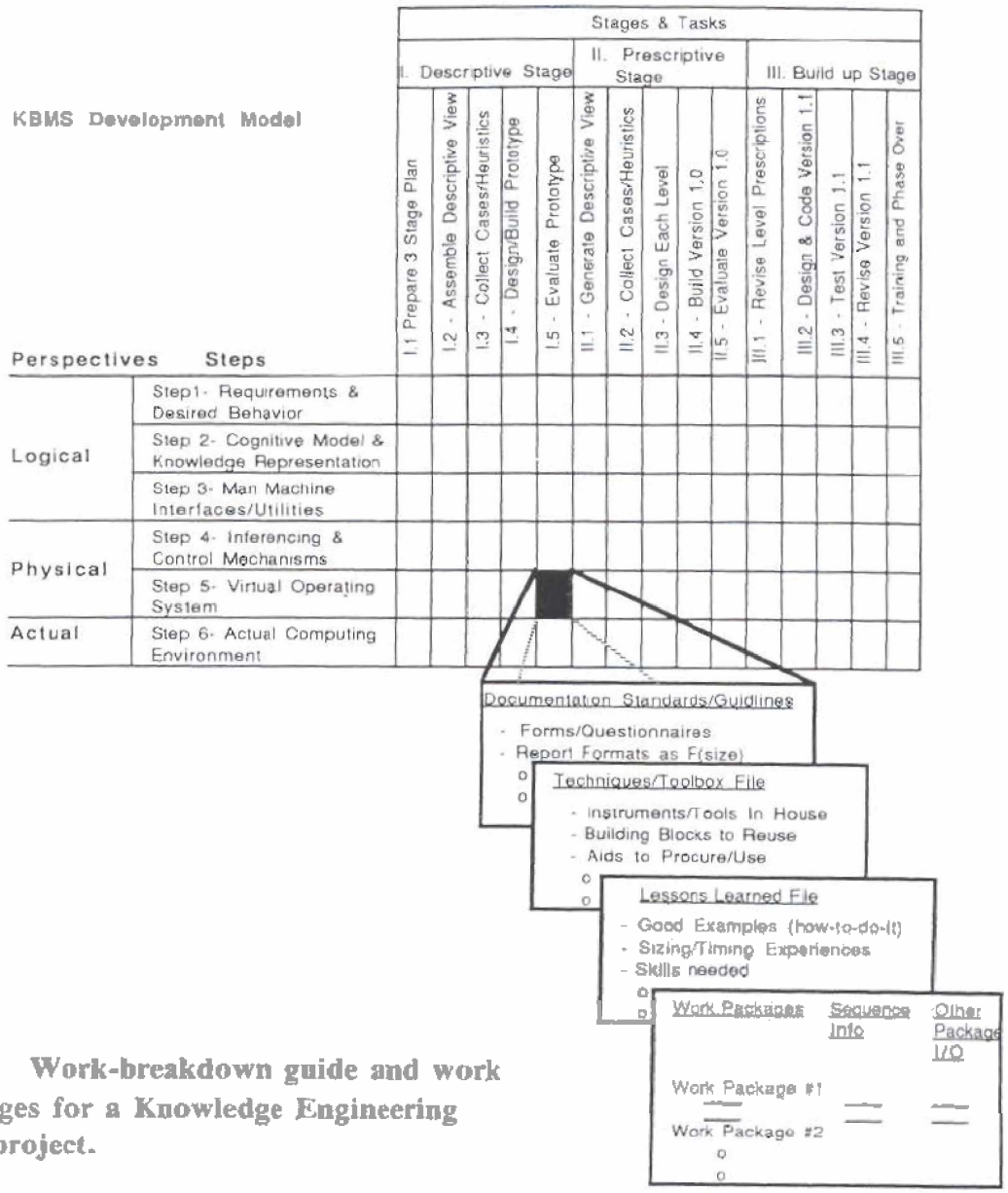

Fig. 7 Work-breakdown guide and work packages for a Knowledge Engineering (KE) project.

function. It explains the purpose and requirements for the KBMS and employs a number of requirements, specifications, and analysis tools. The second KE level describes, for each element or module, the problem-solving protocols between and within elements, reasoning frameworks to be used, and the actual rules, data sets, and KB elements that must be put into the computer. Level two is achieved by use of the full range of knowledge-elicitation techniques (i.e., interviews, thinking aloud, protocol analysis, rule induction, case-analysis decision tree aids, etc.) as well as by structured design of procedural elements.

The system-engineering perspective, which focuses attention on the user, is addressed with a third level of the project manager's model. This level involves the preparation of user and maintainer interfaces that will allow them to customize, install, operate, and update the KBMS. It encompasses

means to explain inferences, run the systems, and display status and results; and skeletal customizable elements of the KBMS.

The symbol perspective (machine view) is addressed by the last three levels of the project manager's model:

The languages, shells, and environments that include such things as mechanisms for encoding definitions of objects, attributes, and values; for parsing rules; for controlling inferencing and reasoning; and for directing search and communication processes. This fourth level also involves concepts for creating new code for novel procedures to integrate parts, or for tools needed at the module or element level.

The fifth level provides the bridge between the KBMS and the hardware environment, which permits integration of the various components into a whole and permits these components to be coded in relative autonomy. This fifth level ideally establishes a "virtual" machine and a distributed operating system that masks the impacts of the actual computers and environments being used, permits different Expert System (ES) shells to be integrated, and provides mechanisms for insuring control and communication across a combination of heterogeneous modules, subsystems, and machines.

The details of installing, connecting, and operating the actual hardware and software environments in which the KBMS will be implemented constitute the sixth level.

Stages and tasks are divided within each level of this KBMS development model into three categories: description, prescription, and buildup—with five subtasks in each category. The descriptive stage sees the domain described. The prescriptive phase sees selections of the KBMS design concept and choice of the usage and maintenance mode. Buildup entails the actual implementation. Detailed tasks involved at each of these stages 5-8 are given in Fig. 7.

Each position in the first two dimensions of the matrix of the six KE steps and the tasks represents a workpackage that may contain several files and documents. Types of files and documents will be considered to make up the third dimension of this matrix. A few of the types of files in the third dimension are work-element lessons learned, techniques and toolboxes needed as functions of the domain variables, and documentation standards and guidelines. These total 90 workpackages giving insight into the KBMS model's scope.

The project manager needs to decide which of the three stages a new project needs. A prototype stops at stage I but a completed KBMS proceeds at least to stage III. He then estimates the steps at which work is needed. For example, a project that uses a microcomputer shell might concern only the top three steps, whereas a custom inference engine might heed all six steps. With these two pieces of information (stages and steps), the project manager can use Fig. 7 to organize all the tasks and workpackages needed.

We have found it useful to group the steps in a manner somewhat different from the perspectives breakout. We use the following partitioning: Logical Model refers to Steps 1 and 2. Physical Model refers to Steps 3, 4, and 5. Actual Model refers to Step 6. In what ensues here we use this terminology.

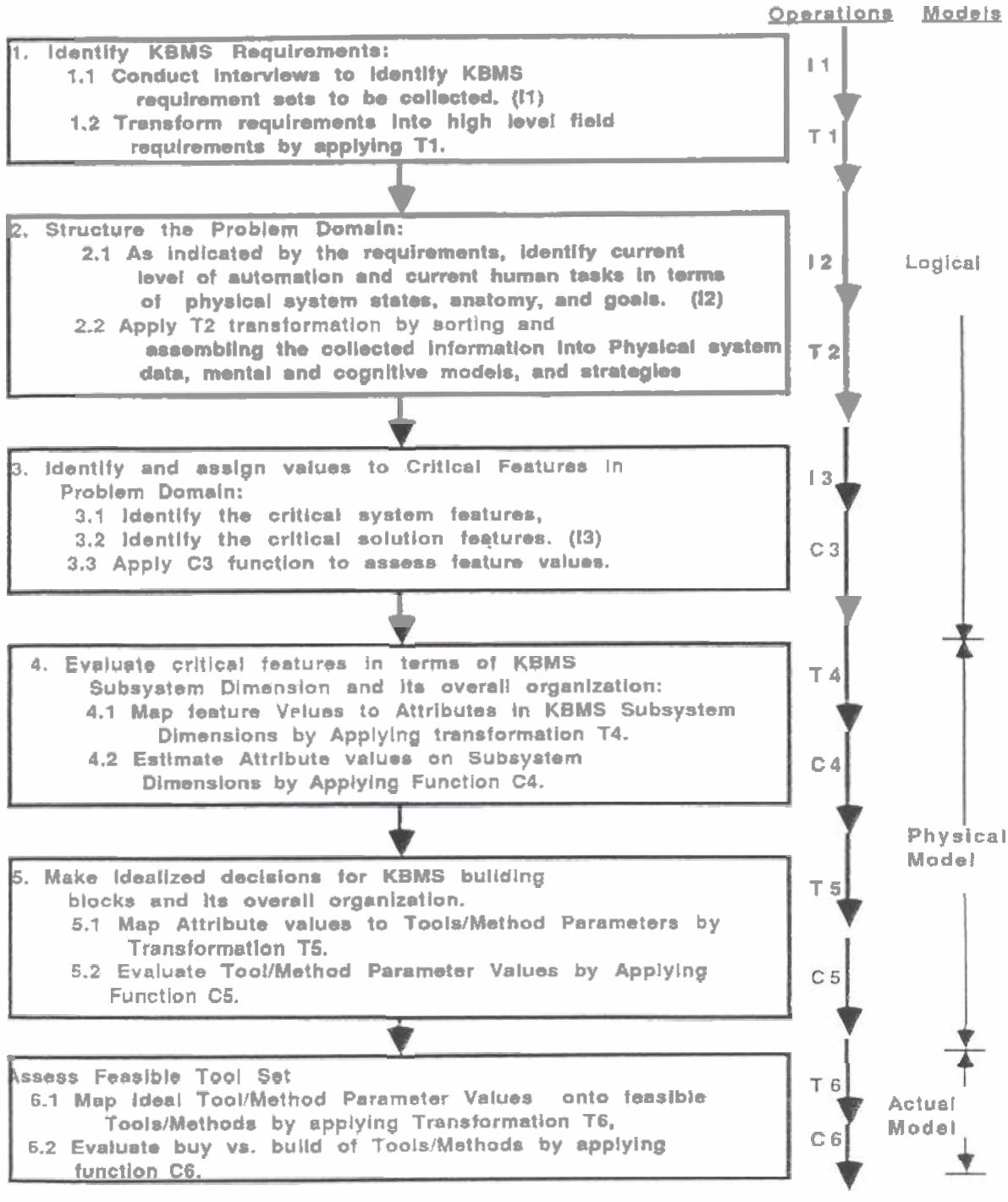

Fig. 8 Summary of KBMS design steps (work packages) for task 1.2 of Fig. 7.

KBMS Descriptive Design Methodology

KB systems with small solution spaces for reliable and fixed data and knowledge can easily be built in either a readily available shell or a custom-designed KBMS. Linear upscaling of a small system's architecture, however, can have disastrous consequences. A KBMS design methodology can help the project manager identify the mix of tools to be bought vs developed. The methodology would be used before prototyping within the descriptive stage of a KBMS development effort (i.e., task 1.2 in Fig. 7).

As previously mentioned, KBMS design commences with development of the logical model during the descriptive stage. Figure 8 summarizes successive design steps or workpackages being applied—from the study of field requirements and problem domain to the prescription of feasible tools to be used and built in the KBMS. This movement from model requirements toward prescription is incremental and is partitioned into KBMS Design Steps (KDSs). Three basic types of operation within the design steps or work packages aid progression from one KDS to the next: I. Identify and assemble pertinent elements from a given KDS; for use in the next KDS, KDS$j+1$.

C. Complete a worksheet (or graph, tree, diagram) on a given KDSj to derive specific values relevant to the features, attributes, or dimensions of that KDSj used in the next KDS, KDS$j+1$.

T. Transform and relate elements in a given KDSj to the pertinent knowledge state in the next KDS, KDS$j+1$.

Design starts with identification of field requirements, which may be obvious or subtle, explicit or implicit. Field requirements (usually imposed by the project sponsors) reflect the sponsors' motiviations for seeking a KBMS solution. Typical motivations for KBMS will be capturing scarce expert information, higher automation, better assurance of system quality, human-error reduction, and improved system autonomy. The first KBMS design workpackage thus includes conducting interviews to assess the motiviation for the KBMS (I1) and transforming the results into high-level field requirements (T1).

In the second workpackage, the designer is expected to examine the human tasks and methodologies associated with the problem domain and current levels of automation. Studies at this level will follow precepts established by Rasmussen.[3] This workpackage begins with an identificiation of system states, anatomy, and goals. This tangible system specific information labeled KDSs—may be readily available from the literature, experts, or observations of the system behavior.

System states, anatomy, and goals are then analyzed and transformed into data, mental and cognitive models, and strategies. T2 in the second workpackage denotes the process of converting tangible system elements to their mental representations. The many possible techniques for T2 range from Human Information Processing models to qualitative reasoning about physical systems.

In the third workpackage, the previous studies (KDSs) are converted into a statement of the expected system and solution features. The system features are items external to the KBMS but are expected to impact on the design of its intefacing subsystems (e.g., Database, Procedures, End-user Interface, etc.) Examples include input/output characteristics, limitations on the hardware and software operating enviornments, system anatomy, and standards guidelines. The system features are identified using a worksheet.

Solution features reside in the domain and tend to influence design and content of internal KBMS subsystems (e.g., Learning/Inferencing, Hypothesis Contention Space, and Knowledge Base). The solution-feature categories are data/knowledge, characteristics of the solution space, human

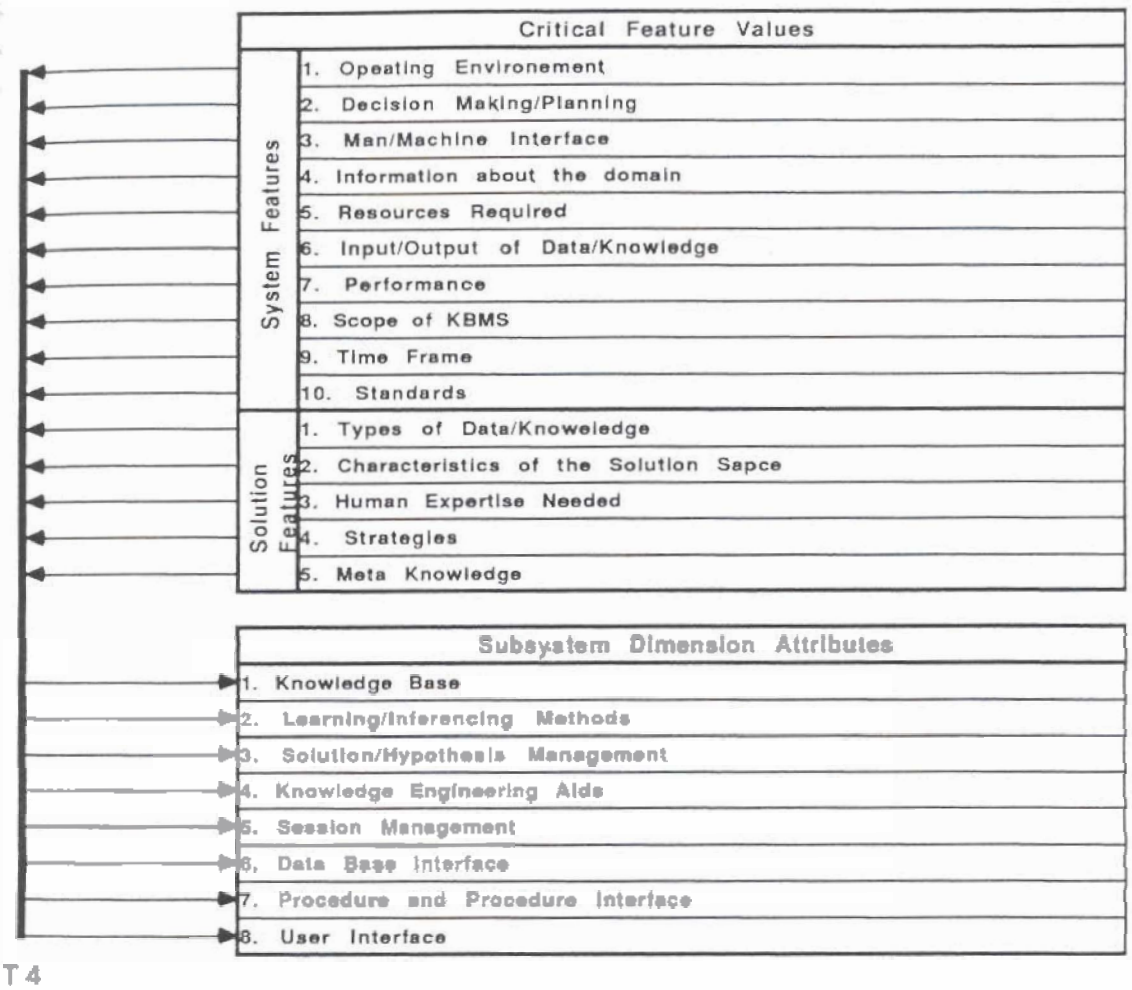

Fig. 9 Transformation T4: an example of the features involved in the mapping process.

expertise in the system, strategies in hypothesis contention space, and forms of meta-knowledge. Again, a worksheet is used to complete this substep. Subsequent analysis of feature values gives insight into operations (C3).

In the fourth workpackage, the critical features are first organized and mapped onto the idealized KBMS dimensions for each subsystem and on the overall organization of a KBMS. As depicted in Fig. 9, this transformation is referred to as T4. Next, the detailed dimensions and attributes of each subsystem are fleshed out using a checklist of subsystem dimensions and attributes.

KB attributes help determine the structure of the knowledge base, how to partition the KB into knowledge sources, and how best to represent knowledge so that it can be used in problem solving. Attributes in the learning/inferencing methods can help determine the level of sophistication needed to traverse the solution space. At a higher level of control, solution/hypothesis management attributes dictate how to steer the solution toward acceptable goals. Database and prodecure interfaces determine the versatility for tapping into other systems for data and canned procedures. Finally, the user-interface attributes help determine the end-user communication pro-

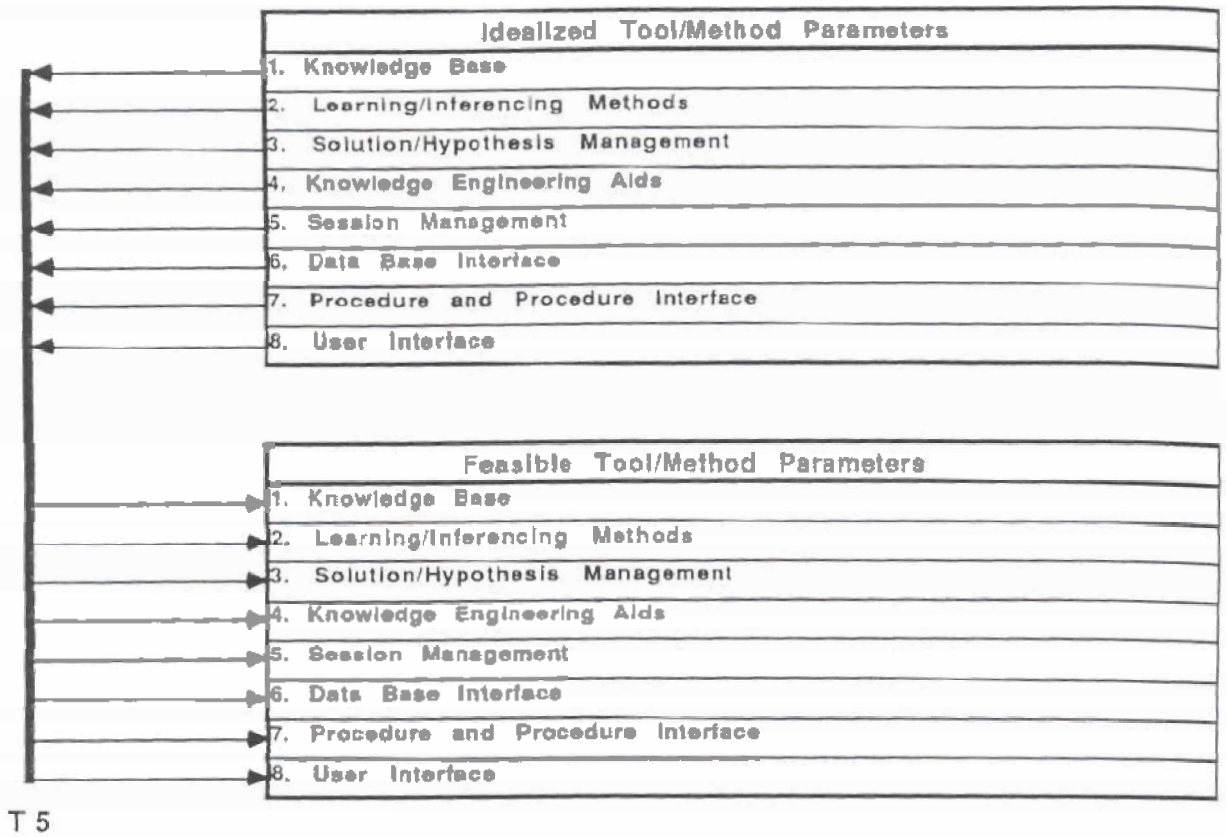

Fig. 10 Transformation T5.

tocols. Following the T4 transformation, C4 is applied to derive detailed descriptions of the attributes and their implications for tools. This step constitutes a high-level design of the KMBS subsystems.

In the fifth workpackage, idealized attribute values are mapped onto tool and method parameters, as depicted in Fig. 10. Many of the necessary parameters are widely documented in the literature, and many combinations of tools and methods are on the market. One of the first considerations in tool selection, knowledge representation, determines means by which domain knowledge will be used. Similarly, learning/inferencing techniques are highly visible selectors.

Solution/hypothesis management contains the paramenters for bookkeeping and guidance. KE aids help determine the tools for constructing a knowledge base and for fine-tuning the solution. Session-management parameters tune system "composure."

Transformation T5 requires a compromise to satisfy both a close match between the idealized features and the available tool/method parameters and constraints, such as the designer's stylistic biases and preferences for tools and methods.

In the sixth workpackage, the idealized tool and method parameters are mapped onto the commercially available tools and methods by transformation T6 (see Fig. 11). Potentially these tools and methods can be part of the operating environment. If so, the KBMS design methodology might have to predict and backtrace the design steps to ascertain the feasibility.

b. Many sources describe commerically available tools and methods.[9-11] Three major categories are easily identified: languages, shells, and skeletal systems. These are commonly used to build artificial intelligence (AI) applications, since they allow features such as list processing, logic programming, and object-oriented programming. Shells offer commonly used techniques in inferencing and knowledge engineering to reduce development time. Skeletal systems are tools for automating ES by providing a domain-independent system. These skeletal systems can also be used as part of building a KBMS system.

Decisions about building vs buying will be made as a last step in the sixth workpackage, C6. They will mainly be based on cost and fit of the tools.

Application and Assessment of KBMS Design Model

The model presented in the previous section can be illustrated and partially tested in terms of a case study on a large-scale ES prototype.[5-8] The study represents an implementation of the first third of the Work Breakdown Structure (Fig. 7) that forms the underpinnings of the chapter; i.e., it straddles tasks 1.1 through 1.5.

The Network Control Center (NCC) forms the case domain. The goal was to study NCC as illustrative of other facilities at GSFC and to evolve a generic shell initially suitable at an advisory level for operations and later to be evolved into an autonomous facility; that is, to build a "Facility Advisor" shell, consisting of a number of validated, skeletal elements, that can be tailored to the individual supervisory positions at any facility, that can support distributed problem-solving, and that can be succesfully extended (or integrated with any previously existing stand-alone supervisory ES) to support stand-alone nondistributed supervisory control at that position.

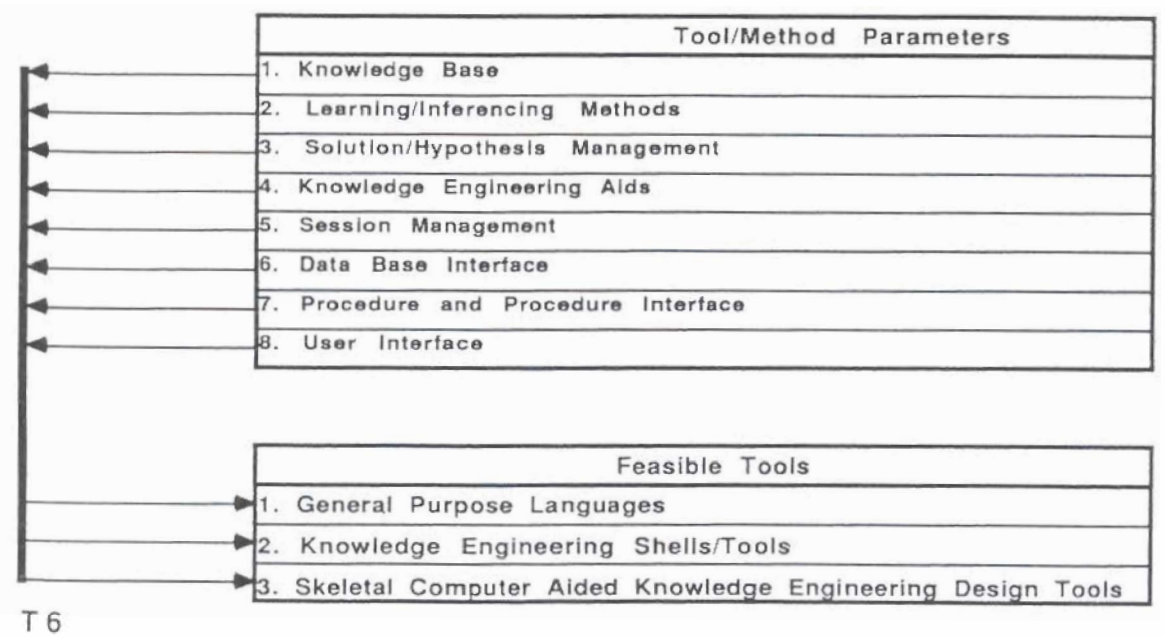

Fig. 11 Transformation T6.

The relative newness of Distributed Expert System (DES) technology and the "no-risk" requirement of facility operations highlighted a need for a DES testbed. Whether or not all goals of the Facility Advisor project could be met, the testbed was expected to provide at least a DES training ground, lessons learned, how-to-do-it manuals, and insights into integrating DESs into facility operations and planning activities.[8]

Identifying KBMS Requirements

Requirements definition, a traditional step in system or software engineering methodology, relatively "loosely" circumscribes work on a KBMS package, leaving the degree of specificity of requirements up to the model user. It simply insists on two steps: 1) identify requirements, I, and 2) transform them, T, into field requirements. The principal requirements delineated for the Facility Advisor readily take this format (Table 2)[7]:

1. *Supervisory Controller Position (SCP) Expert Systems*

Many of the ground-operations personnel are employed at automated workstations. The person plus the workstation comprise three levels of intelligence: the human supervisor and two computerized lower levels of intelligence. The Facility Advisor is comprised of several generic positions referred to as Supervisory Controller Positions (SCPs). (Other terms might do as well—scheduler, operator, analyst, command-controller, etc.) In terms of field requirements, this implies expert systems built for facilities—stand-alone ESs or part of a DES—must be carefully integrated into the SCP tripartite to be useful.

2. *Cooperative, Interdependent Behavior*

While most SCPs have relatively well-defined areas of responsibility, in order to do their job they must interface and cooperate with other SCPs. From a field view, this requirement can be explored by recognizing that, within a given facility, there are generally at least three (and often many more) interacting positions: 1) the scheduler who decides when equipment and other resources may be allocated to support each user, 2) the operator who utilitizes the resources to perform a user-requested service and who detects and corrects quality problems of the end product (e.g., message code errors, data-set noise, etc.), obtaining inputs from the equipment monitor to assist in problem-isolation tasks, and 3) an equipment monitor who detects and isolates equipment and resource problems and either corrects them in time for a given service to be completed or suggests an alternative equipment pathway for the scheduler's consideration. In addition, each of these three SCPs individually must interact and cooperate with their counterparts at other control centers and facilities to solve problems and to do their jobs.

3. *Adaptive, Flexible Reasoning*

The users of facilities desire and continuously make unusual requests for services. These might include a large degree of flexibility in event scheduling, dynamically varying spacecraft-instrument commanding and

Table 2 Handling of cognitive functions at each local SCP by three levels of intelligence

LEVELS OF INTELLIGENCE	COGNITIVE FUNCTIONS							
	Communicating & Negotiating	Blackboard Writing/Reading (internal)	Chair/Task Scheduling	Fision/ Situation Recognition	Problem Solving & Planning	Controlling	Accounting & Logging	Meta-knowledge & Learning
Task Level Computer	o Network listener o Front end processor o Code up transmissions/ requests o Decode resposnses	o General display (infor alerts, action, etc.) o Read tasks	o Service tasks in priority order subject to ongoing constraint	o Automatic sensing o Automatic anomaly and status detection and reporting	o Automatic deduction (check meta-knowledge & past solutions to control or problem to human	o Semi automatic servo-control	o Automatic storage/ retrieval in long term memory	o Execute meta-knowledge about whom to call on for anomalies
Human Inter-active Computer	o Display organizer (translate machine to menu parameters) o Translate human to machine language	o Hold/display info & organize o Translate human input	o FIFO o Run scheduling aids for human	o Organize anomaly & status info hierarchically	o Place problem alerts at the top of information hierarchy	NA	o Exercise retrieval queries/ storage requests	NA
Human Super-visory Operator	o Negotiate o Voice: -in room -over phone -keyboard	o Read display Write tasks o Make mental notes	o Reassign task priorities at will	o React to anomaly alerts and multiple inputs o Remain congnizant of system status	o Full range of plausible deductive reasoning	o Implement solutions & exercise interruptability adjust, take-over	o Summerize, account & log	o Learn system strengths & weaknesses o Reprogram on line as needed

NA- Not applicable

controlling, and alternative telemetry formats, frequencies, and durations. In short, SCPs must exhibit a great deal of adaptive behavior within their "well-defined" responsibility. The Space Station, as one example, is intended to be able to handle unforseen activities, targets of opportunity, synergistic experimental operations, adaptive instrument behavior, etc. and to allow interactive use of the overall system in a flexible, transparent manner ("telescience"). For SCP expert systems this implies two radically divergent modes of reasoning. At some times SCPs must be capable of elaborate planning and adaptive behavior; at other times, they must be prepared simply to execute plans with great speed.

Structuring the Problem Domain

The Facility Advisor project followed model Steps 2.1 and 2.2 precisely. First, the NCC SCPs were studied in terms of states, anatomy, and goals. A task analysis was performed and, in the end, the SCP was seen to consist of eight generic functions performed by each of the three levels of SCP intelligence (one human and two computer), as summarized in Table 2. The Supervisory Control Nodes tend to leave the most intellectually arduous functions (e.g., anomaly troubleshooting of the Real-Time Control function, the stochastic, plausible reasoning of the Problem Solver and Planner function, and the learning side of the Meta-Knowledge and Learning function) to the human supervisory operator.

Table 2 reflects the result of applying the transformation computation (T2) that puts the domian into more of an AI perspective. With today's AI and ES techniques, the Facility Advisor aims to replace many functions of the human at the third and highest level of intelligence.

Estimating Critical-Feature Values

Transformation computation (T2) of the case "problem anatomy" into cognitive protocols and strategies required a large number of behavioral protocol studies; e.g., how each function is performed by a "typical" SCP, how multiple SCPs in one facility cooperate to perform these functions, and how SCPs interact across related facilities.

The result of this knowledge collection was the emergence of a block diagram representation of the "logical model," as depicted in Fig. 12. The designers could see at this point that—

—The ES includes three representative Specialists plus a manager.

—Each Specialist includes nine generic elements that need be built only once and that handle all the cooperation and coordination needs.

—Each Specialist includes a set of "modules" (unnumbered boxes) that are unique to its operation.

There are thus certain generic elements to be added to the Facility Advisor and for any facility to which the Facility Advisor might ultimately be applied. Further, the Specialist-unique module sets contain elements which can and should ideally be standardized, so that any facility that requires a Specialist of that type (as part of its Facility Advisor or as a stand-alone entity) will receive a validated structure and certain validated KB elements. The Specialist can thus be tailored to a given facility primarily by extending and tailoring its rule base.

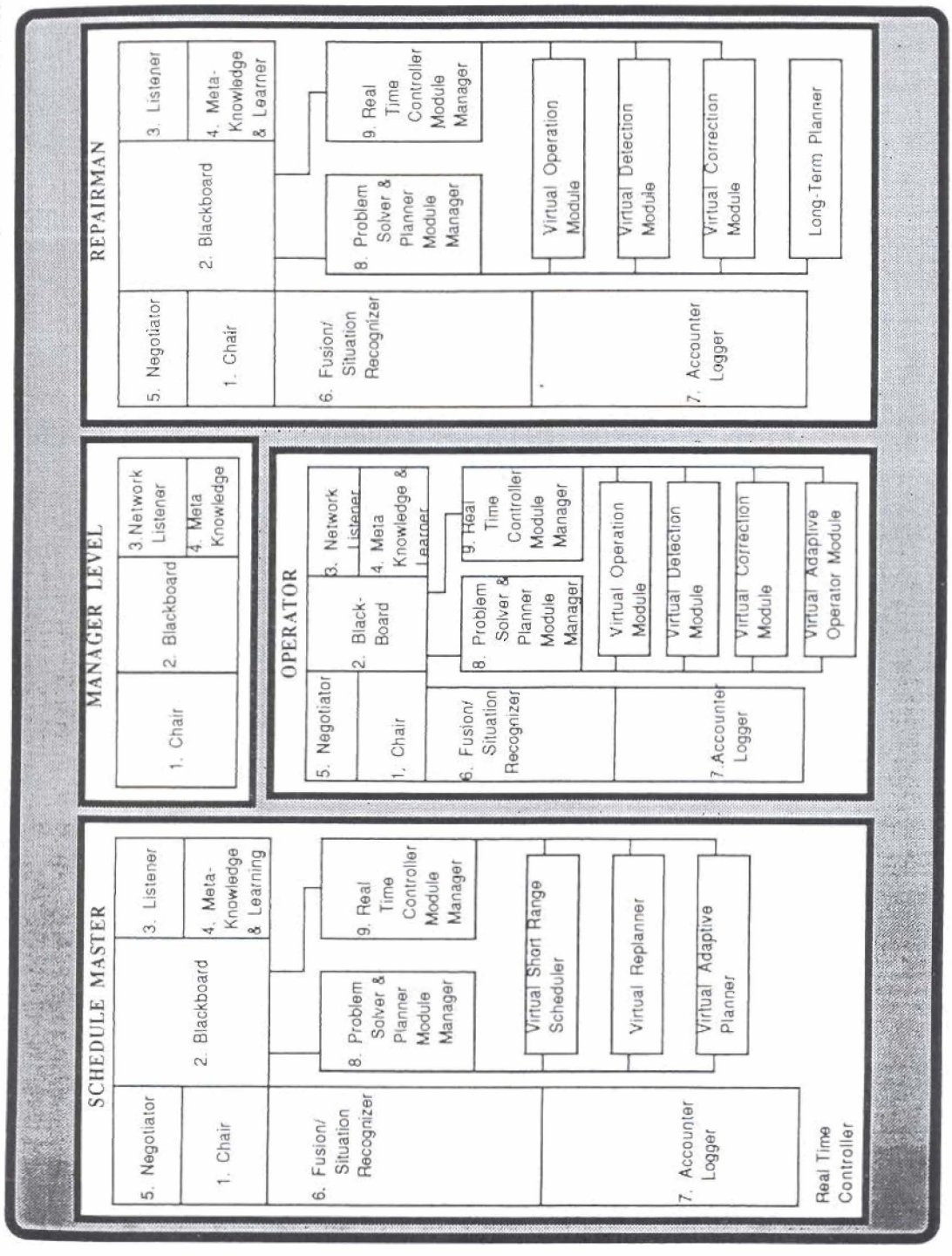

Fig. 12 Nine generic elements and three module sets to be developed for the Facility Advisor.

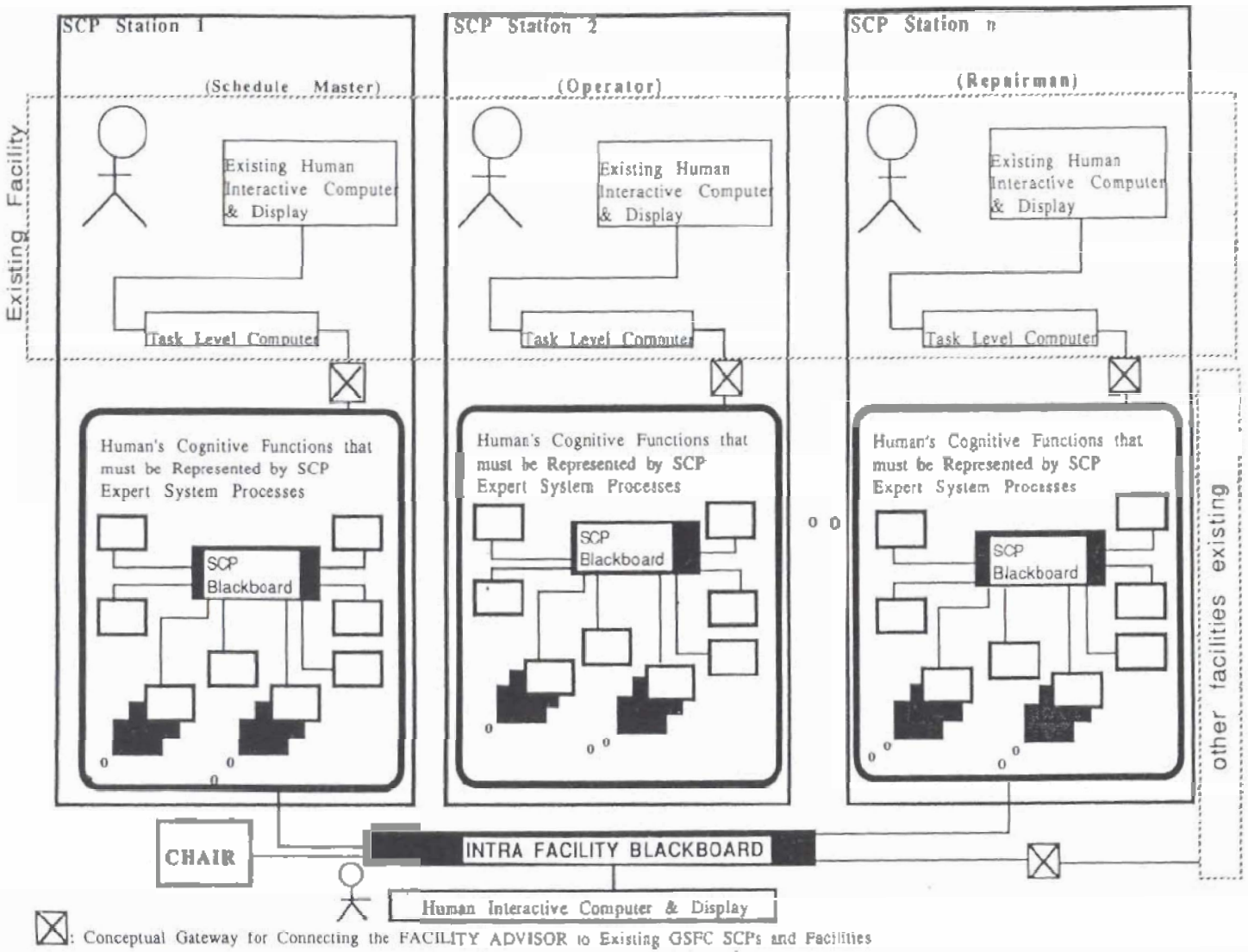

Fig. 13 Facility Advisor logical model and how it fits into exisitng SCPs and facilities.

KBMS Subsystem Dimensions

The fourth workpackage requires a top-down double check, in which the features and values collected thus far are organized and studied in terms of their impact upon each of the eight KBMS subsystems (Fig. 3). Facility Advisor work emphasizes the following subsystems of the KBMS model: solution/hypothesis management, learning/inferencing methods, and knowledge-base interface. Session management and knowledge engineering subsystems will not be discussed further; but the database, the existing procedure interface, and the user-interface subsystems are described below.

Regarding the database and existing procedure-interface subsystems, Fig. 13 shows how the DES mode will connect via conceptual gateways to existing SCPs both within the test site and at other facilities. The conceptual gateway is intended to show that the logical model will be developed using actual GSFC protocols, situations, events, etc.[7]

The interface layout of Fig. 5 will allow the testbed gradually to phase into a real facility without interrupting the performance of the people already working at the SCP consoles. The Facility Advisor will run on separate dedicated computers, and electronically tap the task-level computers, existing software, and databases to receive a duplicate image of the command-telemetry alert messages seen by the operators. When Facility Advisor approaches "human-quality" performance, the operators at the SCPs will begin to use it in an advisory manner.

Regarding the user-interface subsystem, a technique found useful in the design process is to diagram out each screen, window, menu, and all the dialog sequences possible under each. Given a complete overview of the Facility Advisor Logical Mode, it is possible to complete workpackage 4 by filling out a checklist applying step C3.

Idealized Decisions for KBMS

Workpackage 4 involves a long-range top-down view of what should ultimately be built. Efforts on workpackage 5, on the other hand, attempt to pinpoint precise techniques needed in the long run as well as what should be attempted in the prototype. Mapping can be done from the high-level subsystem checklist to a more detailed one, as was suggested by Fig. 4.[7] Mapping based on the Facility Advisor case proceeds as follows:[7]

The KBMS must support a wide range of control schemes (slavelike to fully distributed), several levels of communication interfaces (intermodule, interprocess, intrafacility, interfacility), multiple reasoning schemes (situational calculus, belief-based fusion, and several planning calculi), and synchronization among numerous semiautonomous components, not all of which use AI techniques. Since the Facility Advisor is ultimately targeted for transfer to a variety of facilities, the knowledge-engineering effort must also provide a "virtual machine" that can be mapped into a wide range of underlying hardware/OS environments.

The fundamental problem common to both the interfacility situation and the across-user service request concerns managing processor resources to

responding to a given user-service request. For several reasons it is necessary to be able dynamically to create and manage SCP cognitive processes, modules, or module elements (subprocesses or intelligent assistants). Each of these must be a "virtual entity," able to "clone" a task in the same fashion as the busy entity could have. A virtual entity occupies no memory when it is not needed.

Assessing a Feasible Tool Set

Before prototype development is begun, the final design workpackage requires a reexamination of expectations based on real-world constraints, such as available budget, time to develop, and matching needs with affordable tools.

The Facility Advisor needs a Distributed Expert System (DES) and a testbed that supports cooperating yet more-or-less-autonomous reasoning components. How these components would be built has been discussed in Facility Advisor reports as follows:

> The discussion thus far...suggests a substantial investment of effort for defining the generic elements and precisely how they would work. In addition, effort is needed for coding the validation testing of these generic elements so that they can be offered customizable. The authors have had a chance to design, prototype, and test most of these elements by virtue of several DES and ES jobs. In particular, off-line blackboard and chair have been developed...while the real-time blackboard, chair, and module manager exist in a software environment called HCSE...Schedule Master planner and replanner techniques will be recycled from our...Space Station Customer Scheduling Expert System currently being developed for McDonnell Douglas Astronautics Co. (MDAC). All remaining generic elements exist at the detailed design level, as summarized in earlier reports.[5,8]
>
> The Facility Advisor is intended to insure communications among diverse modules and elements, cooperation between operating systems of different machines on which the parts of the Facility Advisor will reside, and transportability for application purposes. To leverage the effort, it would be fruitful to utilize techniques being developed elsewhere, to the extent possible, such as in DOD's Cooperation Between Operating Systems Package (COP) project at George Washington University. An application (such as Facility Advisor, specialists, blackboards, processes, modules, etc.) can call upon COP to establish logical networks to other applications and to select ways to send mail (e.g., point-to-point, broadcast, etc.)[7]
>
> Because a number of stand-alone expert systems are already under construction at NASA-Goddard, Facility Advisor must provide an open environment that maximizes the ease with which third-party expert systems can be incorporated into it. The Advisor must support reuse of previously constructed components, integration of diverse components and modules, and interfaces to their favorite expert-system development shells (e.g., ART, KEE, or IntelliShell)...
>
> It is difficult to identify an optimal computing environment (sixth KE level) for building the Facility Advisor. Goddard-DSTL has several

Symbolics Lisp machines, two VAX's, at least one Sun, and a high-resolution color-graphics workstation. Other Goddard facilities have yet other computers...

Finally, the range of fifth-generation computers and distributed operating systems that will be available at the end of the contract period is potentially unpredictable...Due to all these considerations, hardware and software requirements for this contract can almost entirely be stipulated in terms of programming-team productivity and motivation.[8]

Conclusions, Future Directions

A single case can hardly suffice to confirm the methods just described as alid. Much remains to be done on the model. Future case results and efinements to the model will be published periodically.

But the case presented here offers a path and insights. To begin with, it s possible to assess the exhaustiveness of the KBMS design methodology or pinpointing all the KBMS subsystems to be built and for guiding lesigners through an appropriate sequence of workpackages. In addition, he case should give feedback on what was done in an actual large-scale prototype design effort and, hence, what may have been omitted from the nethodology. All steps taken in the case study did seem required by the workpackages.

In terms of weaknesses of the KBMS design methodology, the case used a much richer set of representations (e.g., graphs, trees, block diagrams, and textual analysis) than could be included in the simple workpackage write-ups. This is perhaps most acutely evident in the requirements-definition step and the inability of trial checklists to capture all the tools needed. Designers must thus embellish the method presented here with material relevant to their domain.

In summary, the KBMS design methodology seems to provide a good initial framework for focusing on architectural needs. With refinement it promises to support the development of reliable and robust management systems required for highly distributed environments in the future, such as those slated for the Space Station era.

Acknowledgment

The authors wish to thank John Dalton, head of the NASA-Goddard Data Systems Technology Div., and Dolly Perkins, head of its Data Systems Application Br., for their encouragement and support.

References

[1]Blanchard, D. L., "Smart Sensors from a NEEDS Perspective," AIAA/NAA Conference on Smart Sensors, Hampton, VA, 1978.

[2]Computer Technology Associates, Inc., "Data-Base System Architecture Study," NASA contract GSFC NAS5-29893, Feb. 1983.

[3]Rasmussen, J., *On Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering,* Elsevier, New York, 1984.

[4]Brodie, M. L. and Mylopoulos, J., (Eds.), *On Knowledge Base Management Systems,* Springer-Verlag, New York, 1986.

[5]Silverman, B. G., "Distributed Expert Systems," NAS5-28604, Code 522.1 GSFC, 1985.

[6]Silverman, B. G., "Distributed Inference and Fusion Algorithms for Real-Time Supervisory Controller Positions," *IEEE Transactions on SMC,* Institute of Electrical and Electronics Engineers, New York, March 1987.

[7]Silverman, B. G., "The Facility Advisor: A Distributed Expert System for Spacecraft Control Centers," *IEEE-CS Expert Systems in Government Symposium Proceedings,* Institute of Electrical and Electronics Engineers, New York, Oct. 1986.

[8]"The Facility Advisor: A Distributed Expert System Testbed: Functional Requirements Document and Plan," NASA contract GSFC NAS5-28604, June 1986.

[9]Gevarter, W. B., "The Nature and Evaluation of Commercial Expert System Building Tools," *IEEE Computer,* May 1987.

[10]Hayes-Roth, F. and Waterman, D. A., (Eds.), *Building Expert Systems,* Addison-Wesley, Reading, MA, 1983.

[11]Waterman, D., *A Guide to Expert Systems,* Addison-Wesley, Reading, MA, 1986.