Conceptual Model-Based Reasoning for Knowledge-Based Software Project Management

Kent D. Bimson
Lockheed Software Technology Center 2100 East St. Elmo, Austin, Texas 78744 512/448-9719

Linda B. Burris
Artificial Intelligence Center 21 00 East St. Elmo, Austin, Texas 78744 512/448-9712

1988. In *Proceedings of the twenty-first annual Hawaii International Conference on System Sciences.* Vol. 3, *Decision support and systems track,* 255–265. Washington: Computer Society Press of the IEEE.

Conceptual Model-Based Reasoning for Knowledge-Based Software Project Management

Kent D. Bimson
Lockheed Software Technology Center
2100 East St. Elmo, Austin, Texas 78744
512/448-9719

Linda Boehm Burris
Artificial Intelligence Center
2100 East St. Elmo, Austin, Texas 78744
512/448-9712

Abstract: This paper presents a conceptual model for software project management and the power derived from using a conceptual model-based reasoning approach in building intelligent decision–support systems. The Software Project Manager (SPM) has been prototyped in Inference Corporation's Automated Reasoning Tool (ART) on Symbolics artificial intelligence (AI) workstations. This prototype conceptual model is an outgrowth of research conducted under the Knowledge-Based Software Project Management project at Lockheed Software Technology Center in Austin, Texas. In this paper, we present an overview of the management model underlying SPM and define the essential concepts and relationships needed to model the project management domain. We then describe the knowledge representation strategy used to implement this conceptual model. Finally, we illustrate the power of using conceptual model-based reasoning in building intelligent decision-support systems for the project management domain.

THE SOFTWARE PROJECT MANAGEMENT CONCEPTUAL MODEL

In a very real sense, SPM's model is generic: it does not reflect any particular methodology or standard, such as the Waterfall method or its descendent, Department of Defense Standard 2167 (DOD-STD-2167, or simply 2167). Rather, it allows such methodologies to be described in terms of project management domain primitives defined in the knowledge base. We consider most current software project management methodologies to be built from the same underlying primitive concepts, relations, and constraints. Among other things, the conceptual primitives include projects, organizations, schedules, phases, tasks, inputs, outputs, resources (people, hardware, etc.), deliverables, milestones, and budgets. The relational primitives include such notions as precedes, which describes the relationship between two tasks or projects; produces, which describes the relationship between a task and its output; or depends-on, which describes the dependency relationship between a task's inputs and outputs. Constraints typically involve negative impacts on project plans, including reductions in planned schedule, budget, or required resources. These concepts, relations, and constraints are the conceptual realities with which project managers work on a day-to-day basis. Methodologies are the means by which those primitives are arranged to plan projects, solve problems, status tasks, or make projections.

We therefore define strategies like the Waterfall model—and standards such as 2167—as structural (or syntactic) overlays that arrange conceptual project management primitives in specific ways for specific purposes. DOD-STD-2167 requires a particular set of milestones, such as the Preliminary Design Review (PDR), that have a specificed set of deliverables associated with each milestone, such as the Software User's Manual. Each deliverable likewise has a prescribed set of contents arranged in a specific sequence. Typically, each methodology emphasizes different attributes of the knowledge base. For example, Data Flow Diagrams emphasize the flow of data between processes, the Performance Evaluation and Review Technique emphasizes precedence relations between tasks, and the Hierarchy–Input–Process–Output methodology emphsizes task hierarchies, inputs, process descriptions, outputs, and so forth. The project viewpoint taken by a particular methodology necessarily emphasizes a few important characteristics and deemphasizes others, since each methodology has been designed to provide visibility into a particular set of problems. The graphical charting or diagramming techniques associated with each methodology are simply surface representations of the methodology's underlying viewpoint—a display of particular knowledge base attributes for a particular management purpose. In principle, these graphical viewpoints are all generable from the same underlying knowledge base by reasoning differently about the various task, project, and resource attributes and relations.

The following paragraphs define some of the more important concepts used to build SPM's conceptual model, a model designed to be robust enough to support these various methodological viewpoints. The definition of concepts—and the network of relationships that hold between them—has been kept as simple and generic as possible. These definitions will grow as key attributes and relations become important to the reasoning in SPM.

Concept Definitions and Relationships

In this section we define some of the more important concepts in SPM's model, including the terms project, task, input, output, resource (especially money, time, and people), milestone, and deliverable. Because of space limitations, not all definitions are included here.

Project. A project is defined as a set of tasks arranged in time (a schedule) to accomplish a specific objective, which typically is defined by a set of requirements. It also includes a set of resources (people, money, facilities, etc.) required to accomplish those tasks.

Task. A task is a sequence of steps needed to take an input *i* and change it into an output *o*, which fulfills, directly or indirectly, all or some part of a project requirement. The write-Software-User's-Manual-scope-section task in figure 1a, for example, requires the overview-tool-description document as input and produces the scope section as output, which in turn meets the requirement specified in section 10.4 of data item description DI-ECRS-8X21 (fig. 1j). In order to perform this input-to-output transformation, a task requires the use of resources (in our example: Max, a PC, and a word processor),

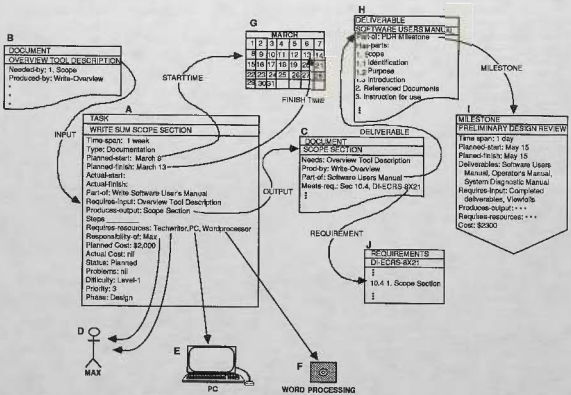

Fig. 1. Partial illustration of SPM's conceptual model

the expenditure of time (one week), and the expenditure of money ($2,000). Some other self-explanatory task attributes include a set of scheduling attributes such as earliest and latest start dates, earliest and latest finish dates, actual start and finish dates, and so on; a set of attributes specifying task characteristics, such as task description, difficulty level, problems, priority level, phase (design, test, development, etc.), and task type (coding, documentation, briefing, etc.); and an attribute describing task status (planned, in progress, or completed).

Input and output. An input is anything that is transformed by the steps in the task; an output is what results from this transformation. In figure1, the overview-tool-description (fig. 1b) is transformed into the scope-section (fig.1c) by the task in question. This transformation is figurative in many cases, of course, since the overview-tool-description is not changed physically. Input and output are functionally defined concepts. An object is an input to a task only if that task transforms it into an output. For this reason, inputs and outputs have no life of their own; they have no lexical definition in the knowledge base. The scope-section is not generically an "output object" but is an output object with respect to this particular writing task because of its functional relationship to the task.

Resource. A resource is anything that is required by the task to facilitate the process of transforming input into required output, but which is not itself subject to the transformation. People, software, hardware, facilities, materials, and data bases are particular kinds of resources required to perform different transformations. In order to write the scope-section, for example, we would require a technical writer, a personal computer, and a word processor (fig. 1d–1f), yet none of these is transformed by the performance of the task.

Each type of resource has particular characteristics and semantic relations to other domain concepts and objects. For example, people write code; code does not write people. Code runs on computers, computers are owned by organizations, organizations are made up of people, and so on. However, because the semantics can become arbitrarily complex, we have chosen to limit the semantics of software project management to what is useful in building an intelligent project management system. Because of space limitations, we do not define all of our resources in this paper, but a few require some comment, including money, time, and people.

Money. In one sense, all other resources can be translated into monetary terms: people's time, computers, and facilities. These costs are either time-dependent or time-independent. The planned costs for people on a task are a function of time: time units x pay rate per unit x overhead factor. The purchase of materials such as paper or acquired software is not a factor of time—the purchase price of a word processor remains the same whether the task takes two days or three months. However, even for these latter resources there may be hidden time-dependent costs not related to their purchase, such as maintenance, depreciation, or software support.

Time. Two kinds of time must be included in the model: *linear time* and *calendar time*. Linear time forms the basis for temporal comparisons such as during, while, before, after, until, and so forth (Allen 1984). Calendar time in some sense overlaps linear time and forms the basis by which managers reason about management problems such as schedules. It is important to the project schedule that this task begin on March 8 and finish on March 13, not just that it will take one week to accomplish (fig. 1g).

People. People are unique resources in that they are responsible for controlling and coordinating the use of all other resources and for accomplishing project tasks. In a software environment, therefore, required attributes include task assignments, grade level, salary, specialties, experience (languages, systems, industry, previous tasks, education level), preferences, organization, manager, and so forth (Bimson and Burris 1986).

Milestone. A milestone is a scheduled event that serves to give management or the customer control over and insight into a project. A milestone has a set of deliverables or products and a set of tasks associated with producing those deliverables. In DOD-STD-2167, the Preliminary Design Review milestone requires the contractor to present the Software Top-Level Design Document and the Software Test Plan for each computer software configuration item. It also requires the contractor to present preliminary versions of the Computer System Operator's Manual, the Software User's Manual, summary reports of internal reviews, and several other documents. The presentation of these materials requires preparation of briefings, overhead viewfoils, and so forth. Therefore, the preparation and presentation of a milestone review usually is considered a task in itself, with its own requirements

for input, output, resources, budget, and time.

Deliverable. A deliverable is any product required by the customer, such as a document, software, or hardware. Typically, deliverables are composite objects that can be broken down into constituent parts or sections, each of which can be broken down further into smaller constituent parts, and so forth. In the 2167 example, a Software User's Manual can be decomposed into several major sections, including (1) Scope, (2) Referenced Documents, and (3) Instructions for Use. The Scope section breaks down into (1.1) Identification, (1.2) Purpose, and (1.3) Introduction (fig. 1h). All of these sections and subsections are the products of low-level tasks and form a composite deliverable (the Software User's Manual) required in the PDR.

Summary. These and other concept definitions form the basic model from which SPM has been built, as well as the relationships described, serve to connect concepts to one another in a complex network, as illustrated in figure 1. Once a conceptual model is properly in place, small, distinct, mission-specific rule bases can be developed to perform management activities semiautonomously. In fact, a robust conceptual model reduces the work of the knowledge base's deductive component, making rule bases easier to modularize and easier to write.

KNOWLEDGE REPRESENTATION IN SPM

The conceptual model discussed above has formed the basis for representing the software project management knowledge in SPM, which has been prototyped in ART on Symbolics AI workstations. In this section, we summarize the representational structure that was used in building SPM, and which is discussed more fully in Bimson and Burris 1986.

SPM's knowledge representation uses semantic networks to build up the declarative project management model and uses rules, functions, and procedures to reason deductively over the model. Semantic networks are discussed in detail throughout the AI literature (Quillian 1966; Minsky 1975; Woods 1975; Brachman 1977, 1979; Fox 1983; Brachman and Schmolze 1985; Sathi et al. 1985, 1986; Bimson and Burris 1986). In summary, semantic nets in SPM are made up of nodes that represent concepts and arrows that represent relationships between concepts. In ART, the concepts are implemented as frame-based structures called schemata; relations are represented by slot names within the schemata. Values within these slots provide the project data for individual instantiations of the model. These are the basics used to build SPM's conceptual model.

Brachman (1979), Fox (1983), and Brachman and Schmolze (1985) demonstrate the importance of representing knowledge in multiple, stratified layers in order to structure knowledge from primitive concepts and relations to more complex, domain-specific concepts and relations. Brachman and Fox define five layers of knowledge—from the most primitive and generic to the most domain-specific—as the implementation layer, the logical layer, the epistemological layer, the semantic (or conceptual) layer, and the domain layer. This representation has been altered in SPM in two ways since Bimson and Burris 1986. First, the domain and semantic layers defined by Brachman and Fox are collapsed into one layer (the semantic layer). Second, we add a new, optional layer, the *syntactic* layer, which is motivated by our conceptual model. This layer may be used to specify a particular arrangement of concepts used to implement a project standard of some type, such as DOD-STD-2167. The syntactic layer is discussed in more detail below.

Implementation Layer

As discussed in Bimson and Burris 1986, the implementation layer consists of machine-interpretable concepts and relations (the declarative knowledge base) as well as rules and procedures that reason about those concepts and relations (the deductive knowledge base). In ART and LISP, the declarative knowledge base is made up of schemata, slots, values, and facts as illustrated in figure 2. The deductive knowledge base is made up of the rules, functions, and procedures that manipulate those structures. All higher-level concepts and relations are implemented using these lower-level language constructs.

Logical Layer

The logical layer determines how implementation layer knowledge is logically structured and interpreted. In ART, *slot name + schema name + slot value* is interpreted as a logical tuple meaning, for example, "the planned-start of the write-Software-User's-Manual-scope-section schema is 3/8." ART places this fact in the knowledge base as (planned-start write-Software-User's-Manual-scope-section (3 8)).

(Schema write-Software-User's-Manual-scope-section

A. Epistemological layer relations replicated in the semantic layer

(element-of task-set)
(instance-of task-prototype)

B. Semantic layer relations

(time-span week)
(planned-start (3 8))
(planned-finish (3 13))
(part-of write-Software User's Manual)
(requires-input overview-tool-description)
(produces-output scope-section)
(steps nil)
(status planned)
(problems nil)
(difficulty level-1)
(priority 3)
(phase design)
(requires-resources (writer pc wordprocessor))
(responsibility-of max)
(planned-cost 2000)
(actual-cost 1725))

Epistemological Layer

Epistemology is "the study of the nature of knowledge" (Webster 1984). The epistemological layer "is the first level at which we consider knowledge to be structured in a meaningful way" (Bimson and Burris 1986, page 9). As defined in Sathi, Fox, and Greenberg 1985, epistemological concepts include *prototype*, *set*, and *individual*. The triadic relationship between these three elemental concepts is illustrated in figure 3, adapted from Bimson and Burris 1986.

A prototype concept represents the common attributes and relations of each member of the set. The set concept contains all individuals having a particular prototypical definition and defines the attributes of the set as an aggregate (such as the number of elements, averages, etc.). An individual concept is an instantiation of a prototype, and a member of a set and has attributes and attribute values that uniquely identify it as an individual. All data are instantiated as instances of the individual concept for a given prototype.

The epistemological relations between these primitive knowledge structures are bidirectional. That is, each has an inverse:

- *Prototype-of* and *has-prototype* relate a prototype to a set and a set to a prototype, respectively.
- *Element-of* and *has-elements* relate an individual to a set and a set to an individual, respectively.
- *Instance-of* and *has-instances* relate an individual to a prototype and a prototype to an individual, respectively.

These three pairs of inverse relations link epistemological concepts to each other in the knowledge base. They also serve other purposes, such as relating epistemological concepts to their semantic layer counterparts (*individual has-instance write-scope-section*) and relating semantic layer concepts

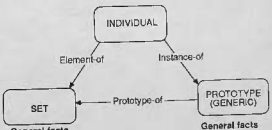

Fig. 3. Epistemological concepts and relations in SPM

to each other (*task-prototype prototype-of task-set*) (fig. 4). Two other pairs of relations also serve these latter two purposes:

- *Subset-of* and *has-subsets* relate the epistemological set concept to semantic layer sets (*set has-subset task–set*) and semantic layer sets to one another (*resource-set has-subset computer-set*).
- *Is-a* and *has-kinds* relate the epistemological prototype concept to semantic layer prototypes (*prototype has-kinds task-prototype*) and semantic layer prototypes to one another (*resource-prototype has-kinds computer-prototype*).

Inheritance flows across the instance-of link from prototypes to individuals when an individual is defined by a user. These three pairs of inverse relations link epistemological concepts to one another in the knowledge base. These relations also are used to link semantic layer concepts to one another. For example, write-Software-User's-Manual-scope-section is an instance-of a task-prototype, which is a prototype-of a task-set, and so forth. In this way, the structure and meaning built up in the triadic relation between epistemological layer concepts is replicated in the semantic layer, as illustrated in figure 4. The replication of these epistemological relations in the semantic layer is illustrated in figure 2a, using an ART schema as the implementation structure. Figure 4 illustrates the completed network of relations. In this case, the epistemological layer concepts (set, individual, and prototype) are related to semantic layer concepts (task-set and task-prototype) via interlayer relations.

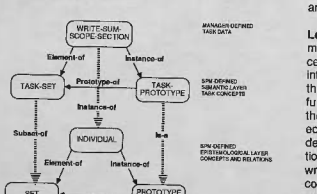

Fig. 4. Relationship of epistemological layer concepts to semantic layer concepts in SPM

Semantic Layer

The semantic layer concepts and relations form the domain-specific knowledge for the knowledge base. It is in the semantic layer that we represent SPM's conceptual model, built from the primitives provided in the lower-level layers. Each semantic layer concept is defined by a triadic construct—as defined in the epistemological layer—composed of a prototype-concept, a set-concept, and an individual-concept. The semantic layer concept *project*, for example, is defined by the triadic construct: *project-prototype + project-set + project-individual* as illustrated in figure 5. These semantic layer concepts are related to one another through intralayer relations such as part-of or requires-input. Likewise, each semantic layer concept is related to the epistemological primitives which define its triadic structure through interlayer relations such as instance-of or is-a, as illustrated by the broken lines in figure 5. The same relations frequently are used as both interlayer and intralayer relations.

Most of the semantic layer concepts introduced in the conceptual modeling section of this paper have been implemented in the SPM prototype. Space does not permit a presentation of all of the concepts discussed for the model as they are implemented in ART, although detailed tables of these concepts and relations are provided in table 4 of Bimson and Burris 1986. Figure 5 gives readers a feeling for the network of relationships in SPM's semantic layer conceptual model. Built into SPM's semantic layer knowledge representation are the notions of *leaf* and *composite* concepts.

Leaf and Composite Concepts. A concept may be either a *leaf* concept or a *composite* concept. A leaf concept is one that is not decomposed into component parts. A composite concept is one that is decomposed into component parts that are fully defined concepts in their own right—that is, they have their own lexical definitions in the knowledge base. Composite concepts form the basis for defining subnetworks associated with tasks, or sections associated with document deliverables. The write-Software-User's-Manual task, for example, is a composite task with four subtasks composing a subnetwork. Similarly, the Software User's Manual is a composite deliverable that is decomposed into various sections. We also distinguish between *homogeneous* and *heterogeneous* composite concepts. Homogeneous composites are composed of parts that are, in essence, morphological clones of the parent concepts. A composite task, for example,

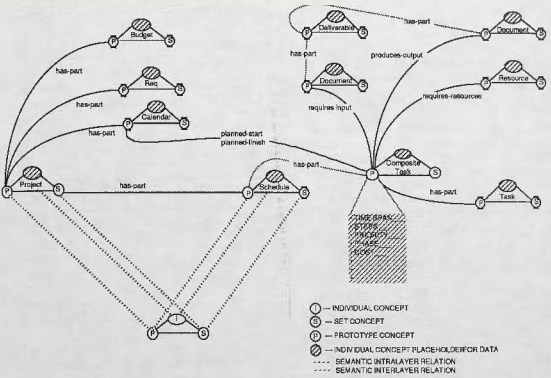

Fig. 5. Network of relations between semantic layer triadic-concepts in SPM

is composed of other tasks that are structurally identical to it. Hetergeneous composites, such as projects, are composed of nonidentical parts such as tasks, resources milestones, deliverables, budget, schedule, organizations, and so forth. In general, we reason about homogeneous composites as sums of their component concepts, whereas we reason about heterogeneous composites as an integrated network of distinctly different components which make up the whole concept, much as we think about the drive train, engine, chassis, and electrical systems as different components making up a car. Using composite concepts to enhance reasoning is discussed in detail in the Reasoning section below.

The values of attributes in the composite concept govern and restrict the values of corresponding attributes in the constituent objects. For example, the planned-start-date of the write-Scope Section must not precede the planned-start-date of the write Software-User's-Manual task, since the former is a component of the latter. In aggregate, all constituent tasks serve to accomplish the composite task's output objective within the aggregate time and resource restrictions imposed by the composite task. It is by means of these composite concepts that SPM addresses the granularity problem, allowing managers to have views of the project that reflect supervisory, middle, and upper management levels of abstraction.

Syntactic Layer

The syntactic layer has not yet been prototyped in SPM because it did not form part of the original SPM program requirements. We consider the syntactic layer to be an optional layer used to structure projects in prescribed ways. It is a layer imposed by some standard, such as DOD-STD-2167, which requires a particular configuration of project phases, milestones, deliverables, tasks, and so on, and a

prearranged sequencing of those structures. The Preliminary Design Review, for example, has a particular set of generic deliverables associated with it, such as the Software User's Manual. It also has a prescribed sequence relative to other milestones, such as the fact that it must precede the Critical Design Review. The idea is to allow a user to define a syntactic template, which may serve as a project standard, and to describe particular projects in terms of that syntax. DOD-STD-2167 is just one configuration around which projects may be defined in SPM. Other configurations can be built for the new NASA Software Acquisition Life Cycle standard or for a rapid prototyping environment. Each of these configurations is simply a different way of structuring the generic software project management concepts. If no particular project configuration is required, a user may ignore the syntactic layer altogether.

REASONING ACTIVITIES ENHANCED BY KNOWLEDGE STRUCTURE ROBUSTNESS

Once the declarative component of a conceptual model is implemented, then small, mission-specific rule bases can be developed to perform various management activities semi-independently. In fact, a robust conceptual model reduces the work of the knowledge base's deductive component, making rule bases easier to modularize and easier to write. In implementing our conceptual model of software project management, each attribute, relation, or concept that was added to the knowledge base enhanced SPM's deductive capabilities in unexpected ways.

Reasoning Over Attributes and Relations

When SPM was in its first stages of prototype development, we began by implementing simple time attributes (time-span, planned-start, planned-finish, actual-start, and actual-finish) and precedence relations (precedes and succeeds) for each task. These pieces of information allowed us to build critical path rules, simple task statusing rules, and cycle detection algorithms (fig. 6, col. 1). For each attribute added to the knowledge base, we found that we were frequently able to add significantly to SPM's deductive capabilities with just a few rules, primarily due to the network of interrelationships built into the conceptual model. For example, once input and output (I/O) relations were added to the task concept definition, it was relatively simple to increase the deductive capabilities of SPM (fig. 6, col. 2) in the following four ways:

Declarative Knowledge Available	Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Time Intervals/Actual Start and Planned	✔	✔	✔	✔	✔
Precedence Relations	✔	(✔)	(✔)	(✔)	✔
Input/Output Relations		✔	✔	✔	✔
Composite Deliverables			✔	✔	✔
Component Dependencies in Composite Deliverables				✔	✔
Milestones with Composite Deliverables					✔
Deductive Power Available	∓	∓	∓	∓	∓
Determine Critical Path	✔	✔	✔	✔	✔
Detect Cycles	✔	✔	✔	✔	✔
Status Current Tasks	✔	✔	✔	✔	✔
Trace Input/Output Dependencies		✔	✔	✔	✔
Infer Precedence Relations		✔	✔	✔	✔
Check Precedence Consistency		✔	✔	✔	✔
Ask Simple "Why" Questions*		✔	✔	✔	✔
Status Deliverables**			✔	✔	✔
Check Completeness			✔	✔	✔
Support Knowledge Aggregation			✔	✔	✔
Infer Task Inputs with Only Outputs Defined				✔	✔
Infer Relations Between Subnetworks				✔	✔
Status Milestones					✔
Make Meaningful Projections					✔

* Why is task N late?
** What must be done to finish Part B Spec?
(✔) Optional

Fig. 6. Declarative knowledge requirements for deductive power

1. *Inferring precedence relations.* We found that we could deduce precedence relations directly from the I/O dependencies themselves. Users may define tasks, task inputs, and task outputs without explicitly defining any precedence relations. One relatively simple rule, compute-precedence-from-I/O-relations (fig. 7) then asserts all of the precedence relations automatically into the task schemata.

2. *Checking precedence consistency.* The inherent redundancy between precedence relations and I/O relations allows managers to procede in one of two ways: (a) they may either concentrate initially on task definitions, including I/O specifications, without worrying about precedence relations, or (b) they may specify precedence relations without regard to complete task definitions. If precedence relations are defined early in the project—case b—and I/O information is added later, then SPM uses a find-precedence-without-I/O-dependency rule to cross-check the manager-defined precedence relations against those inferred by SPM's compute-precedence-from-I/O-relations

```
(defrule compute-precedence-from-I/O-relations ""
   (utterance ? ? (mouse-click left 1 ? ? compute-precedence))
   (schema ?deliverable
           (instance_of proto-deliverable)
           (output_produced_by ?task-1-schema)
           (input_required_by ?task-2-schema))
   (schema ?task-1-schema
             (job-name ?preceding-job)
           (not (succeeding_tasks ?task-2-schema)))
   (schema ?task-2-schema (job-name ?succeeding-job))
   =>
   (printn"?preceding-job PRODUCES ?deliverable WHICH IS REQUIRED BY ?succeeding-job.")
   (if (YES-OR-NO-P "SHOULD I CREATE A PRECEDENCE BETWEEN THE TASKS"
           then
           (assert (add-link ?task-1-schema ?task-2-schema))))
```

Fig. 7. Compute-precedence-from-I/O-relations rule, which computes precedence relations from I/O relations

rule. SPM then advises managers of inconsistencies in their network by informing them, for example, that "Task 1 preceeds Task 2. However, no deliverable dependency exists. Do you want to delete the precedence relation?" Similar rules have been written to check for an output produced by multiple tasks, to find outputs or deliverables with unknown origins, to find tasks producing undefined outputs, to find tasks requiring undefined input and so forth.

3. *Tracing I/O dependencies.* Three simple rules in SPM trace I/O dependencies through the network: one queries the user for the input or output to trace, another initiates the trace, and a third builds the path of dependencies.

4. *Answering status questions.* While some simple task statusing has been developed for SPM using only start and finish time attributes; the addition of I/O relations makes it possible to extend project status explanations in a meaningful way. Questions such as "Why did the write-referenced-documents-section finish behind schedule?" can be answered in SPM by interpreting the project network in the following way:

- Because the WRDS task was planned to start on March 30 (planned-start (3/8))
- But it began on April 5 (actual-start (3/13))
- Because the Instruction Section was not completed until April 4 (actual-finish (4/4) for preceding-task-write-instruction-section)
- And the WRDS task needed the Instruction Section as input to start (output-produced-by write-instruction-section and input-required-by write-referenced-documents- section)

This preliminary explanation capability, based on what we call "interpreted networks," will form the basis of more extensive explanation capabilities planned for SPM in the near future.

In conventional project management systems —those having no conceptual model— adding new attributes, relations, or concepts to the data base requires major modifications to the system. In SPM, the addition of each new attribute immediately enhances the system's capabilities, since the inheritance mechanism provided by ART ensures that all previously defined schemata inherit new attributes or relations upon recompilation. This provides a natural environment for evolving a complex, conceptual model over time.

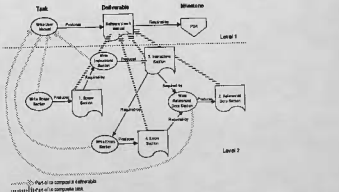

Fig. 8. Relationship between composite tasks and composite deliverables in SPM

Reasoning Over Composite Concepts

Figure 8 illustrates the significance of the interrelationships among composite deliverables and the composite tasks that produce them. Clearly, all sections of the Software User's Manual deliverable should be produced by subtasks of the write-User's-Manual task. This knowledge can be leveraged in many different ways, as shown in the scenario below. This scenario demonstrates the fundamental importance of composite concepts in providing an environment in which to reason intelligently about project management. It also helps to illustrate the current research directions in SPM, since these capabilities are not yet operational in the prototype.

In defining a project, SPM will support decision making by anticipating the needs of the manager in project, schedule, and task definition. After managers have defined the write-User's-Manual task and the Software User's Manual deliverable, and if they have decomposed the Software User's Manual into its component parts, then SPM will be able to anticipate the kinds of subtasks that might be needed to accomplish the write-Software-User's-Manual task. SPM also will be able to infer some of the attributes and relations of each subtask, as well as the time, resource, and cost parameters within which each subtask must be accomplished. For example, SPM will be able to infer the following:

•Each component of the *Software User's Manual deliverable* must be *output* from some task.

•Each component of Software User's Manual is a *section*, which is of type *document*.

•Each component task is therefore a *documentation* task.

• Documentation *requires resources* of type *people* and *word processors*.

In this way, SPM not only will be able to infer what subtasks need to be defined to produce the composite deliverable Software User's Manual but also will be able to infer many of the attributes and relations for each task. Furthermore, if managers have specified a network of interdependencies among the various Software User's Manual sections in defining their deliverable—(instruction-section depends-on scope-section) and (referenced-–documents–section depends-on instruction-section), for example—then SPM also will be able to infer the precedence relations among the tasks it is itself automatically defining (fig. 6, col. 4). Once this is accomplished, SPM can infer when the first task must begin and when the last task must finish and can request verification of its assumptions from the user. Finally, SPM will prompt the user to fill in gaps in the knowledge base that cannot be filled in by its inferencing capabilities.

CONCLUSION

Considerable power can be gained by using conceptual models as a basis for embedding intelligence in decision-support systems, as proved to be the case in building SPM. A robust conceptual model enhances the reasoning capabilities of knowledge-based systems in three important ways:

- It greatly facilitates the building of the reasoning portion of the system (the deductive component).

- It provides a foundation for extending the knowledge base through the addition of new attributes, relations, and concepts and for integrating those additions into the existing network of relationships.

- Each minor extension to the knowledge base is multiplicative in its impact on the potential power of the deductive component.

We have found that using conceptual models is critically important in building a truly intelligent project management decision-support system. In mapping out our course for extending SPM in the near future, we have become increasingly aware of the power of the approach and are convinced that we have only begun to scratch its surface.

REFERENCES

Allen, J. F. 1984. General theory of action and time. *Artificial Intelligence* 23(2).

Bimson, K. D., and L. Boehm Burris. 1986. Knowledge representation in software project management: Theory and practice. Paper presented at Forum on Artificial Intelligence in Management, 20 May, at Defense Systems Management College, Richmond, Virginia.

Bimson, K. D., and L. Boehm Burris. 1987. The craft of engineering knowledge. In *Proceedings of the twentieth annual Hawaii International Conference on System Sciences.* Vol. 1, *Architecture, decision-support systems and knowledge-based systems,* 460-469. North Hollywood: Western Periodicals.

Brachman, R. J. 1977. A structural paradigm for representing knowledge. Ph.D. diss., Harvard University, Cambridge.

Brachman, R. J. .1979. On the epistemological status of semantic networks. In *Associative networks: Representation and use of knowledge by computers,* ed. N.V. Findler, 3-50. New York: Academic Press.

Brachman, R. J., and J. G. Schmolze. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2):272–216.

Fox, M. S. 1983. Constraint-directed search: A case study of job-shop scheduling. Ph.D. diss., Department of Computer Science, Carnegie-Mellon University, Pittsburgh.

Minsky, M. 1975. A framework for representing knowledge. In *The psychology of computer vision,* ed. P.H. Winston, 211-277. New York: McGraw-Hill.

Quillian, M. R. 1966. Semantic memory. Ph.D. diss., Department of Computer Science, Carnegie- Mellon University, Pittsburgh.

Sathi, A., M. S. Fox, and M. Greenberg. 1985. Representation of activity knowledge for project management. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* PAMI-7(5): 531–552.

Sathi, A., T. E. Morton, and S. F. Roth. 1986. Callisto: An intelligent project management system. *AI Magazine,* Winter, 34–52.

Webster's ninth new college dictionary. 1984. Springfield: Merriam-Webster.

Woods, W. A. 1975. What's in a link: Foundations for semantic networks. In *Representation and understanding: Studies in cognitive science,* ed. D. G. Bobrow and A. M. Collins, 35–82. New York: Academic Press.