DOMAIN-SPECIFIC DSS TOOLS FOR KNOWLEDGE-BASED MODEL BUILDING

Meral Binbasioglu
and
Matthias Jarke

July 1985

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

Working Paper Series

CRIS #97

GBA #85-59

Table of Contents

ABSTRACT

The formulation of complex planning models, such as linear programming (LP) systems, is a difficult task that enjoys little support by current decision support systems tools. It is hypothesized that current artificial intelligence technology is insufficient to build generalized formulation tools that would be usable by OR-naive end users. As an alternative, this paper presents a domain-specific approach to knowledge-based model formulation which combines the use of "syntactic" knowledge about linear programming with "semantic" guidance by knowledge specific to some application domain. As a prototype of this approach, a model formulation tool for LP-based production management is under development at New York University.

1. INTRODUCTION

A Decision Support System (DSS) is a computerized system which utilizes knowledge about a particular application area to help decision makers working in that area to solve ill-structured problems [Bonczek, et al, 1984]. DSS need a number of content abilities [Holsapple and Moskowitz, 1983] to support the three stages of the decision making process (intelligence, design, and choice) identified by [Simon, 1960]. In this paper, we examine the content ability of model formulation required mostly in the process phase of design which involves the processes to: clarify and understand the problem; to invent, develop and analyze potential solutions to the problem; and to test the solutions for feasibility. In particular, we are interested in the question how general (i.e., application-independent) a DSS tool for model formulation can and should be made.

DSS generators attempt to offer generalized modelling tools that help managers formulate and solve decision problems. However, the kinds of models offered by such systems tend to be quite simple, involving, e.g., spreadsheet systems but no automatic solution-seeking. As shown in [Dhar, 1984], however, even the formulation and maintenance of large

spreadsheet models may require substantial use of complex Artificial Intelligence (AI) tools.

The difficulties increase if the problem at hand requires the use of more sophisticated goal-seeking models, such as linear programming (LP). Managers typically use an intermediary to get such models built. With this approach, the process of formulating and executing a decision model tends to get quite lengthy and indirect, and the risk of misunderstandings increases. Therefore, it seems desirable to provide the manager with automatic model building tools usable by him directly, rather than through an intermediary.

The approach to model formulation we present in this paper combines structural knowledge about management science models, with application-specific knowledge about a particular domain of interest. The general case for this approach is made in section 2. Section 3 describes a knowledge base structure for the example model formulation tool we have chosen for this research: linear programming models for production management. The capabilities of such a combined knowledge representation technique are illustrated by a detailed example in section 4. Section 5 summarizes the discussion and points out future research directions.

2. MODEL FORMULATION TOOLS IN MODEL MANAGEMENT

As pointed out in [Bonczek, et al., 1984], as well as in a recent survey by Hwang [1985], research in automatic or computer-aided model building is still in its early stages. Research in this area is typically described in the broader context of model management [Elam et al., 1981].

Three levels of model management capability can be distinguished [Bonzcek, et al, 1982]. With the first modelling level, a user procedurally specifies the model's algorithm. As pointed out earlier, this option requires an intermediary if models become complex.

Under the second alternative, a user is familiar with a collection of pre-specified models available to the DSS and selects one of these for execution. User-friendly model manipulation languages can support this task. For example, Blanning [1982] presents a theory of model management where the user views a model as a virtual relation representing a mapping from input attributes to output attributes. Using relational operations, the user can then synthesize more complex models out of existing ones.

Under the third alternative, a user does not directly formulate or select a model; he or she may even be unaware that the DSS uses models in generating responses. Upon receipt of the user's problem description, an appropriate model is selected or composed by the DSS itself. For example, Sivasankaran and Jarke [1985] describe a system called the Actuarial Consulting System (ACS) that composes models in actuarial science (life insurance mathematics) from a library of stored elementary formulas using AI techniques to search through a relational structure similar to Blanning's. Another system -- outside the DSS area -- based on this design principle of "formulation by configuration" is the well-known expert system R1 [McDermott, 1984] which configures VAX computers. If the set of problems under consideration is too broad or unstructured to permit the definition of such a library, models must be formulated from scratch.

Model formulation from scratch consists of two steps: (a) identifying the appropriate modelling technique (e.g., LP, dynamic programming, etc.) and application domain boundaries, and (b) formulating the model within the chosen modelling/domain combination. This paper is concerned with model formulation, i.e., task (b), for LP models; automatic model selection has also been studied recently (see [Dolk and Konsynski, 1984; Goul et al., 1984; Hwang, 1985; Sivasankaran and Jarke, 1985], among others).

LINEAR PROGRAM LIFE-CYCLE

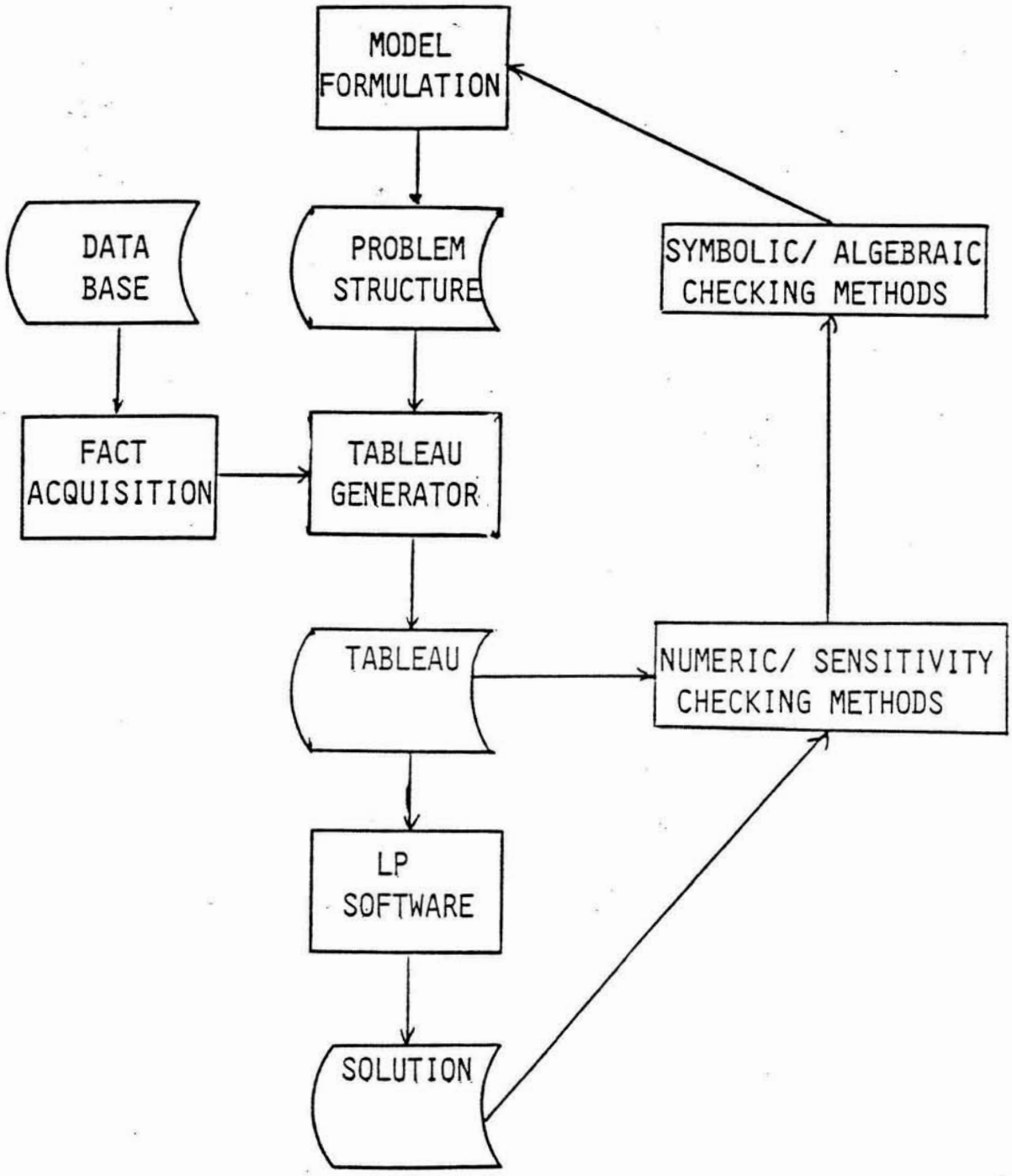


Figure 1

There are at least two approaches to building model formulation tools. The first approach relies on structural knowledge about a particular modelling tool. For example, Murphy and Stohr [1985] propose a LP model formulation tool based on a decomposition approach, relying on the inherent network structure of major portions in almost every large linear program. This tool is chiefly intended to support an operations research specialist in building very large LP models. For a managerial end user, such a system has the drawback that it does not remind the user of application domain-specific knowledge he may have to employ. In other words, the "structural knowledge" approach supports you in formulating a constraint but it does not tell you which constraint to formulate.

Experience with knowledge-based systems ("expert systems") in AI, the rapid growth of the market in tailored domain-specific software packages (i.e., databases for real-estate rather than generic DBMS), and recent work on the definition of Knowledge Base Management Systems [Mylopoulos and Brodie, 1985] all suggest that more model formulation support can be provided if the structural knowledge base component is augmented by an application knowledge base that guides the user not only in the syntactic but also in the semantic aspects of model formulation.

In the remainder of this paper, we describe a formulation tool that combines structural LP knowledge with application knowledge about production management. A PROLOG implementation of such a system is being developed at NYU within the context of a long-range research effort that studies the role of artificial intelligence in management information systems and decision support systems [Jarke and Vassiliou, 1984].

3. LP MODELS IN PRODUCTION MANAGEMENT: KNOWLEDGE REPRESENTATION

3.1. Knowledge Base Structure and Systems Architecture

Linear programming is one of the most successful operations research methods for solving very large optimization problems. Figure 1 illustrates the typical life cycle of linear program development and usage. In this paper, we are concerned with the first step of this life cycle, the conceptual development and symbolic (as contrasted to numeric) formulation of the model. The result of this model formulation step can then be turned over to any of a number of commercially available matrix generators, e.g., LINDO or OMNI. Computerized tools are available for the checking and sensitivity analysis of existing models [Greenberg, 1983]. In contrast, the model formulation step has frequently been considered too fuzzy to be computerized effectively. This is one of the reasons why we propose the combined use of methodological and application knowledge to support this process.

The system architecture is summarized in Figure 2. It divides the model formulation problem into three steps or levels, using different kinds of knowledge bases.

The context identification step accepts the input problem description, and identifies the problem area within a knowledge base for the business application (here: production management). It then refines -- interactively if necessary -- the problem description by identifying all the relevant business objects and the relationships among them.

The problem formulation step instantiates the context identified in the previous stage, determines the decision variables using an additional knowledge base of structural knowledge, assigns indices to the variables, and constructs the format of the constraints and the objective function, using dummy parameter values.

Finally, the model building step selects and accesses (or computes) the parameters that go with the constraints and objective function.

SYSTEM ARCHITECTURE

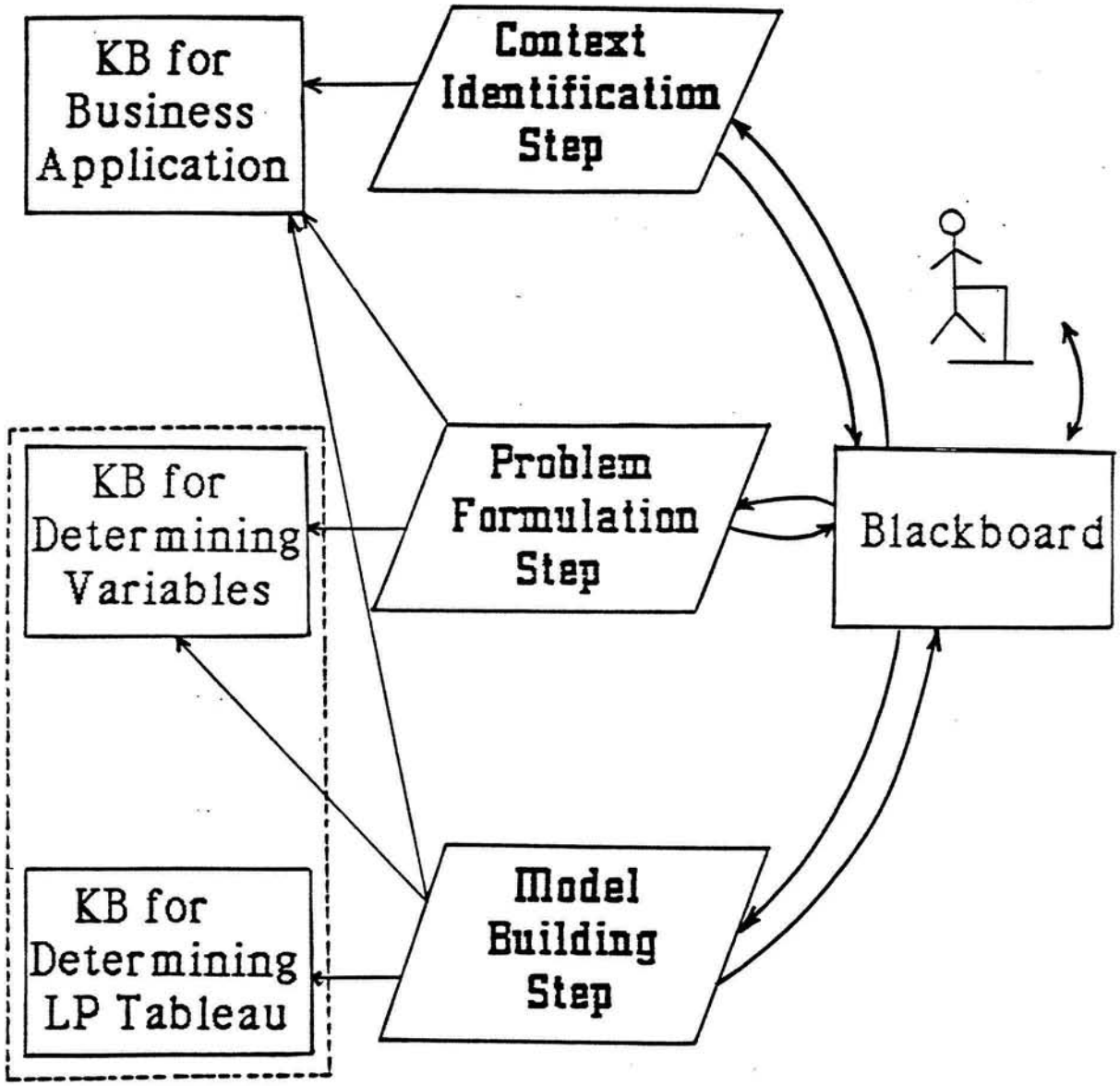


Figure 2

First, each parameter is semantically identified by analyzing the left-hand side and right-hand side components of the constraint. Then, its unit of measure is determined syntactically based on the units of the components. This uses a third knowledge base for the transformation of units of measure and similar relationships.

The idea behind this hierarchical design of the problem solving steps is to approach the problem with a wholistic view [Stefik, 1980]. This will help formulating the problem without optimizing any subpart of it at the expense of the whole. The information at any level determines and coordinates the activities in the next level without preventing return to the upper levels if previous decisions turn out to be wrong. A blackboard is used to store intermediate results, and access to a matrix generator will be provided in case the user wants to see the solution to a partially formulated problem.

In the following subsections, a brief overview of the two knowledge sources of the system will be given. Then, a knowledge representation scheme tailored to the integration of these knowledge sources will be described.

3.2. LP Knowledge Bases

Mathematical models are symbolic representations which incorporate the essential features of actual problems. In particular, a linear programming (LP) problem is a problem of minimizing or maximizing a linear function in the presence of linear constraints. It can be represented mathematically [Bazaraa and Jarvis, 1977; Charnes and Cooper, 1967] as:

Maximize/minimize:

$\Sigma_j \; c_j X_j$

Subject to:

$\Sigma_j \; a_{i,j} X_j \; <= \; b_i$ for each i

X_j >= 0 for each j

where: j = 1, ..., n
i = 1, ..., m

The decision variables, or activity levels, to be determined are represented by X_1, X_2, ..., X_n. c_1, c_2, ..., c_n represent the cost or return coefficients that these variables take. The coefficients $a_{i,j}$ are called the technological coefficients.

Mathematically speaking, the problem of model formulation is the problem of determining the index sets i and j, the decision variables, and the coefficient values. There are many ways to formulate such models. As Charnes and Cooper [1967] point out it is possible for models to be inadequate, to overlook essentials, or to incorporate extraneous features and thereby misrepresent the situation.

In a model, each variable offers a range of possibilities so that the omission of a variable generally eliminates the corresponding opportunities from explicit consideration. The reverse mistake can also be made. Certain conditions may be omitted despite their critical importance. Each additional condition, in general, restricts the range of opportunities. Thus the omission of any constraining condition may allow opportunities to appear which are not relevant to the actual situation.

There are two knowledge bases associated with LP knowledge. The first one will contain knowledge about defining index sets, naming and selecting variables, etc. The second one will be more concerned with the actual procurement and correct interpretation of parameter values.

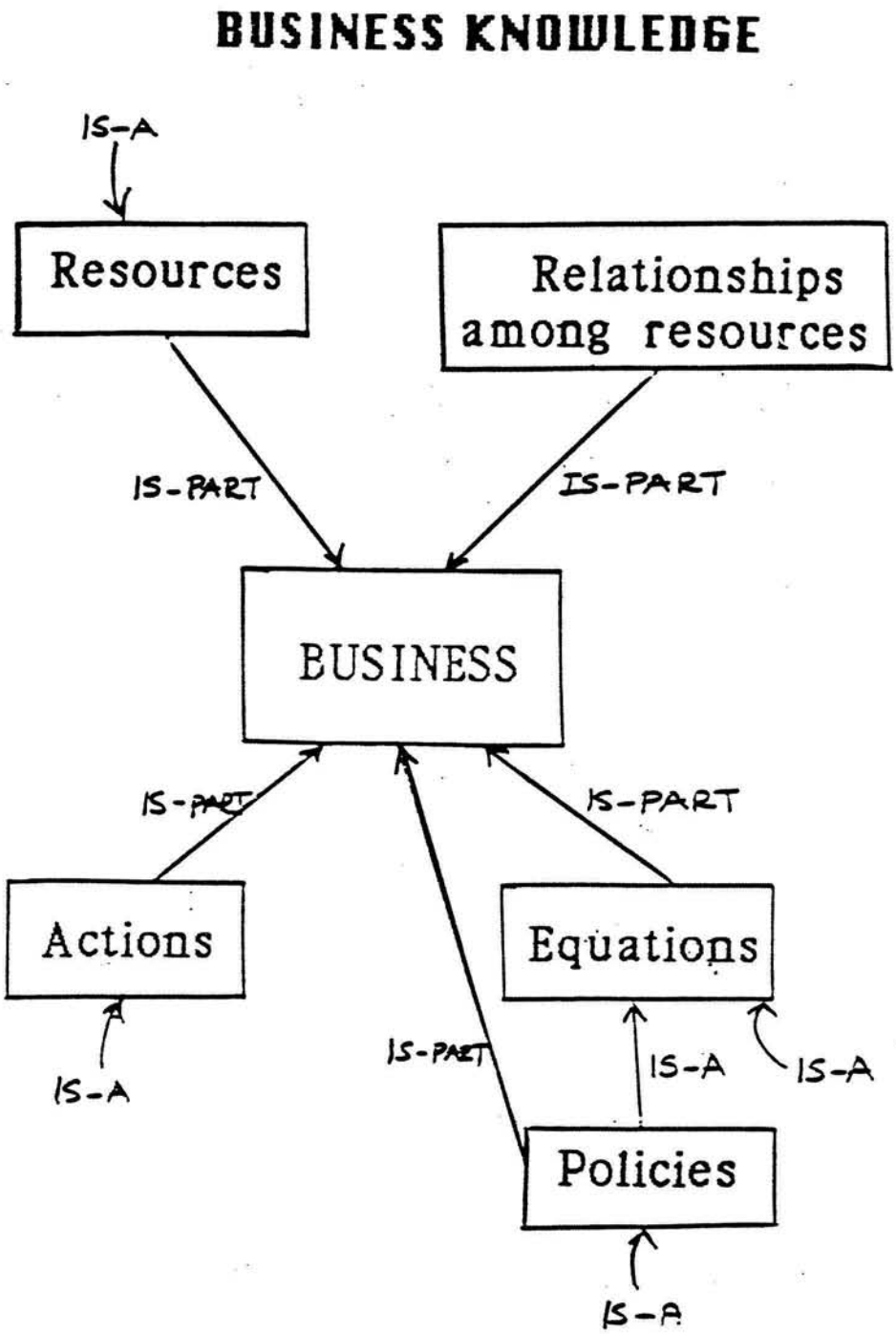


Figure 3

3.3. Application Knowledge Base

The system's domain is resource allocation and production planning. This covers a broad range of problems, such as the selection and allocation of resources, the relative composition and distribution of marketable products, the allocation of resources to products, or any combination of them. If we want a DSS tool to understand business at the level that it can formulate management science models we have to equip it with real-life knowledge about business.

The system has to know the types of resources, their properties, the type of actions that operate on these resources, and all the possible relationships among these components. Some of these relationships may take the form of equations. Therefore, the general pattern of object properties, and of relations among the objects should be identified and represented.

Managerial decisions are based on careful use of resources while achieving the firm's objectives. On a very high level of abstraction, these resources are employees, space, machines, money, and material. The firm plans the allocation of these resources to various activities. Basically, resources have a 'state' and there are some 'actions' which change the states of these resources. For instance, 'hire' and 'fire' are actions which both operate on resource 'employees', where the former increases, the latter decreases the level of employees.

Actions such as procurement of materials or hiring/firing employees, production of products, etc. are governed by the firm's policies. Some possible policies are: limiting overtime to a certain proportion of regular production hours, maintaining a smooth production by not allowing the fluctuations between the periods to be more than a certain percentage, allowing backorders, maintaining a service level of at least a certain proportion of demand, maintaining only a certain fraction of estimated sales for inventories for some time periods, etc.

Actions and changes of the state of the resources should not violate the firm's policies.

Figure 3 summarizes the conceptual relationships between the object types mentioned in this subsection. In the sequel, a knowledge representation scheme tailored to this kind of knowledge will be presented.

3.4. Knowledge Representation Scheme

Knowledge representation can be viewed at two levels [Newell, 1981]. The "Symbol Level" involves looking at knowledge in terms of how it is held, for instance, collection of nodes, or routines for indexing and inheritance. The "Knowledge Level" does not distinguish among the different ways of capturing the same information or even between explicit and implicit information storage. It only considers what the entire body of information says about the world, that is, how well the knowledge base provides a clear picture of the world that it represents [Brachman and Levesque, 1984]. In this paper we will briefly review the representation at the knowledge and symbol levels. Details of the representation and its implementation will be discussed in a forthcoming paper.

The scheme has to represent business knowledge about resources, possible actions on resources and policies that govern these, in the context of planning with LP. Concepts like MACHINE, EMPLOYEE, MONEY, WAREHOUSE, PRODUCTION, SALES, DEMAND, FLOW-EQUATION,... are represented with their possible attributes within the domain of production planning and resource allocation. In addition to the attributes, relationships to other objects will also be specified. These could be resources, actions, equations or relations on and among the resources. For example, MACHINE and PRODUCT have a relation of the form <Product-name, Machine-name, Machine capacity required to produce a unit of product>. We have to define this relation separately from MACHINE and PRODUCT, because it is

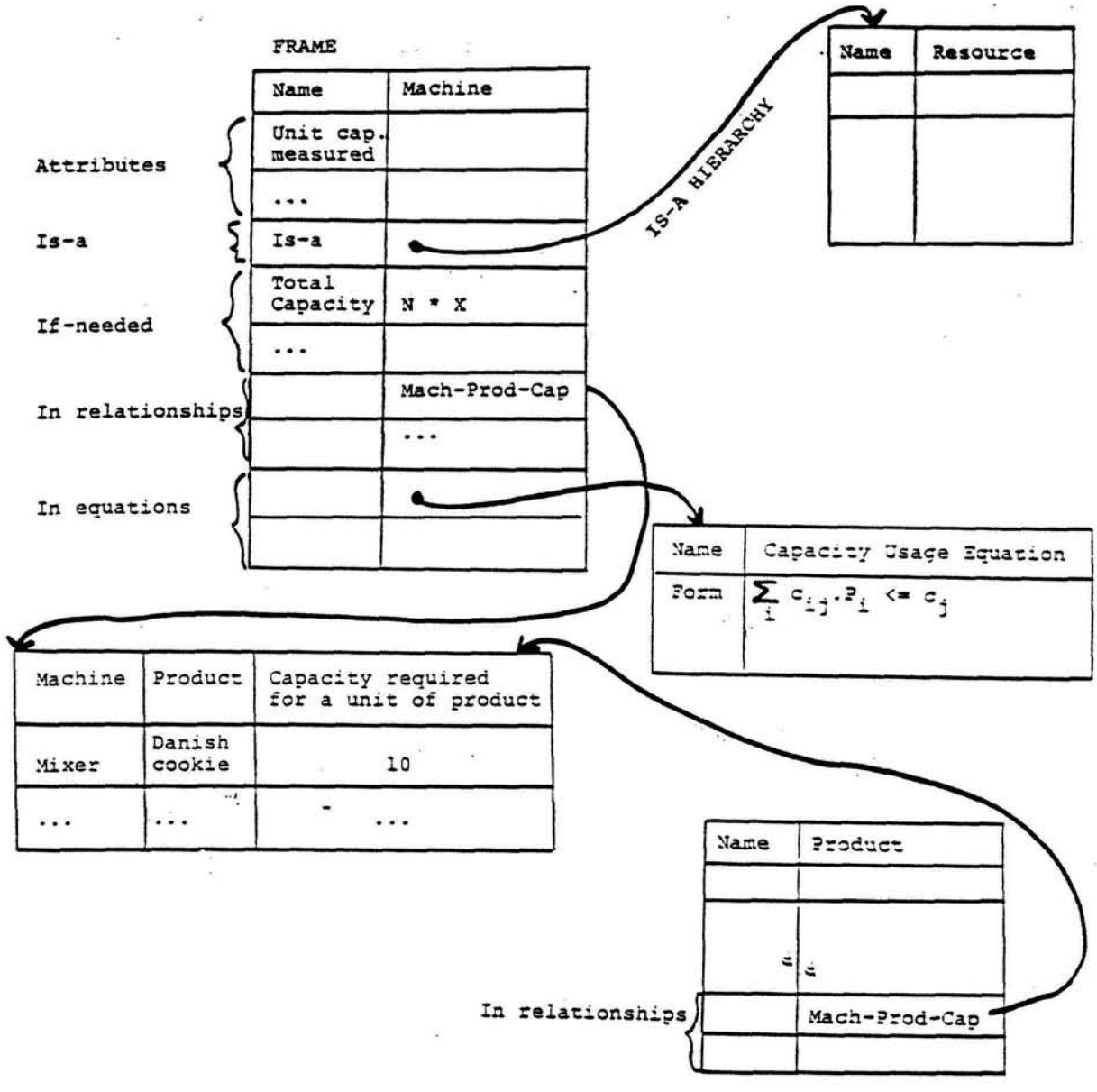
not an attribute unique to any of them. However, the relationship object should be pointed to by both parent objects.

To be able to represent business knowledge, we need to have at least five different types of conceptual objects: resources, relationships among the resources, actions, policies, and equations. We will represent them by using abstraction methods such as aggregation and generalization discussed in Smith and Smith [1977] and Jarke [1982].

In the domain of production management, the highest level of abstraction is BUSINESS, an aggregation of resources, relationships among resources, actions, policies, and equations. The latter are the most generic object types in the business knowledge base. Objects at any given level of abstraction are related to more generic ones through an IS-A hierarchy. For example, the instantiations of the concept 'resources', namely, EMPLOYEE, MONEY, MATERIALS,... are linked to RESOURCES via IS-A links.

At the symbol level, a common use of this hierarchy is to minimize conceptual and storage redundancies by allowing properties associated with general object types to be inherited by more specialized ones, as well as providing the means for the overall organization and management of a large knowledge base [Mylopoulos and Levesque, 1984]. We view property inheritance as a default which the description of the specialized class can override.

Objects will be represented as 'frames' [Minsky, 1975] where all the facts about the given object are attached to slots provided by the frame structure. Some of these slots will be used to describe the properties unique to the given object, others will employ procedural attachments to use when some facts are not explicitly provided. The above-mentioned generalization hierarchy will be achieved by a slot called "is-a" which will store the name of the next generic object the



RULE:

IF X has a state THEN X can appear in a flow-equation.

Figure 4: KNOWLEDGE REPRESENTATION

given object is related to. Depending on the characteristics of the object to be described, the number of slots and the values that are stored in them may vary. All the objects in Figure 3 will be defined using frame representations, except for the "relationships among resources" which will be represented as tables. In addition to these explicit representations of relationships, rules of inference will be employed to derive implicit facts.

Figure 4 illustrates the different knowledge representation techniques used at the symbol level. The knowledge representation scheme is being implemented in PROLOG with added object-oriented capabilities.

4. LP MODELS FOR PRODUCTION MANAGEMENT: MODEL FORMULATION EXAMPLE

In this section, the use of the combined structural and application knowledge base will be illustrated by means of a concrete model formulation example. The interaction syntax will loosely follow a PROLOG syntax. Figure 5 shows an extract of the knowledge base to be used in this example.

4.1. Context Identification Step

The interaction alternates between user-driven and system-driven dialog, depending on the level of initial knowledge the user can express. The function of the context identification step is to locate the relevant area of the knowledge base from which a more detailed analysis of the formulation problem at hand can be initiated.

Consider the production process in a bakery shop. The baker initially has a vague idea that some planning is needed in the area of cookies production, and the purchasing of associated raw materials. Moreover, he suspects that there are constraints on sales, the minimum required service level, the availability of his mixer, and raw material budgets. The initial problem statement would look as follows:

```
?- problem(production(cookies), purchase(raw_material)),
```

```
constraints(mixer, sugar, service_level(cookies),
            sales_limits(cookies)).
```

The system tries to associate the information given to it with certain nodes and arcs in the knowledge base. For example, production(cookies) will be associated with the node "production". Since the system does not know the term "cookies", it has to disambiguate among the possible classes "final product" and "intermediate product". This can be done quite easily since there is a constraint on sales for "cookies"; therefore, cookies must be a final product. However, it might be one final product or a whole class of them (some of which may be stored in the knowledge base already). Thus, the system displays the set of known products and asks the user to check those that belong to the class of cookies:

```
:- WHICH OF THE FOLLOWING
   FINAL PRODUCTS
   ARE "COOKIES"?
    1 - ICE CREAM
    2 - PEANUT_BUTTER_COOKIE
    3 - WHOLE_WHEAT_ROLL
    4 - DANISH_BUTTER_COOKIE
    5 - CHOCOLATE_CHIP_COOKIE
    ...
    OTHERS (LIST) ?
```

The user answers with 2, 4, and 5. Since there is more than one product in the group, the system infers that there is a product mix problem. The knowledge base is amended by the new class definition, as shown in Figure 6.

The system knows that "sugar" is a kind of raw material (in the bakery) and infers that there is a constraint on sugar availability by looking at the node for "raw material" and then further at the node "resource" where it finds out that resources tend to be limited.

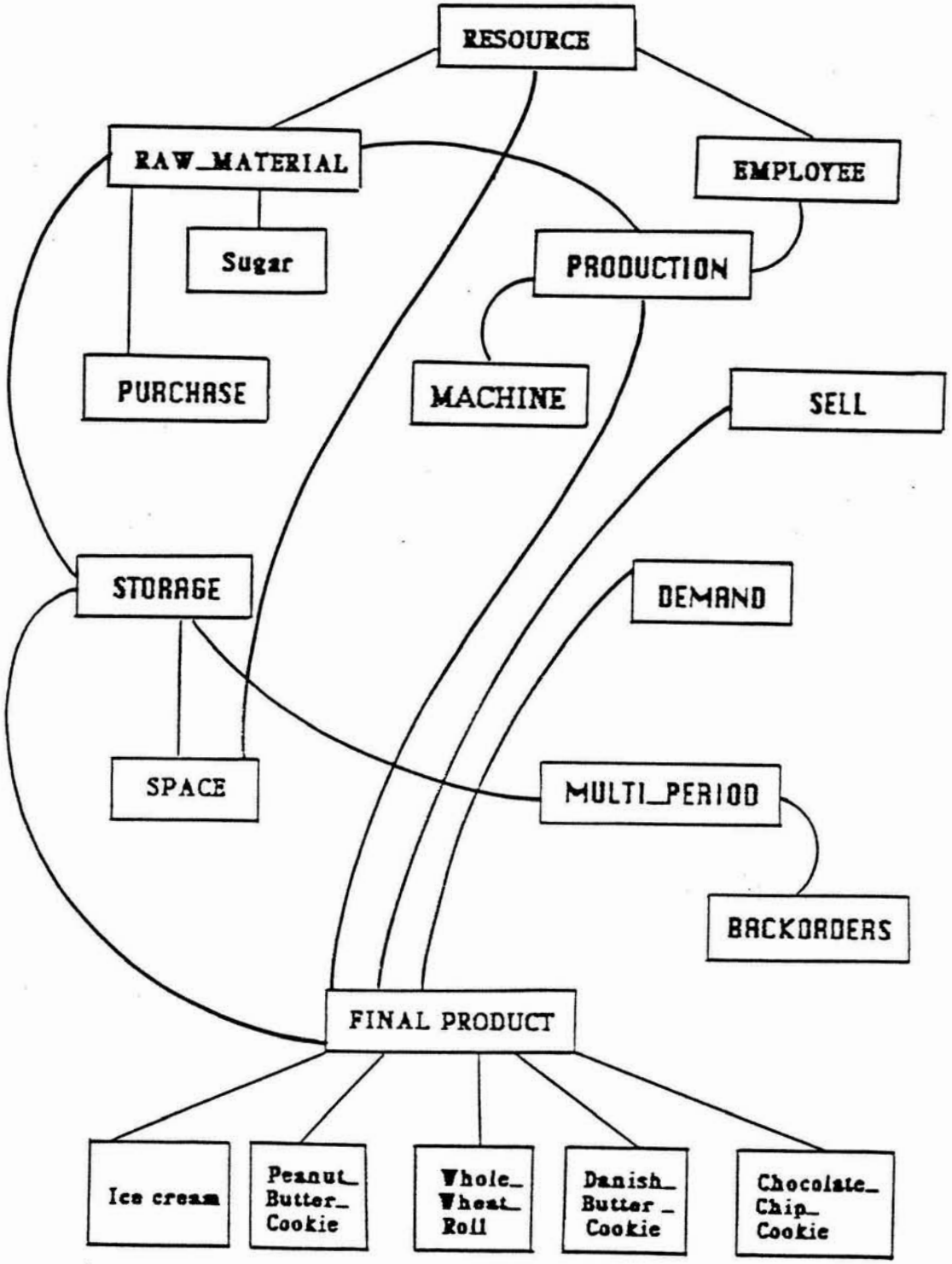


Figure 5: Initial Knowledge Base

On the other hand, the system doesn't know the term "mixer". All it knows is that "mixer" refers to a constraint. A system-driven dialog with the user is initiated to determine the meaning more precisely. The system knows that most constraints are associated with resources and can thus follow down the IS-A hierarchy of resources, using menu selection:

```
:- IS "MIXER"
   - EMPLOYEE (1)
   - MACHINE (2)
   - RAW MATERIAL (3)
   - ...
   - NO RESOURCE
```

The user could answer this by selecting (2). The system infers that the constraint is a capacity constraint; furthermore, it can continue the dialog to find out whether "mixer" is a synonym for some machine instance appearing in the knowledge base. If that is not the case, it would have to request units of capacity measure, the capacity itself, etc., in order to fill the machine-type slots.

By now, the system has marked all the nodes mentioned in the original problem statement. The next step is to check for incompleteness of the problem statement. Incompleteness is detected in two ways. First, the system asks the user whether certain neighbors of the marked nodes are also of interest. For example, it may ask whether there are employee problems (coming from the production node). Similarly, it may suggest the existence of storage problems (coming from cookies via its generalization, final products). Assume that the user answers the latter affirmatively.

The second method of detecting incompleteness identifies disconnected components of the knowledge base and tries to establish additional nodes that connect these components. For example, since there is a storage problem associated with the sales/product mix problem, the system hypothesizes that the problem is really a multi-period problem. Indeed, this is confirmed by the user.

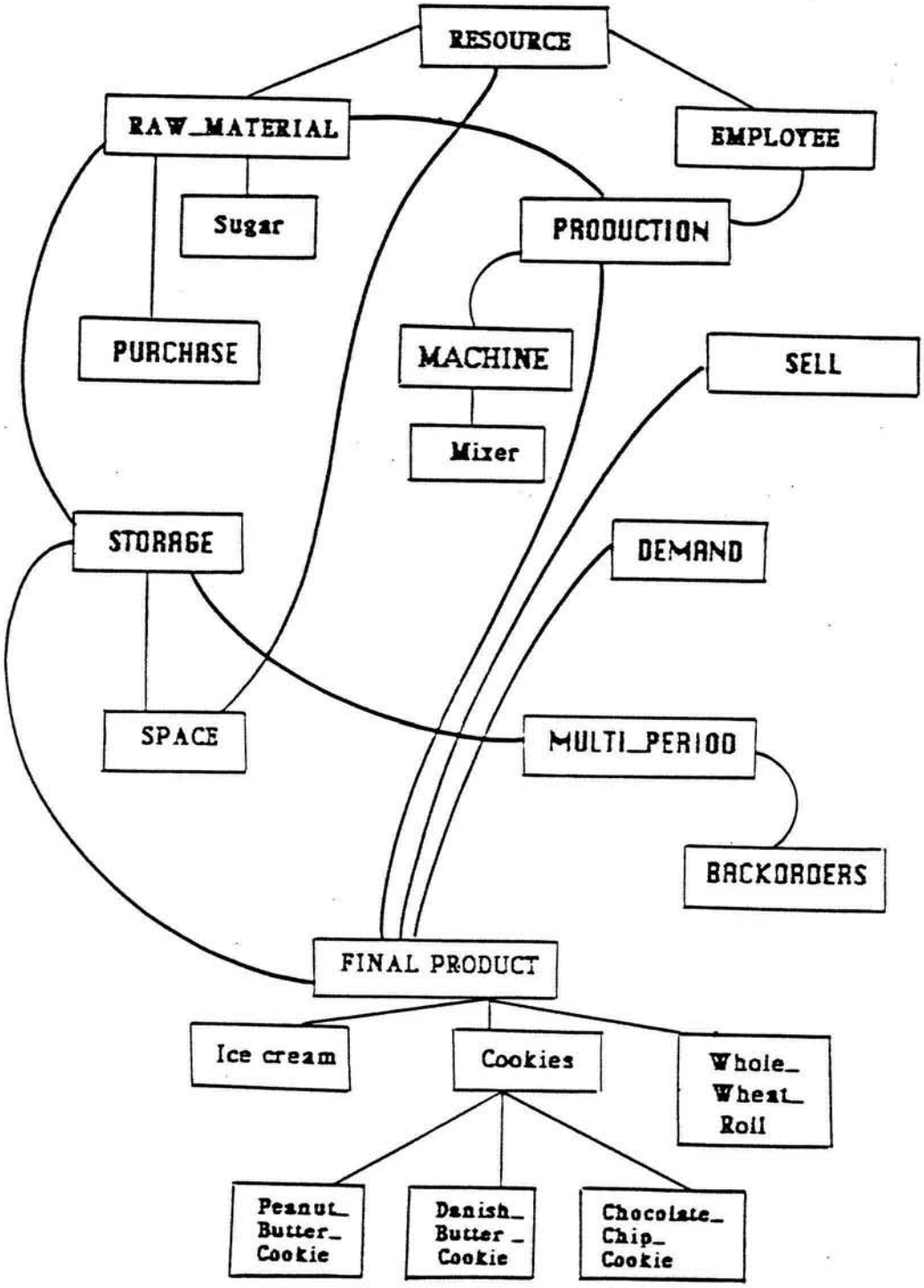


Figure 6: Knowledge Base after problem statement is disambiguated

In summary, the context identification step has produced the following result. The problem was originally stated as a production problem of cookies, with purchasing of raw material sugar. Using the application-specific knowledge base, the system refined the problem definition to a multi-period, product-mix, and purchasing problem with storage considerations.

4.2. Problem Formulation Step

After identifying the boundaries of the problem context, the system proceeds to assist the user in determining the necessary constraints, as well as specifying the format of these constraints and of the objective function.

The first step in this process is the choice of a suitable problem decomposition. Metarules for this step essentially follow the principle of minimal coupling and maximum cohesion among subproblems known from structured design [DeMarco, 1978]. These metarules are applied both to the initial decomposition of the problem, and to the later integration of submodels. The example problem is initially decomposed as shown in Figure 7 into two subproblems: product-mix (problem-I) and raw material purchasing (problem-II). Note, that both problems are coupled only via the decision variables to be associated with raw material.

Next, the decision variables and their position in relevant constraints and in the objective function are determined for each subproblem. The relevant constraints are retrieved from the knowledge base using the "in-equation" slots of the object frames identified in the Context Identification Steps (see Figure 4). However, the system can ask the user to confirm their relevance in a particular case.

The KB for structural knowledge is now employed to construct the actual constraints and objective function. This step is shown for some example equations of the bakery example, below. We shall consider the following constraints for problem-I.

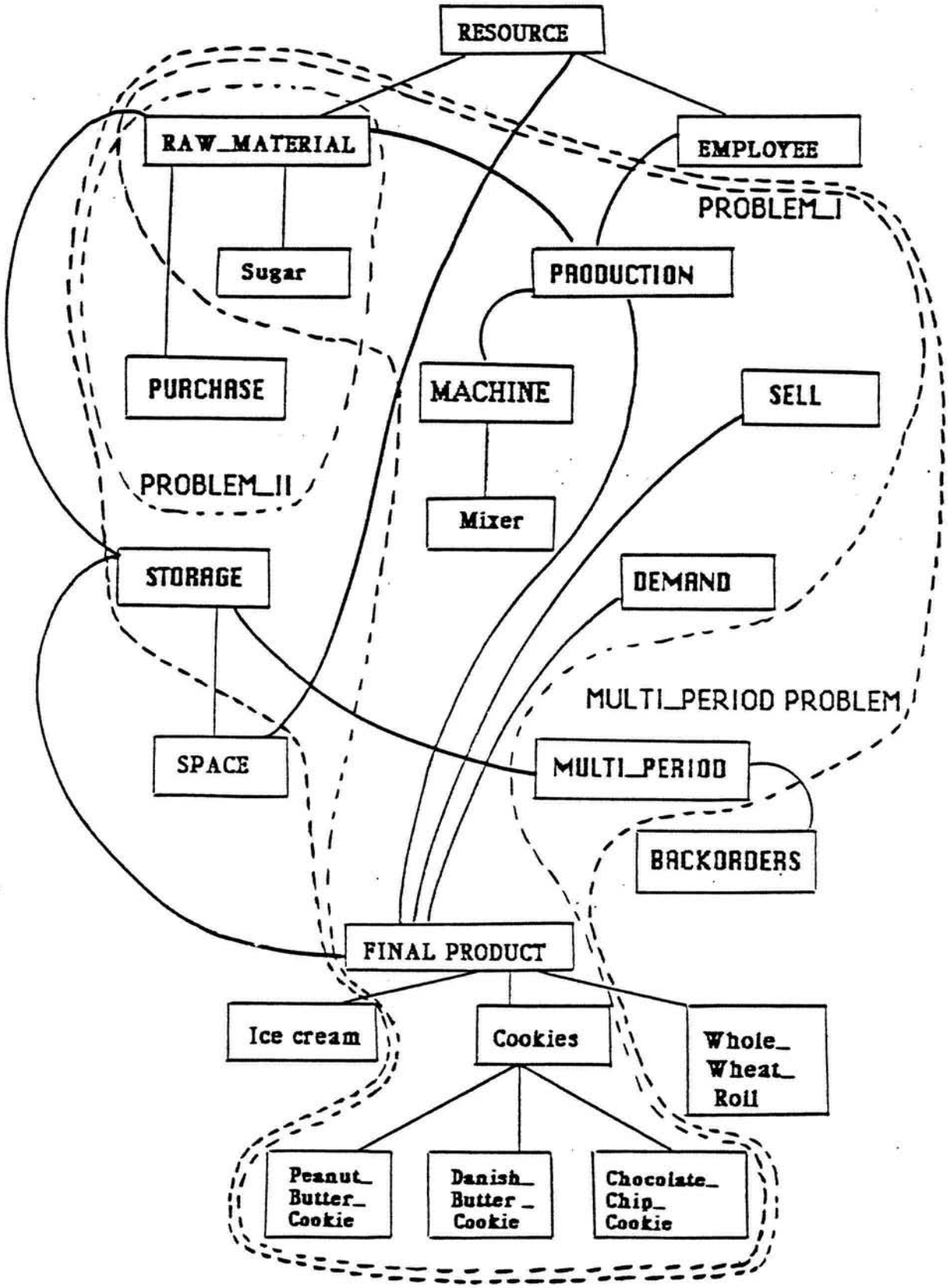


Figure 7: Problem Decomposition

(I.1) Machine availability constraint
(I.2) Raw material availability constraint
(I.3) Market limitation (Demand)
(I.4) Service level
(I.5) Objective function.

A major problem in LP formulation is the choice of the decision variables and their index sets. The following rule can be used to determine the decision variable: "IF the aim is to determine the production level of final products THEN the decision variable, P, is the level of final product to be produced." Another rule says that "IF the problem type is product-mix THEN an index for product (say i) is needed." Now consider the actual formulation of the equations for subproblem I.

(I.1) Machine Availability Constraint:

The stored form of this constraint looks as follows:

$$\Sigma_i \ \text{capusage}_{i,j} P_i <= \ \text{Capacity}_j$$

where:
$\text{capusage}_{i,j}$ = Units of time each unit of product i requires on machine j.

Using the knowledge acquired during the Context Identification Step, this standard form can be specialized. Since the problem is product mix, the index i is required and takes the values defined in the set "cookies", i.e., "PEANUT_BUTTER_COOKIE, DANISH_BUTTER_COOKIE, CHOCOLATE_CHIP_COOKIE". On the other hand, the system knows that there is only one machine which could be a bottleneck, namely the "MIXER". Therefore, the index j can be dropped. Thus, we get the specialized constraint:

$$\Sigma_{i \varepsilon \text{COOKIES}} \ \text{capusage}_{i,\text{MIXER}} P_i <= \text{Capacity}_{\text{MIXER}}$$

In a similar way, the other constraints can be specialized:

(I.2) Raw Material Availability Constraint:

$$\Sigma_{i \in COOKIES} \text{rawusage}_{i,SUGAR} P_i <= \text{Availability}_{SUGAR}$$

(I.3) Market Limitation (Demand)

$$P_i <= \text{Demand}_i$$

(I.4) Service Level Equations

$$P_i >= \text{MinimumSales}_i$$

(I.5) Objective Function

$$\text{Maximize } \Sigma_{i \in COOKIES} \; P_i \text{Contribution}_i$$

In equal fashion, we determine constraints and objective function in problem-II:

(II.1) Meet the Internal Demand:

$$\text{Purchase}_{SUGAR} > = \Sigma_{i \in COOKIES} \text{rawusage}_{i,SUGAR} P_i$$

(II.2) Objective Function:

$$\text{Minimize cost}_{SUGAR} \text{Purchase}_{SUGAR}$$

Note that in (II.2) the Σ sign has been removed from a standard formula since there is only one summand.

When the system combines the two subproblems, it cannot simply use the existing equations but has to modify them. While equations I.1, I.3, and I.4 will be used as they are, equation I.2 will be merged with II.1:

$$\text{Availability}_{SUGAR} + \text{Purchase}_{SUGAR} >= \Sigma_{i \in COOKIES} \text{rawusage}_{i,SUGAR} P_i .$$

Moreover, the global objective function will include components from both subproblems. A new contribution margin is computed for the products which does not include the SUGAR costs. This change in contribution margin is necessary because the combined model takes care of raw material cost explicitly.

The combination of the subproblems must also take into account that the problem is multi-period. We can purchase raw materials earlier to use in production in later periods or produce final products earlier to meet the future demand. Therefore, IF the problem is a multi-period problem THEN it is necessary to distinguish the quantities of each raw material bought, used and stored and the quantities of each final product produced, sold and stored at each time period. Moreover, there is a rule that "IF a problem is multiperiod THEN add an index t to all variables". (There are also additional, more complex rules which are skipped here for simplicity of exposition.)

All the previous equations should be indexed by t to accomodate the time feature in multi-period analysis. Equation I.3. has to be changed to reflect the fact that Sales for any period -instead of production- must be less than the Demand for the period. The objective function has to be modified and balance flow equations have to be added.

(I.3'):

$$Sales_{i,t} <= Demand_{i,t}$$

We assume here (realistic in a bakery) that all demand not satisfied in the period is lost. If backorders are allowed sales will be a function of current demand and previously unfilled demand. In addition, another rule states that multi-period problems require balance flow equations (cf. Figure 4):

$$Storage_{SUGAR,\ t-1} + Purchase_{SUGAR,t} = Usage_{SUGAR,t} + Storage_{SUGAR,t}$$

$$Storage_{i,t-1} + P_{i,t} = Sales_{i,t} + Storage_{i,t}$$

where $i \varepsilon$ COOKIES.

The new objective function will use the sales level instead of the production level, and will accomodate the minimization of storage costs. The final version has the following form:

Maximize

$$\Sigma_t \Sigma_i \left(Sales_{i,t} NewContribution_{i,t} - Storage_{i,t} StorCost_{i,t}\right)$$

$$- \left(\Sigma_t \left(Purchase_{SUGAR,t} Cost_{SUGAR,t} + Storage_{SUGAR,t} StorCost(SUGAR,t)\right)\right.$$

4.3. Model Building Step

After the completion of the model structure, the next and final step of model formulation is the instantiation of the right-hand-side and coefficient values. If these values are available explicitly in the knowledge base they are just retrieved. If there are "if-needed" slots in the related frames (cf. Figure 4), the values are computed using the formulas in these slots. Otherwise, the user is requested to supply the missing values.

This completes the formulation of the model. The result is now converted into a suitable matrix generator format and submitted for computation. In the discussion above, we have neglected the important issue of consistency checking. Although some of the consistency problems present in model formulation are removed by the knowledge-based approach presented here, other issues will remain. Structural linear programming knowledge will be required to do the associated checks; see [Murphy and Stohr, 1985] for a detailed discussion.

5. CONCLUDING REMARKS

The example in the previous section should have demonstrated the usefulness of domain-specific knowledge in model formulation. Without such knowledge, little guidance can be expected from a formulation support tool. Instead, the tool will have to focus on problem structuring and consistency checking. Both are extremely important, especially in the formulation of very large models, such as envisioned by the syntactic tool developed by Murphy and Stohr [1985] in a parallel effort. However, if model formulation by end users is intended, semantic guidance must also be offered. This confirms a similar result of Dhar [1984] who developed a spreadsheet formulation system in a manufacturing environment which also relies heavily on domain-specific knowledge.

A system incorporating the capabilities described in this paper is being implemented in a version of PROLOG [Clocksin and Mellish, 1981] enhanced by object-oriented features that facilitate the implementation of frame representations, as in Figure 4. In further research, we shall try to integrate this knowledge-based tool with the more structurally oriented method of [Murphy and Stohr, 1985].

Another question of substantial interest is the construction of a meaningful domain knowledge base. Bouwman [1983] describes a way to extract a knowledge base (in financial analysis) from experienced analysts, essentially modelling the psychological structures of the analysts as objects of the knowledge base. Our initial solution is based more on textbook knowledge of the firm. Experience with the actual system will have to show whether that level of knowledge is sufficient, and what will be the optimal scope of the application domain. Finally, as shown in the cookie example, the interaction with the end user can also lead to incremental enhancement of the knowledge base through limited machine learning features.

6. REFERENCES

Bazaraa, M., and Jarvis, J.J. (1977). Linear Programming and Network Flows, John Wiley and Sons.

Blanning, R.W. (1982). What is happening in DSS? Interfaces, 13(5), pp.71-80.

Blanning, R.W. (1985). A relational theory of model management, Working Paper, No. 85-106, Owen Graduate School of Management, Vanderbilt University.

Bonczek, R.H., Holsapple, C.W., and Whinston, A.B. (1981). Foundations of Decision Support Systems, Academic Press.

Bonczek, R.H., Holsapple, C.W., and Whinston, A.B. (1982). The evolution from MIS to DSS: extension of data management to model management, in Ginzberg, M.J., Reitman W.R., and Stohr, E.A. (eds.), Decision Support Systems, North-Holland.

Bonczek, R.H.; Holsapple, C.W. and Whinston A.B. (1984). Developments in decision support systems, Advances in Computers, Vol.23, Academic Press, pp.141-175.

Bouwman, M.J. (1983). Human diagnostic reasoning by computer: an illustration from financial analysis, Management Science, 29(6), pp.653-672.

Brachman, R.J., and Levesque, H.J. (1984). What makes a knowledge base knowledgeable? a view of databases from the knowledge level, Proceedings First International Workshop on Expert Database Systems, Kiawah Island, S.C.

Charnes, A., and Cooper, W.W. (1967). Management Models and Industrial Applications of Linear Programmimg, John Wiley and Sons.

Clocksin, W.F. and Mellish, C.S. (1981). Programming in Prolog, Springer-Verlag.

DeMarco, T. (1978). Structured Analysis and System Specification, Yourdon.

Dhar, V. (1984). PLANET: an intelligent decision support system for the formulation and investigation of formal planning models, unpublished Ph.D. thesis, University of Pittsburgh, Pa.

Dolk, D., and Konsynski, B.R. (1984). Knowledge representation for model management, IEEE Transactions on Software Engineering, 10.

Elam, J.J., Henderson, J.C., and Miller, L.W. (1980). Model management systems: an approach to decision support in complex organizations, Proceedings First International Conference on Information Systems.

Goul M. et al. (1984). Designing the expert component of a decision support system, paper delivered at the ORSA/TIMS meeting, San Francisco.

Greenberg, H.J. (1983). A functional description of ANALYZE: a computer-assisted analysis system for linear programming models, ACM Transactions on Mathematical Software, 9(1, pp.18-56.

Holsapple, C.W. and Moskowitz, H. (1980). A conceptual framework for studying complex decision processes, Policy Sciences, 12(1).

Hwang, S. (1985). Automatic model building systems: a survey, DSS-85 Transactions, San Francisco, Ca., pp.22-32.

Jarke, M. (1982). Developing decision support systems: a container management example, Journal of Policy Analysis and Information Systems, 6(4), pp.351-372.

Jarke, M., and Vassiliou, Y. (1984). Coupling expert systems and database management systems, in Reitman, W. (ed.), Artificial Intelligence Applications for Business, Norwood: NJ: Ablex, pp.65-85.

McDermott, J. (1982). R1: a rule-based configurer of computer systems, Artificial Intelligence, 19(1), pp.39-88.

Minsky, M. (1975). A framework for representing knowledge, in Winston, P.H. (ed.), The Psychology of Computer Vision, New York: McGraw-Hill, pp.211-277.

Murphy, F.H. and Stohr, E.A. (1985). An intelligent system for formulating linear programs, New York University Working Paper Series CRIS #95, GBA 85-40.

Mylopoulos, J., and Brodie, M.L., eds. (1985). On Knowledge Base Management Systems, Springer-Verlag.

Mylopoulos, J., and Levesque, H.J. (1984). An overview of kKnowledge representation, in Brodie, M.L., Mylopoulos, J., and Schmidt, J.W. (eds.), On Conceptual Modelling, Springer-Verlag.

Newell, A. (1981). The knowledge level, AI Magazine, 2(2, pp.1-20.

Simon, H. (1960). The New Science of Management Decisions, Harper and Row, New York.

Sivasankaran, T.R., and Jarke, M. (1985). Knowledge-based formula management strategies in an Actuarial Consulting System, Decision Support Systems, 1(3).

Smith, J.M., and Smith, D.C.P. (1977). Database abstraction: aggregation, Communications of the ACM, 20(6), pp.405-413.

Stefik, M.J. (1980). Planning with constraints, Ph.D. dissertation, Computer Science Department, Stanford University.