1991-09-26

On the integration of data and mathematical modeling languages

Bhargava, Hemant K.

Monterey, California. Naval Postgraduate School

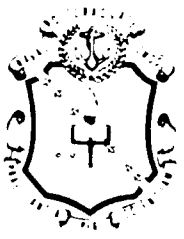
Department of Administrative Sciences

ON THE INTEGRATION OF DATA AND MATHEMATICAL MODELING LANGUAGES

Hemant K. Bhargava
and
Ramayya Krishnan, Sumitra Mukherjee

September 26, 1991

Working Paper No. 91-05



Naval Postgraduate School
Monterey, California

Working papers of the Naval Postgraduate School Department of Administrative Sciences are preliminary materials circulated to stimulate discussion and critical comment. The views stated herein are the author's and not necessarily those of the Department of the Navy or the Naval Postgraduate School.

List of working papers on inside backcover.
For additional copies, write to:

Department of Administrative Sciences
Working Paper Series
Code AS
Naval Postgraduate School
Monterey, California 93943-5026
(408) 646-2471

On the Integration of Data and Mathematical Modeling Languages

Hemant K. Bhargava
Code AS/BH
Naval Postgraduate School
Monterey CA 93943 *

Ramayya Krishnan
SUPA
Carnegie-Mellon University
Pittsburgh PA 15213

Sumitra Mukherjee
SUPA
Carnegie-Mellon University
Pittsburgh PA 15213

September 26, 1991

Abstract

This paper examines ways in which the addition of data modeling features can enhance the capabilities of mathematical modeling languages, and demonstrates how such integration might be achieved as an application of the embedded languages technique proposed by Bhargava and Kimbrough, [4]. Decision-making, and decision support systems, require the representation and manipulation of both data and mathematical models. Several data modeling languages as well as several mathematical modeling languages exist, but they have differences sets of capabilities. We motivate with a detailed example the need for the integration of these languages. We describe the benefits that might result, and claim that this could lead to a significant improvement in the functionality of model management systems. Then we present our approach for the integration of these languages, and specify how the claimed benefits are realized.

*This author's work on this paper was performed in conjunction with research funded by the Naval Postgraduate School.

1 Introduction

This paper examines ways in which the addition of data modeling features can enhance the capabilities of mathematical modeling systems, and presents a methodology for integrating data and mathematical modeling languages. Our research is based on the recognition that decision-making, and decision support systems, require the representation and manipulation of both data and mathematical models (see e.g., [42]). Research in database systems has led to the development of several data modeling languages (e.g., languages based on the semantic data model [29]). Similarly, several languages have been proposed for mathematical modeling as well (e.g., algebraic modeling languages [21]). However, these two sets of languages have usually been developed independently of each other, and have various differences in their capabilities. Data modeling languages have few features for representing mathematical relationships between elements of the domain, while mathematical modeling languages lack many of the facilities, found in data modeling languages, for representing qualitative relationships between these elements.

There is a consensus among researchers (e.g., [6, 4, 27, 16]) that decision support and modeling systems should support the entire modeling life-cycle. In recent years there has been much research on the model management component of a decision support system, aimed at model representation and manipulation. This work has been based on several different approaches such as structured modeling [27, 26], graph-based modeling [31], embedded languages [6, 4], and executable modeling languages [21, 20, 9, 4, 24]. These approaches have led to the development of several model representation languages, such as SML [24] and LSM [12] (for structured modeling), $L^{\uparrow}$ and $L_{\downarrow}$ [6, 4] (in the embedded languages approach), and AMPL [20] (an executable modeling language). It has resulted in several general modeling systems, including FW/SM [23] for structured modeling, TEFA [6]—based on embedded languages, NETWORKS [31]—a graph-based modeling system, and GAMS [9]—based on an algebraic executable modeling language, as well as special-purpose systems to support specific modeling activities. For example, ANALYZE [28] supports understanding and analysis of linear programming models and solutions, and LPFORM

[35] supports the formulation of linear programs.

These approaches have emphasized the development of mathematical modeling capabilities, but have largely ignored data modeling features,[1] e.g., the representation of set-theoretic structural, qualitative relationships[2] that exist among the elements being modeled. Such relationships influence the modeling process, and the mathematical models these systems represent, in several ways. One, many potential users of mathematical modeling techniques find it easier to conceptualize a problem in terms of the data modeling relationships rather than the mathematical relationships ([32]. Two, problem-specific data is required for the solution of these models—it is desirable to access this data from an existing database rather than to create, and maintain, a copy of the data for the solver being used. Three, the structural and qualitative relationships are often the foundations of, and the assumptions underlying, the mathematical relationships that approximate the real problem. The ability to represent and reason with the justifications for the mathematical formulation is particularly useful in model formulation, in model communication, in understanding model solutions, and in model maintenance [40].

While several languages (e.g. semantic data modeling languages) do exist for the representation of such semantic relationships, these have few mathematical modeling capabilities. There is, therefore, a need to integrate these two sets of capabilities within a single framework. One way to achieve such integration is to create a new unifying conceptual framework and modeling language, as has been done in the case of structured modeling. Another is to provide systematic means for integrating existing languages, thus allowing their users to continue using those languages. Our research adopts the latter alternative.[3]

[1] Structured modeling is an exception to this statement, but, as we argue in the sequel, our approach is fundamentally different from that taken in structured modeling.

[2] These include what are often termed *abstraction* relationships (aggregation, generalization, grouping, specialization) as well as other qualitative relationships (e.g., a *supply* relationship between a set of plants and a set of customers) between model elements, which are reducible to set-theoretic operations.

[3] There is another fundamental distinction between our approach for such integration and that of structured modeling. In structured modeling, the design of the relational scheme (elemental tables) is customized to meet the input and output requirements of a particular structured model. (In fact, the normalized tables can be generated automatically from model schemas.) This relational scheme could either define tables that actually store the elemental data, or could define a view of other existing tables. In our approach, the data modeling can be entirely independent of the data requirements of individual mathematical models. The

The rest of this paper is organized as follows. In §2 we discuss the principal benefits (from the perspective of mathematical modeling systems) that should accrue from the integration of mathematical and data modeling features, and illustrate these benefits with an example. In §3, we explain how such integration can be achieved in a systematic and generalizable manner. We propose that the embedded languages technique, proposed by Bhargava and Kimbrough [4], is a useful technique for achieving our objectives. We use this technique to integrate a) a generic executable mathematical modeling language L_m, and b) an executable data modeling language L_d, within a common language. In §4 we illustrate, using our earlier example, how the potential benefits discussed in §2 are realized, and how the functionality of mathematical modeling systems can be improved as a result. The final section (§5) discusses the contribution of our work and suggests directions for further research.

2 Motivation

In this section we present our motivation for developing a language that integrates data and mathematical modeling features. We do so by arguing that certain desirable features can be implemented in modeling languages and systems only if the modeling language is able to represent both data and mathematical relationships. We begin by considering a problem faced by designers of a telecommunication network for a hypothetical firm.

Example 1 *Communications Network Design*

> Host computers and terminal controllers are to be connected to concentrators. The terminals are partitioned into various clusters, with each cluster being controlled by a terminal controller. A host computer may also serve as a terminal controller. For most design purposes, host computers and terminal controllers are equivalent, and are considered customer sites. The telecommunications network consists of connections between these customer sites and concentrators. The concentrators and customer sites are called network elements.

relationships between the data stored in the database and the inputs or outputs of the models are captured by explicitly declared mappings.

Each site must be served by exactly one concentrator, though the same concentrator may serve various sites. The average load (the data traffic to and from the host) offered by each host computer and each terminal is known, and is measured in bits per second. For a terminal cluster, the sum of the loads at all the terminals in the cluster is regarded as the load for that cluster. There is an upper limit, called the maximum bandwidth, to the data flow that can be handled by each concentrator.

The existing links between customer sites and concentators are of two types: leased links, and owned links. An existing link is "valid" if and only if the customer site and concentrator it connects are compatible, i.e., they support the same protocol. In general, the cost of using an owned link is proportional to the speed of the link. The cost of using a leased link is a non-linear function of the traffic on the link. However, we consider a simplified scenario in which all link usage costs are constant, irrespective of the type of link. There are also fixed set-up costs associated with locating and operating concentrators.

The objective is to develop a linkage and location plan that satisfies the load requirements of customer sites at a minimum cost.

The problem of developing the linkage and location plan can be formulated as a mathematical programming model, as represented below (the cost of using "invalid" links is set to infinity).

$$\text{Minimize} \quad \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{S}} C_{ij} X_{ij} + \sum_{i \in \mathcal{C}} F_i Z_i \tag{1}$$

$$s.t. \sum_{i \in \mathcal{C}} X_{ij} = 1 \qquad \forall j \in \mathcal{S} \tag{2}$$

$$\sum_{j \in \mathcal{S}} L_j X_{ij} \leq Z_i K_i \qquad \forall i \in \mathcal{C} \tag{3}$$

$$X_{ij}, Z_i \in \{0, 1\} \tag{4}$$

where

S is the set of customer sites (hosts and terminal controllers)

C is the set of concentrators

$X_{ij} = \begin{cases} 1 & \text{if concentrator } i \text{ serves site } j \\ 0 & \text{otherwise} \end{cases}$

$Z_i = \begin{cases} 1 & \text{if concentrator } i \text{ is operated} \\ 0 & \text{otherwise} \end{cases}$

C_{ij}: Cost of using link (i, j)

F_i: Set-up cost of locating and operating concentrator i

L_j: Load of customer site j

K_i: Bandwidth of concentrator i

2.1 Problem Conceptualization

Consider a fragment of the data model for example 1 shown in Figure 1. The information about the problem contained in this model can be explained informally as follows (formal definitions of the italicised terms are given in §3). The nodes "Concentrators," "Sites," and "Terminals" represent the set of concentrators, customer sites, and individual terminals, respectively. The nodes "Owned-links" and "Leased-links" represent the set of owned and leased links that exist between the concentrators and sites. Each of these link nodes is an *aggregation* (a Cartesian product) of concentrators and customer sites. The node "Links," which represents all the available links between concentrators and sites, is a *generalization* of these two link nodes. The node "Serves" is a *specialization* of links, and represents those links that are operated to serve sites from concentrators. A "Network-element" is a generalization of concentrators and sites. "Host-computers" and "Controllers" are specializations of sites. The node "Clusters" is a *grouping-of* terminals, and each cluster controls a terminal controller.

This fragment of the data model is useful in conceptualizing the original problem since it captures, explicitly and directly, essential information about the problem. Of course, we must note that certain of the information captured in the data model could also be represented in mathematical modeling languages. For example, the existence of network elements, customer sites and concentrators, and of the linkage between them, is represented in AMPL as:

- *set* $\mathcal{N}$; set of network elements.

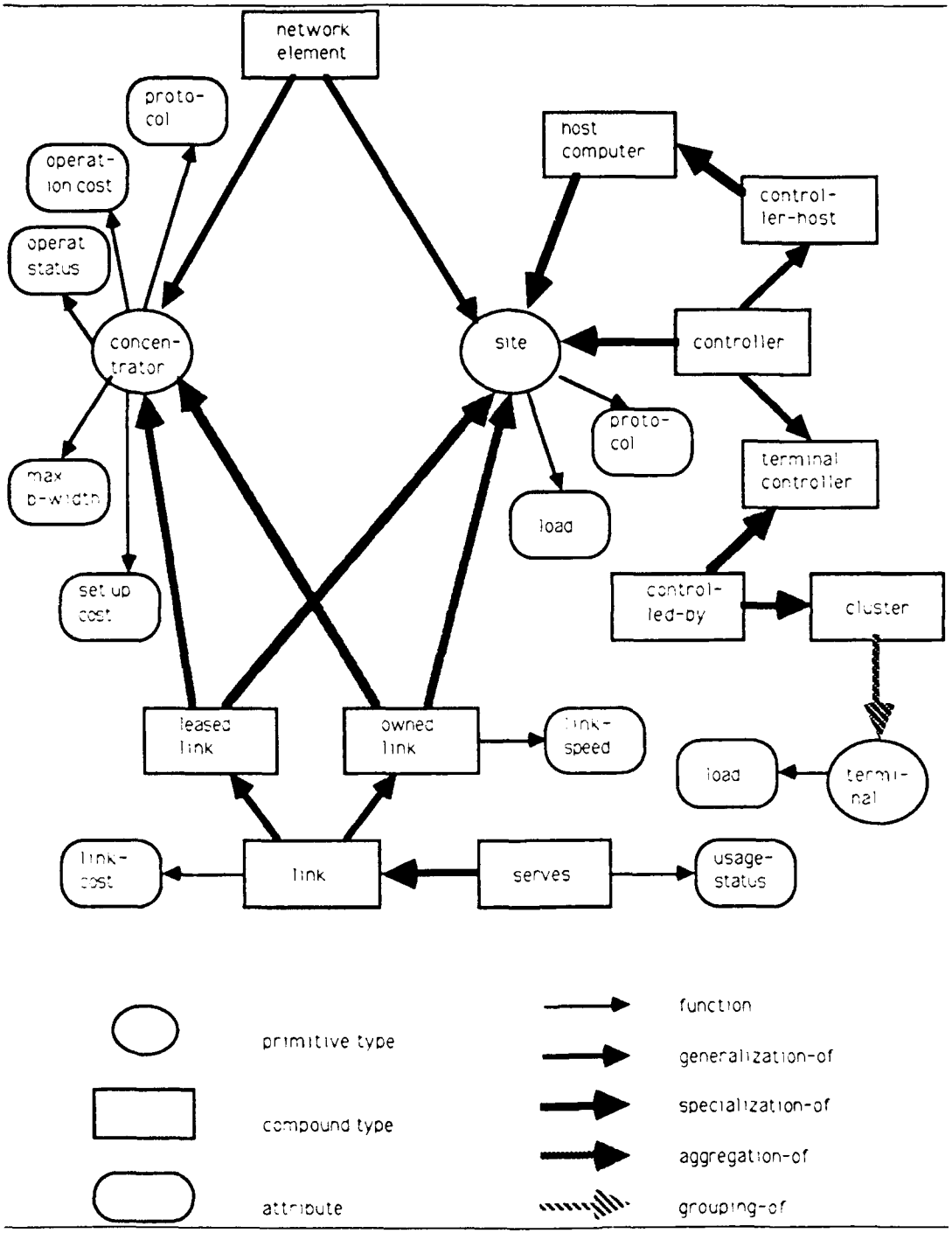


Figure 1: *Communications network design* data model

- *set* $\mathcal{C}$; set of concentrators.
- *set* $\mathcal{S}$; set of customer sites.
- *set* $\mathcal{T}$; set of terminal controllers.
- *set* $\mathcal{H}$; set of host computers.
- *set* $\mathcal{O}$; set of available owned links between concentrators and customer sites.
- *set* $\mathcal{M}$; set of available leased links between concentrators and customer sites.

However, since the mathematical formulation deals with only customer sites and concentrators, it is unlikely that $\mathcal{N}$, $\mathcal{T}$, $\mathcal{H}$, $\mathcal{O}$, and $\mathcal{M}$ would be mentioned at all in the representation. While EMLs provide an excellent representation of the information necessary for *solving* a given mathematical formulation for a problem, they fall short in capturing other information relevant to the original problem. We illustrate this by considering a few other aspects of the problem.

1. Both concentrators and customer sites are network elements. (A network element is a *generalization* of concentrators and customer sites.)
2. There are two kinds of customer sites, hosts and terminal controllers. These two kinds are not mutually exclusive, since a host computer can also be a terminal controller. (Host computers and terminal controllers are both a *specialization* of customer sites.)
3. There is a link-cost associated with both owned links and leased links between concentrators and customer sites. There is also a link-speed associated with each owned link between concentrators and sites.
4. A cluster is a group of terminals.

These specific items of information about the problem are captured adequately using constructs of a data modeling language (see Figure 1), but can

not be represented adequately in existing mathematical modeling languages, as discussed below.[4]

First, note that both the generalization and specialization relationships (items 1 and 2 above) are represented in EMLs, such as AMPL or GAMS, using the same set-theoretic operator:

- $\mathcal{N} = \mathcal{C} \cup \mathcal{S}$, and
- $\mathcal{H} \subseteq \mathcal{S}, \mathcal{T} \subseteq \mathcal{S}, \mathcal{S} = \mathcal{H} \cup \mathcal{T}$.

Due to this semantic overloading of the set union operator, the qualitative distinction between these two kinds of relationships is lost. Second, the attributes of the relationships between objects (e.g., *link-speed, link-cost*—item 3 above) can be represented in EMLs only by using indexed variables (where the index sets represent the objects), whereas data modeling languages directly represent these as attributes of the relationships. In the data model, *link-speed* is represented as an attribute of the relationship *owned-links* between *concentrators* and *sites*, and *link-cost* is an attribute of *links* (see Figure 1). In the EML representation, however, the indexed variables C_{ij} do not convey information as to *which* relationship—owned-link or leased-link—they are attributes of. Third, the grouping relationship (item 4) has no adequate counterpart in mathematical modeling languages [25].

In general, the evolution of semantic data modeling languages has been guided by the need to provide constructs for the direct and explicit representation of structural relationships between objects. Thus, the inclusion of data modeling constructs in languages for mathematical modeling should facilitate problem conceptualization. For example, in a successful application of management science techniques to the scheduling of the 1992 Olympic games, Andreu and Corominas begin by developing an entity-relationship data model of the problem [3]. Fourer argued that an algebraic representation of the *mathematical formulation* reduces problems of verification, modification, and documentation, is more readable and understandable, makes use of powerful abstractions commonly used by modelers, and is independent of particular algorithms [21]. In

[4] It is not surprising that this is the case, since such information is not required to solve the model, and since EMLs primarily aim to provide an alternative (to matrix generators) representation that can be transformed to that required by a solver.

a similar way, a representation of the structural and qualitative relationships, using data modeling constructs, facilitates verification, modification, and documentation of the *problem*, and is independent of a particular mathematical formulation of it. This, we shall see, is a significant consideration in model revision and version management.

2.2 Ensuring Integrity of Data

Consider, in example 1, the statement *An existing link is "valid" if and only if the customer site and concentrator it connects are compatible, i.e., they support the same protocol.* It essentially states that the problem data on available links should include no pair (i, j) such that customer site i and concentrator j are not compatible. In other words, the sets of owned links and leased links should both be subsets of the set of elements (i, j) where i and j have the same protocol. For any invalid (i, j) pair, the cost of using that link is assumed to be infinite. This information is represented in a data modeling language as integrity constraints, and is enforced via statements equivalent to the expressions below.

$$\mathcal{O} \cup \mathcal{M} \subseteq \{\langle i, j\rangle : i \in \mathcal{C}, j \in \mathcal{S}, \text{protocol}(i) = \text{protocol}(j)\} \tag{5}$$

$$(\langle i, j\rangle \notin \mathcal{O} \cup \mathcal{M}) \rightarrow (\text{link-cost}(\langle i, j\rangle) = \infty) \tag{6}$$

Note that, viewed by the user of an EML, this is a constraint on the "input" data for the model, and is thus a pre-processing problem (unlike the constraint *Each site must be served by exactly one concentrator* which is a constraint on the solution). From a data modeling viewpoint, however, this is simply an integrity constraint on the data—it might constrain the inputs for one model, and the solution of another. In general, data modeling languages and database systems emphasize features for ensuring integrity of data. This (i.e., for input data) is typically not considered a function of a mathematical modeling language—it is assumed that the integrity of problem-specific data has been ensured externally.[5]

[5]Recent extensions to AMPL do allow modelers to declare a "check" statement to ensure integrity.

2.3 Model Formulation

Consider the modeling variables in the network design problem of example 1. Each variable used in the mathematical programming formulation represents a problem-specific concept. For instance, the variable X_{ij} denotes the existence (or lack of it) of a link between concentrator i and customer site j. In the data model for this problem, the aggregation node *links* represents the concept of concentrator–site linkage, which concept causes the inclusion of the variable X_{ij} in the mathematical formulation. Similarly, consider the constraint $\sum_i X_{ij} \leq 1$. This constraint is derived from the statement *Each site must be served by exactly one concentrator, though the same concentrator may serve various sites* in the problem description. In the data model, this statement is represented in the functional dependency between sites and concentrators:

$$sites \xrightarrow{serves} concentrators. \tag{7}$$

This component of the data model serves to justify the presence, and specific form, of this constraint in the mathematical model. This is depicted in Figure 2 which shows the relevant fragment of the justification network for this model. In general, in formulating a model, one identifies the modeling variables and specifies the relationships between them. Each of these components is introduced by the modeler to represent some aspect of the problem being modeled. Previous studies in model formulation [39, 40] have found that expert modelers explain their formulations by relating components of the model to the objects and relationships in the problem statement. Since the data model is a qualitative representation of a problem, components of the mathematical model can be *justified* in terms of some element(s) of the data model. Thus, the information formalized in the data model serves two related purposes in the formulation of the mathematical model. First, this information is useful in the creative part of model formulation, such as in introducing a new variable or a new constraint. Second, this information is useful in justifying components of the mathematical model, and serves as useful active documentation of the same.

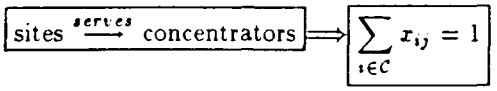


Figure 2: Fragment of the justification network

2.4 Model Reformulation and Version Management

Now consider the following modification to the problem being modeled. In the original statement, we considered a scenario in which "all link usage costs [were] constant, irrespective of the type of link." Consider, now, a scenario in which the link usage costs are measured more accurately. First, we must distinguish between the cost of using owned links (call it C'_{ij}) and the cost of using leased links (call it C''_{ij}). Second, the C'_{ij}'s should be computed as some function g of the link speed, and the C''_{ij}'s should be estimated using a model, which specifies a non-linear functional relationship between the cost and volume of traffic on a link:

$$C'_{ij} = g(S_j) \tag{8}$$

$$C''_{ij} = f(L_j, \sigma^2(L_j)) \tag{9}$$

where S_j is the speed of link j, L_j is the average load on host j, $\sigma(L_j)$ is the standard deviation of the load on host j, and f is a specified non-linear function. The cost function C_{ij} can now be written as

$$C_{ij} = \begin{cases} g(S_j) & \text{if the link (i,j) is owned} \\ f(L_j, \sigma^2(L_j)) & \text{if the link (i,j) is leased} \end{cases} \tag{10}$$

These modifications result effectively in a new mathematical formulation, with a non-linear objective function. However, note that the underlying data model is still the same, and that the new expressions in the formulation can still be justified by components of the original data model. In fact these modifications are the consequences of considering attributes, which were previously ignored, already present in the data model.

For a given problem, several models may be formulated and explored in the course of model development. We refer to each distinct model that results from such formulations as a *version*. Each version is the result of making certain assumptions about the problem. In the current example, we replaced the assumption about constant link usage costs for all links with one where there were different costs for leased and owned links. The effect of the new assumption was to replace the total usage cost expression $\sum_{i \in C} \sum_{j \in S} C_{ij} X_{ij}$ with a new total usage cost expression (expressions 8–10). When model formulations are large or complex, it is a non-trivial task to identify and replace model components that are affected, directly or indirectly, by changes in assumptions. However, this can be facilitated by examining the justifications, in terms of elements of the data model, for each component of the mathematical model. A model component that does not have at least one justification can be retracted as it lacks support in the version being created. example? maybe

Further, comparing the original and new formulation for the communications network problem, we see that the models share a great deal of structure: apart from their objective functions they are the same models. One plausible approach to reformulation is to start with the original model and its underlying justifications, and to selectively make the changes required to create a new version. The original data model (or, in general, a substantial part of it) can be re-used to justify and document the new formulation. Thus, the data model and the associated justifications support this kind of version creation, by promoting re-use of previous modeling effort.

When several model versions are created, one may lose track of the similarities and differences betwen versions. Software tools are available for mitigating this problem. For instance, if each version is stored in a file, an operating system utility (such as *dif* in the UNIX system) can be used to determine differences in terms of lines present in one file that are absent in another. In the network design example, *dif* may be used to determine that the objective function in one version is $\sum_{i \in C} \sum_{j \in S} C_{ij} X_{ij}$ while the objective function in another is $\sum_{i \in C} \sum_{j \in S} g(S_j) X_{ij} + f(L_j, \sigma^2(Lj))$. However, such utilties lack means to provide reasons as to why these objective functions are different. The data model and associated justifications can be used to do that and much more, as shown

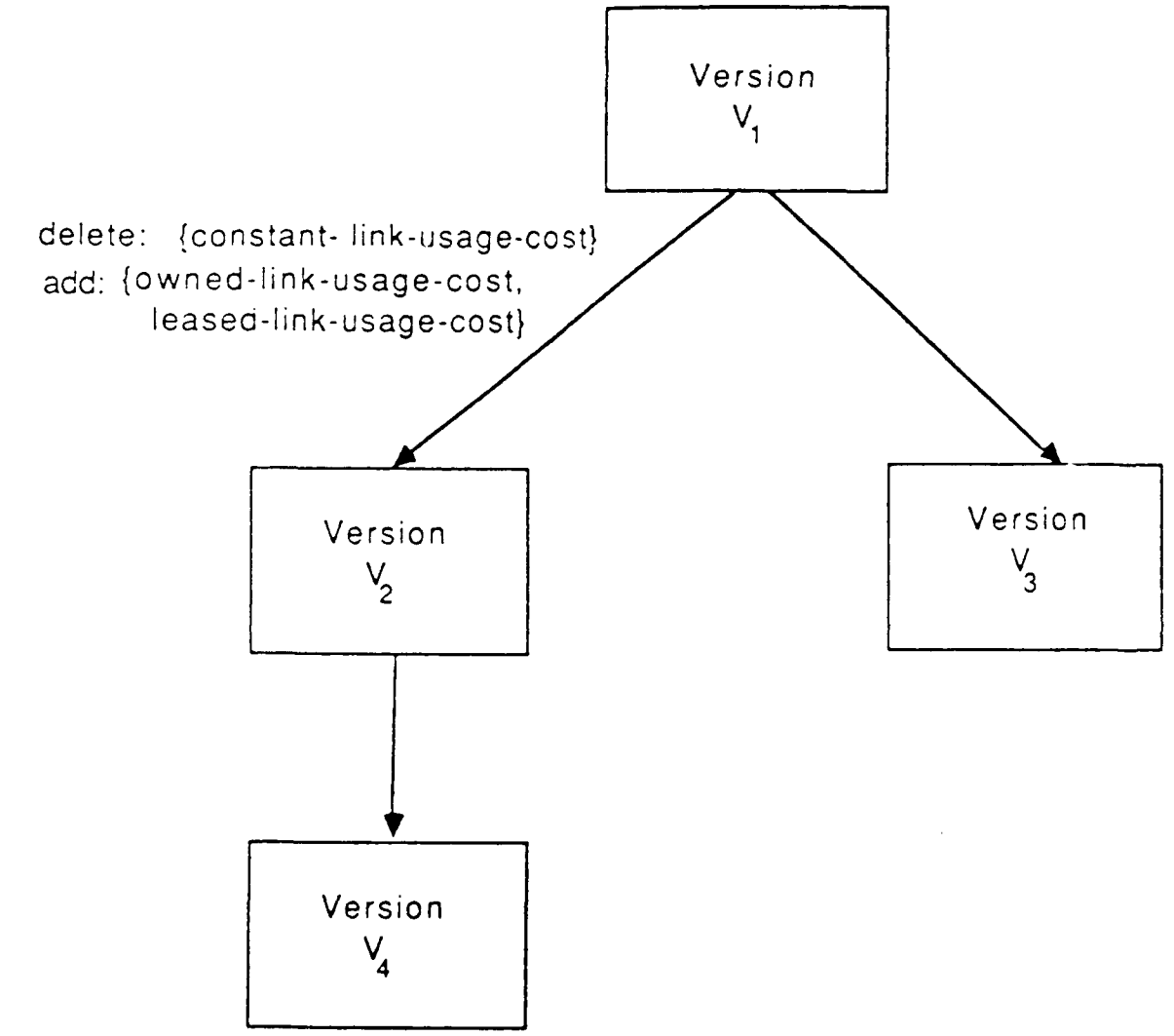


Figure 3: *Fragment of version graph: Communications network design* data model

below.

Consider the version graph for the communications network problem. Each node in this graph represents a model version for the problem. A directed arc from a node V_1 to node V_2 indicates that version V_2 was obtained by modifying version V_1. The labels on the arcs represent the assumptions that were deleted and / or added in order to move from V_1 to V_2. A fragment of this graph is shown in Figure 3. Thus, this graph captures the sequence in which various versions were developed, the differences between versions, as well as the reasons for developing new model versions. It documents the chronology of the model development process, and can be a useful repository of modeling experience. Complex models can be hard to solve, and successful modelers often evolve solution strategies by developing several versions [37]. They solve relaxed models, selectively increase the complexity by introducing new assumptions, and use the solutions from the relaxed model to solve the new model version. For a given problem, clues about a useful solution strategy can be obtained by examining the chronology of successful model development processes for related models.

2.5 Discussion

We have illustrated several benefits that would result from integrating features of data modeling languages into mathematical modeling languages. Put together, we believe they make a convincing case for providing such integration. In our view, one of the most significant implications of such integration is that it allows a modeler to document justifications, in terms of elements of the data model, for various components of a mathematical model. There are several benefits that follow from the availability of such justifications. These include improvements in a modeling system's ability to explain and communicate the model, to track changes to models, to explain similarities and differences between various versions of a model, and to examine consistency of a model formulation in terms of its justifications. Having made the case that such integration is desirable, how do we achieve it? We present our approach to the integration of data and mathematical modeling languages in the next section.

3 A Formalism for Language Integration

Our method for language integration is based on the embedded languages technique developed by Bhargava and Kimbrough [4]. The embedded languages technique provides a systematic means for integrating multiple "embedded" languages within a single "embedding" language. In our context, one embedded language is a generic data modeling language L_d, and the other embedded language is a generic mathematical language L_m. Following the convention of Bhargava and Kimbrough, the embedding language will be called $L^{\dagger}$.

3.1 A Generic Data Modeling Language L_d

We begin by formalising a generic semantic data modeling language, which we call L_d. The language supports data modeling constructs (aggregation, grouping, generalization, and specialization) as a means for the direct and explicit representation of structural relationships between data elements. Our development of the language is based on the set theoretic development described in [29]. The reader may find it useful to refer to Figures 1 and 4, where we represent pictorially and textually, respectively, the data model for the communications network model (example 1).

In a data modeling language, the real-world is conceptualized as a collection of objects and relationships between these objects. An *object type* is a set of objects. For any object type A we will denote the set of objects that it represents by $\dot{A}$. The specification of an object type may be either *primitive* or *compound*. A semantic data modeling language provides a set of abstraction relationships that are used to specify compound object types in terms of other object types, as well as to capture relationships between object types.

Definition 1 *Primitive Specification of an Object Type*

> A primitive specification of an object type consists of a definition for the object type as a set of objects drawn from a collection of known domains. The commonly occurring domains are the sets of reals, integers, boolean, and strings. Other domains, subsets of these, may be defined by users of the language.

```
## PRIMITIVE OBJECT TYPES ##
object-type(terminals);              # individual terminals
object-type(sites);                  # customer sites
object-type(controllers);            # terminal controllers

## COMPOUND OBJECT TYPES (SEMANTIC RELATIONSHIPS) ##
aggregation-of([concentrators, sites],owned-links);          #
owned-links
aggregation-of([concentrators, sites],leased-links);         #
leased-links
aggregation-of([clusters, controllers],controlled-by);       #
controllers of clusters
group-of(terminals,clusters);                                #
clusters
generalization-of([sites, concentrators],network-elements);  #
network-elements
specialization(owned-links, links);
#links
specialization(leased-links,links);                          #
links
specialization-of(sites,hosts);                              #
host computers
specialization-of(sites,controllers);                        #
concentrators
specialization-of(links,serves);                             #
serves

## FUNCTIONS (ATTRIBUTES) ##
range-of(setup-cost(concentrators),reals)
range-of(operation-cost(concentrators),reals)
range-of(max-bandwith(concentrators),integers)
range-of(link-speed(owned-links),reals)
```

Figure 4: Data model for *Communications network design* expressed in L_d

Example 2 *Primitive Object Type: Host Computers*

The collection of host computers, labelled *hosts*, has a primitive specification in our example. The object type *hosts* is defined as a collection of objects that denote specific host computers, say *host-1*, ..., *host-n*. The terms *host-1*, ..., *host-n* are drawn from the domain of strings. In L_d this specification is achieved by statements such as the following:

```
object(host-1)
object-type(hosts)
element-of(host-1, hosts)
```

which represent that *hosts* is an object-type and the object *host-1* belongs to that type.

Definition 2 *Compound Specification of an Object Type*

A compound specification of an object type consists of a definition of that object type in terms of other object types and one of the following abstraction relationships: aggregation, generalization, specialization, and grouping.

Each of these abstraction relationships and its use in object specification is discussed below.

Definition 3 *Aggregation*

The aggregation of a set of object types $A_1, \ldots, A_n$ is an object type A such that the set of objects represented by A is a subset of the Cartesian product of the sets of objects represented by $A_1, \ldots, A_n$, i.e.,

$$\hat{A} \subseteq \bigotimes_{i=1}^{n} \hat{A}_i$$

Example 3 *Compound Specification: Aggregation*

The object types *owned-links* and *leased-links* which represent connections between concentrators and customer sites (see example 1) are both aggregations of *concentrators* and *sites*. These are a collection of 2-tuples of the form $\langle c, s \rangle$ where c is a concentrator and s is a customer site. This information is represented in the data model by the aggregation nodes *owned-link* and *leased-link*, and is specified in the language as follows:

```
aggregation-of([concentrators, sites],owned-links)
aggregation-of([concentrators, sites],leased-links)
element-of(<concentrator-1,host-1>, owned-links)
element-of(<concentrator-2,controller-3>, leased-links)
```

These statements represent that *owned-links* and *leased-links* are object types formed by aggregating *concentrators* and *sites*, that a specific owned link is the link between *concentrator-1* and *host-1*, and that a specific leased link is the link between *concentrator-2* and *controller-3*.

Definition 4 *Grouping*

The grouping over an object type A is an object type B such that the set of objects represented by B is a power set of the set of objects represented by A. Thus, any subset of $\hat{A}$ is an object of type B, and

$$\hat{B} = \{S : S \subseteq \hat{A}\}$$

Example 4 *Compound Specification: Grouping*

A terminal controller unit controls a cluster, i.e., a collection, of terminals. A particular cluster is some subset of the set of objects of type *terminals*. The object type *clusters* is represented in a data model as a grouping of *terminals*, and is specified in the language as follows:

```
group-of(terminals,clusters)
```

```
element-of({terminal-1, terminal-2, terminal-3}, clusters)
element-of({terminal-5,terminal-4}, clusters)
```

Definition 5 *Specialization*

A specialization of an object type A is an object type B such that the set of objects represented by B is a subset of the set of objects represented by A, i.e., $\hat{B} \subset \hat{A}$. The objects of type B inherit the structure of the objects of type A. There can be several specializations of an object type, and these specializations are not required to be disjoint.

Example 5 *Compound Specification: Specialization*

In example 1, *host computers and clusters of terminals are equivalent, and are referred to as customer sites,* and a *host computer may also serve as a terminal controller.* The *hosts* and *controllers* are represented in a data model as specializations of the object type *sites*, and are specified in the language as follows:

```
specialization-of(sites,hosts)
specialization-of(sites,controllers)
element-of(host-1,hosts)
element-of(host-1,controllers)
element-of(concentrator-1,concentrators)
```

The specialization relationship allows the intersection of the sets *hosts* and *controllers* to be non-null, which is indeed the case here since *host-1* is both a host computer and a terminal controller.

Definition 6 *Generalization*

A generalization of a set of object types $A_1, \ldots, A_n$ is an object type A such that the set of objects represented by A contains all the objects represented by $A_1, \ldots, A_n$, i.e.,

$$\hat{A} = \bigcup_{i=1}^{n} \hat{A}_i$$

The sets $A_1, \ldots, A_n$ are assumed to be pair-wise disjoint.

Example 6 *Compound Specification: Generalization*

In example 1, a network element is either a concentrator or a host computer. In the data model, the object type *network-elements* is a generalization of the object types *concentrators* and *hosts*, and is specified in the language as follows:

```
generalization-of([sites, concentrators],network-elements)
element-of(host-5,network-elements)
element-of(concentrator-1,network-elements)
```

The abstractions discussed above are useful in capturing some of the commonly occurring data structures and relationships between data. The constructs *specialization* and *generalization* capture the "hierarchical" relationships among objects in the problem domain, while *aggregation* and *grouping* capture the "horizontal" relationships. In addition, functional relationships among object types are represented as *attributes* of the object types in the data model. For example, the cost of operating concentrators is represented as an attribute *operation-cost* of concentrators.

To summarize, L_d is a specialized first-order language with a) an open vocabulary of individual constants representing objects and object types in the data model, b) an open vocabulary of function constants representing attributes of objects or object types, and c) the following special predicate constants (in the notation below, A, $A_1, \ldots, A_n$, and B denote individual constants):

- **object**: A unary predicate, such that the statement object(A) asserts that A is an object,
- **object-type**: A unary predicate, such that the statement object-type(A) asserts that A is an object type,
- **element-of**: A binary predicate, such that element-of(A, B) asserts that the object A is an element of the object type B,[6]

[6] Here, and elsewhere, it is not necessary that the predicate be defined extensionally in L_d. For example, the objects belonging to a certain object type might be declared by pointing, using a query language, to a column in a database that stores the objects of that type.

- **aggregation-of**: A binary predicate, such that aggregation-of($[A_1, \ldots, A_n]$,B) asserts that the object type B is an aggregation of the object types $A_1, \ldots, A_n$,

- **grouping-of**: A binary predicate, such that grouping-of(A, B) asserts that the object type B is formed as a grouping over object type A,

- **specialization-of**: A binary predicate, such that specialization-of(A, B) asserts that the object type B is a specialization of object type A,

- **generalization-of**: A binary predicate, such that generalization-of($[A_1, \ldots, A_n]$,B) asserts that the object type B is a generalization of the object types $A_1, \ldots, A_n$, subtype(A,B) declares that the primitive

- **function-range**: A binary predicate, such that function-range(f(A),B) represents that the range of function f with domain A is B.

As a simple example, if L_d were to be a relational data modeling language, the names of the relation schemes would be object types in L_d, and the columns of these relations would be function constants in L_d.

3.2 A Generic Mathematical Modeling Language L_m

For our purposes in this paper, any of the existing executable algebraic modeling languages (such as AMPL, GAMS, $L_\downarrow$, LINGO) could serve as the mathematical modeling language L_m. While there are some differences between these languages, they are very similar in the basic structure and in the characteristics that we are concerned with in this paper. Hence instead of specifying a new language, or of illustrating our ideas on a specific language, we will assume a generic modeling langauge. For details on any of these particular languages, we refer the reader to the appropriate references mentioned in §1. An executable modeling language based on first-order logic is discussed in [6]. Here, we restrict ourselves to a simple illustration of this language, by representing the model of example 1 in an AMPL-like syntax (see Figure 5) .

```
### SETS ###

set C;                  # Concentrators
set S;                  # customer Sites (hosts and controllers)

### PARAMETERS ###

param cost {C,S} > 0;  # cost(i,j) of using link between
concentrator i and site j
param fcost {C} > 0;   # fixed setup cost(i) of locating and
operating concentrator i
param load {S} >= 0;   # load(j) at customer site j
param k {C} > 0;       # bandwidth(i) of concentrator i

### VARIABLES ###

var x {C,S} binary;    # x(i,j) = 1 if concentrator i serves site
j, 0 otherwise
var z {i} binary;      # z(i) = 1 if concentrator i is operated, 0
otherwise

### OBJECTIVE FUNCTION ###

minimize total cost:
      sum {i in C}(sum {j in S} (cost[i,j] * x[i,j]) + fcost[i] *
z[i]);

### CONSTRAINTS ###

subject to linkages {j in S}: sum {i in C} (x[i,j]) = 1;
                        # each site must be served by exactly 1
concentrator
subject to capacity {i in C}: sum {j in S} (load[j] * x[i,j]) <=
z[i] * k[i];
                        # each concentrator (if open) has a
bandwidth capacity
```

Figure 5: *Communications network design* model expressed in L_m

3.3 Integrating L_d and L_m in $L^{\uparrow}$

In [4], Bhargava and Kimbrough developed the embedded languages technique and explained how an algebraic modeling language (a specialized first-order logic language, $L_{\downarrow}$) is embedded in $L^{\uparrow}$. The embedded languages technique provides a systematic and rigorous means for integrating, and reasoning about, multiple (embedded) languages. In that framework, an embedded language ($L_{\downarrow}$)—which models a semi-formal target language ($L^{*}_{\downarrow}$)—is embedded within an embedding language, ($L^{\uparrow}$), which is used to represent information about formulas and terms in the embedded language, and to translate one embedded language into another.

Central to the embedded languages technique is the idea of an image function $\mathcal{I}$, and a translation function $\mathcal{F}$. An embedding is a triple $\langle \mathcal{I}, \mathcal{F}, \Delta \rangle$, where Δ is a collection of formulas, in $L^{\uparrow}$, that represents the rules of inference and transformation of $L_{\downarrow}$. The image function $\mathcal{I}$ uniquely maps all expressions—terms and formulas—in $L_{\downarrow}$ into terms in $L^{\uparrow}$. The translation function $\mathcal{F}$ uniquely maps the images of all formulas in $L_{\downarrow}$ into formulas in $L^{\uparrow}$. Therefore, in order to embed L_d and L_m in $L^{\uparrow}$, we require functions $\mathcal{I}$ and $\mathcal{F}$ such that a) the well-formed formulas as well as terms of L_d and L_m be interpretable as terms in $L^{\uparrow}$, and that b) there be a formula in $L^{\uparrow}$ corresponding to, and making an assertion regarding, each formula ϕ in L_d.

Bhargava and Kimbrough discussed the image and translation functions in detail in the context of embedding an algebraic modeling language. The functions are developed along similar lines for embedding a data modeling language. In what follows, we focus on the predicates that are required to relate information across the two embedded languages. Of these, the predicates used to represent justification networks and the predicates that declare the belief status of the components that are the nodes of the networks are formalizations of the work reported in [40].

To begin with, assume that there are predicates **wff-L_d** and **wff-L_m** in $L^{\uparrow}$ with the following interpretation:

- **wff-L_m**$(\mathcal{I}(\phi))$ states that ϕ is a wff in L_m
- **wff-L_d**$(\mathcal{I}(\psi))$ states that ϕ is a wff in L_d

data model is not mapped to any variable in the mathematical model, that might suggest that a new variable needs to be introduced.

4.2 Ensuring Integrity of Data

Consider, again, the statement *Each site must be served by exactly one concentrator*. It implies that for any objects i_1, i_2, and j, if $\langle i_1, j\rangle$ and $\langle i_2, j\rangle$ are both objects of type *serves* and if i_1 and i_2 are distinct, then there is a problem with the data, i.e.,

$$\begin{aligned} &(\langle i_1, j\rangle \in \text{serves}) \wedge (\langle i_2, j\rangle \in \text{serves}) \wedge (i_1 \neq i_2) \\ &\wedge (i_1 \in \text{concentrators}) \wedge (i_2 \in \text{concentrators}) \wedge (j \in \text{sites}) \\ \rightarrow \quad &\text{not-ok}((\langle i_1, j\rangle \in \text{serves AND } \langle i_2, j\rangle \in \text{serves})) \end{aligned} \tag{11}$$

This is represented by the following $L^{\uparrow}$ formula:

$$\begin{aligned} &\text{wff-}L_d(\mathcal{I}(\text{element-of}(\langle i_1, j\rangle, \text{serves}))) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(\langle i_2, j\rangle, \text{serves}))) \\ &\wedge (\mathcal{I}(i_1) \neq \mathcal{I}(i_2)) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(i_1, \text{concentrators}))) \\ &\wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(i_2, \text{concentrators}))) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(j, \text{sites}))) \\ &\rightarrow \text{not-ok}(\mathcal{I}(\text{element-of}(\langle i_1, j\rangle, \text{serves})) \text{ AND } \mathcal{I}(\text{element-of}(\langle i_2, j\rangle, \text{serves}))) \end{aligned} \tag{12}$$

The ability to make such statements means that constraints that are normally not part of the mathematical formulation can now be included in the model representation. In fact, statements of this sort can be derived from a more general $L^{\uparrow}$ formula (as explained below), which means that such constraints can be enforced simply by declaration of the functional dependencies in the data model.

Let us introduce in $L^{\uparrow}$ a predicate **f-d** such that the formula $\textbf{f-d}(\mathcal{I}(\phi_1), \mathcal{I}(\psi), \mathcal{I}(\phi_2))$ means that there is a functional dependency of the form $\phi_1 \xrightarrow{\psi} \phi_2$ in the data model. This dependency is meant to ensure that elements

$\langle i_1, j\rangle$ and $\langle i_2, j\rangle$ can both belong to ψ only if i_1 is equal to i_2. This rule is stated in $L^{\uparrow}$ as indicated below.

$$\forall\phi_1\forall\phi_2\forall\psi \quad (\text{f-d}(\mathcal{I}(\phi_1),\mathcal{I}(\psi),\mathcal{I}(\phi_2)) \rightarrow$$
$$\forall i_1\forall i_2\forall j(\text{wff-}L_d(\mathcal{I}(\text{element-of}(\langle i_1,j\rangle,\psi))) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(\langle i_2,j\rangle,\psi)))$$
$$\wedge(\mathcal{I}(i_1) \neq \mathcal{I}(i_2)) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(i_1,\phi_1)))$$
$$\wedge\text{wff-}L_d(\mathcal{I}(\text{element-of}(i_2,\phi_1))) \wedge \text{wff-}L_d(\mathcal{I}(\text{element-of}(j,\phi_2)))$$
$$\rightarrow \text{not-ok}(\mathcal{I}(\text{element-of}(\langle i_1,j\rangle,\psi) \text{ AND element-of}(\langle i_2,j\rangle,\psi))))) \qquad (13)$$

4.3 Model Formulation

Consider the example in §2.3 illustrating the relationship between the problem statement and its mathematical model. The statement "Each site must be served by exactly one concentrator, though the same concentrator may serve various sites" *justifies* the constraint $\sum_i X_{ij} \leq 1$. It is stated in L^I as shown below.

$$\text{justifies}(\mathcal{I}(\text{sites} \xrightarrow{\text{serves}} \text{concentrators}),\mathcal{I}(\sum_i X_{ij} \leq 1),v1) \qquad (14)$$

The same component may be justified in different ways. These distinct justifications are referred to as disjunctive justifications. When several components (elements of the data or the mathematical model) jointly justify another component, such a justification is a conjunctive justification. An example of such a justification is that of the expression $\sum_{j\in S} L_j X_{ij} \leq Z_i K_i$ by the variables X_{ij}, Z_i, K_i, and L_j. This can be stated in L^I using *list []* notation to indicate conjunction.

$$\text{justifies}([\mathcal{I}(X_{ij}),\mathcal{I}(Z_i),\mathcal{I}(K_i),\mathcal{I}(L_j)],\mathcal{I}(\sum_{j\in S} L_j X_{ij} \leq Z_i K_i),v1) \qquad (15)$$

There are several benefits that can accrue from explicitly declaring justifications as a part of the model representation.

- The justification network which is the set of all justfications associated with a formulation can serve as an active documentation of the model. It can be queried, and thereby can promote model understandability. The

queries can not only extract information explicitly asserted but also infer *chains* of justfications.

- Model formulation can suffer from flawed reasoning. The justfication network can help detect such flaws. A cycle in a chain of justifications associated with a component indicates such a flaw. Cycle detection algorithms [2] can be specified in the $L^{\dagger}$ language.

4.4 Model Reformulation and Version Management

The justfication networks, with some extensions, can help with model reformulation and in version management. This section will discuss the nature of these extensions, and how they can fruitfully assist in reformulation and version management.

Model reformulation entails changes to components. All components directly or indirectly (i.e., on a justification path) justified by a changed component are affected. For example, consider the assumption that link usage costs depend *only on the (i,j) pair they are an attribute of*. Now suppose we replace this with an assumption that they are also a function of link type. This implies that in the new formulation C_{ij} is removed, as is the old objective function it (jointly) justifies. It is replaced by new cost functions for leased and owned links, and a new objective function as discussed in §2.4. If these changes were made within the context of a single formulation, then the justifications could simply be altered to reflect the new assumptions. However, what should be done if the modeler chooses to retain the original model, and simply investigate these changes in the context of a new version? Now, in addition to declaring the justification relationships, we need a mechanism to declare if the model components used in the justification relationships are, or are not, believed in a given version. In our example, the original assumption about link usage cost is believed (declared to be *IN*) in the original model but not believed (declared to be *OUT*) in the new version. These declarations of belief are stated as shown below.

$$\text{in-label}(\mathcal{I}(\text{depends-only-on}(C_{ij},(i,j))),v1) \tag{16}$$

$$\text{out-label}(\mathcal{I}(\text{depends-only-on}(C_{ij},(i,j))),v2) \quad (17)$$

The first assertion states that the belief that C_{ij} depends only on the (i,j) pair and not link type (i.e., the constant link usage cost assumption) is IN in version v1, the label used to refer to the original formulation. The second assertion states that this belief is OUT in the new version labelled v2.

With these representations (i.e., justification networks and belief status) in place, specific functions can be defined in $L^{\uparrow}$ to specifically support model reformulation and version management. We give two specific examples: a) a function that propagates a change in belief status of a model component (node) in the justification network, and b) a function that computes similarities and differences between versions. These have been adpated from [39, 40] where a fuller discussion of functions that can be used to support reformulation and version management can be found.

When the belief status of a model component changes, the belief status of all the components justified directly or indirectly by it can change. Two functions, in-propagate and out-propagate, are specfied in$L^{\uparrow}$ to manage these changes in belief status.

```
function in-propagate(Component,Ver);
  if out-label(I(Component),V) and Ver ∈ V
  then in-label(I(Component),Ver)
  endif
  ∀C justifies(I(Component),I(C),Ver)
  do in-propagate(C,Ver)
end function in-propagate
```

When the belief status of component is changed to IN, the belief status of all the model components that are directly or indirectly justified by this component are also changed to IN. The justification network is used to identify the components whose belief status should be changed.

```
function out-propagate(Component,Ver);
  if in-label(I(Component),Ver)
  then out-label(I(Component),Ver.V)
  endif
  ∀C justifies(I(Component),I(C),Ver)
  if ¬∃K justifies(I(K),I(C))
  then do out-propagate(C,Ver)
  endif
end function out-propagate
```

When the belief status of a component is changed to OUT, the belief status of all components *solely* justified by it are also changed to OUT.

Over the course of a modeling project, several model versions may be constructed. Some of these versions will share components. For instance, different link usage cost assumptions resulted in two distinct versions with similar constraint structures but different objective functions. The ability to enquire about the similarities and differences between versions can be very useful when a modeler works with multiple versions. The function *determine-justification* supports this feature.

```
function determine-justification(Component,Ver);
  if primitive-node(I(Component),Ver)
  then return Component
  else let S be the list of components such that ∀E ∈ S
  justifies(I(E),I(Component),Ver) and in-label(I(E),Ver)
  recursive-determine-justifications(S,Ver)
end function determine-justifications
function recursive-determine-justification(Set,Ver);
  return append(determine-justification(first(Set),Ver),
  recursive-determine-justification(rest(Set),ver)
end function recursive-determine-function
```

The functions determine the justification chain associated with a given model component in a version. This function can now be used to determine similarities and differences. Consider the variable C_{ij}, which represents the link usage cost, in version v1 and v2. This variable is a component whose belief status is IN in both versions. However, it has a different set of justifications in v1 (i.e., that it depends only on the (i,j) pair it is an attribute of) and v2 (i.e., that it also depends on link type). So here we the same component that is present

in two different versions being justified in different ways. Using the *determine-justifications* function, the above mentioned justifications can be determined. Upon comparison, the similarities and differences can be uncovered.

5 Conclusions

In this paper we began with the proposition that there is much to gain by making data modeling language constructs available to modelers using an algebriac modeling language. We have discussed one way of achieving this, namely via an embedded languages approach, and have applied it to define several useful modeling support functions.

Specifically, we discussed the use of justifications to document the reasons for a particular mathematical formulation in terms of components of the data model. The advantages of such justifications need to be investigated further. For example, we believe that they can play an useful role in model integration. When models are integrated manually, modelers use information about the assumptions underlying components of the models being integrated. If two models with conflicting assumptions are being integrated, these conflicts must be identfied and resolved by the modeler. Systematic support for tasks such as this can be provided using justification networks. Another potential advantage of our approach might be realized in the initial stages of model reuse. Model reuse begins with the identification of candidates for reuse [34]. When there are several candidates, in the absence of support, this can be hard. The additional information offered by the justifications could provide such support.

To conclude, we have laid out a systematic approach for combining the strengths of data and mathematical modeling languages. We have argued that this can significantly improve the functionality of model management systems and have demonstrated some of the benefits. Much more remains to be done, but we hope to have raised issues for debate and future research.

References

[1] Abiteboul, S., and R. Hull, "IFO : A Formal Semantic Database Model," *ACM Transactions on Database Systems*, **12: 4**, December 1987.

[2] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Mass., 1974.

[3] Andreu, R., and A. Corominas, "SUCCESS92: A DSS for Scheduling the Olympic Games," *Interfaces*, **19: 5**, September-October 1989, pp. 1-12.

[4] Bhargava, H.K., and S.O. Kimbrough, "Model Management: An Embedded Languages Approach," forthcoming, *Decision Support Systems*, 1992.

[5] Bhargava, H.K., S.O. Kimbrough, and R. Krishnan, "Unique Names Violations, a Problem for Model Integration or You Say Tomato, I Say Tomahto," *ORSA Journal on Computing*, **3: 2**, Spring 1991, pp. 107-120.

[6] Bhargava, H.K., "A Logic Model for Model Management: An Embedded Languages Approach," Ph.D. thesis, University of Pennsylvania, Department of Decision Sciences, 1990.

[7] Bhargava, H.K., and R. Krishnan, "A Formal Approach for Model Formulation in a Model Management System," *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, Vol. III*, J. F. Nunamaker, Jr., ed., IEEE Computer Society Press, Los Alamitos California, January 1990 , pp. 453-462.

[8] Binbasioglu, M., and M. Jarke, "Domain-specific DSS Tools for Knowledge-based Model Building," *Decision Support Systems*, **2: 3**, 1986.

[9] Bisschop, J., and A. Meeraus, "On the Development of a General Algebraic Modeling System in a Strategic Planning Environment," *Mathematical Programming Study* **20**, 1982.

[10] Bonczek R.H., C.W. Holsapple, and A.B. Whinston, "Foundations of Decision Support Systems," *Academic Press*, 1981.

[11] Bradley, G.H., and R.D. Clemence, "Model Integration with a Typed Executable Modeling Language," *Proceedings of the Twenty-First Annual Hawaii International Conference on Systems Sciences,* 1988.

[12] Chari, S., and R. Krishnan, "Towards a Logical Reconstruction of Structured Modeling,", forthcoming in *Decision Support Systems*

[13] Chen P.P., "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Trans on Database Systems,* **1: 1,** 1976.

[14] Choobineh, J., "SQLMP: A Data SubLanguage for Representation and Solution of Linear Mathematical Models", Working Paper, Department of Business Analysis and Research, Texas A & M University, College Station, TX. 1990.

[15] Codd, E.F. , "A Relational Model for large shared data banks," *Communications. of the ACM* **13: 6,** June 1970.

[16] Dolk D.R. , "A Generalized Model Management System for Mathematical Programming," *ACM Transactions on Mathematical Software,* **12: 2,** June 1986.

[17] Dolk D.R., "Model Management and Structured Modeling : The Role of an Information Resource Dictionary System," *Communications of the ACM* 31: 6, June 1985 , pp. 704-718.

[18] Dutta A. and A. Basu, "An Artificial Intelligence Approach to Model Management in Decision Support Systems," *IEEE Trans. on Computers,* September 1984.

[19] Elam, J.J., J.C. Henderson, and L.W. Miller, "Model Management Systems: An Approach to Decision Support in Complex Organizations," *Proceedings of the International Conference in Information Systems,* 1980.

[20] Fourer, R., D. Gay, and B.W. Kernighan, "A Mathematical Programming Language," *Management Science,* **36: 5,** May 1990.

[21] Fourer, R., "Modeling Languages versus Matrix Generators for Linear Programming," *ACM Transactions on Mathematical Software*, **9: 2**, 1983 .

[22] Garg, P.K., "Abstraction Mechanisms in Hypertext," *Communications of the ACM.*, **31: 7**, July 1988.

[23] Geoffrion, A.M., "FW/SM: A Prototype Structured Modeling Environment." *Working Paper No. 377, Western Management Science Institute, UCLA*, May 1990.

[24] Geoffrion, A. M.,"SML: A Model Definition Language for Structured Modeling," *Working Paper No. 360, Western Management Science Institute, UCLA*, August 1990.

[25] Geoffrion, A.M., "Indexing in Modeling Languages for Mathematical Programming," *Working Paper No. 371, Western Management Science Institute, UCLA*, November 1989 (*Management Science*).

[26] Geoffrion, A.M. , "The Formal Aspects of Structured Modeling," *Operations Research*, **37: 1**, 1988 pp. 30–51.

[27] Geoffrion, A. M., "An Introduction to Structured Modeling," *Management Science*, **33: 5**, 1987 , pp. 547–588.

[28] Greenberg H.J., "A Functional Description of Analyze : A Computer Assisted Analysis System for Linear Programming Models," *ACM Transactions on Mathematical Software*, **9: 1**, March 1983.

[29] Hull, R. and R. King, "Semantic Data Modeling: Survey, Application and Research Issues," *ACM Computing Surveys* **19: 3**, September 1987.

[30] Hwang S., "Automatic Model Building Systems: A Survey," *Transactions on Decision Support Systems*, April 1985.

[31] Jones, C.V., "An Introduction to Graph Based Modeling Systems, Part I: Overview," **2:2**, *ORSA Journal on Computing*, Spring 1990, pp. 136–151.

[32] Krishnan, R., "Knowledge Based Aids for Model Construction", Unpublished PhD Thesis, University of Texas at Austin, Dec. 1987.

[33] Krishnan, R., "A Logic Modeling Language for Model Construction," *Decision Support Systems,* **6**, 1990, pp 123-152.

[34] Krishnan, R., P. Piela, and A. Westerberg, "Reusing Mathematical Models in ASCEND", forthcoming in the *Proceedings of the NATO ASI on Decision Support Systems*, A.B. Whinston,

et. al. (eds.), Springer-Verlag, New York, 1991.

[35] Ma P., F.H. Murphy, and E.A. Stohr, "Computer Assisted Formulation of Linear Programs," *IMA Journal of Mathematics in Management,* September 1987.

[36] Peckham, J. and F. Maryanski, "Semantic Data Models," *ACM Computing Surveys,* **20: 3**, September 1988.

[37] Piela, P., "ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis", Ph. D., Dissertation, Carnegie Mellon University, 1989.

[38] Ramirez, R., C. Ching, and St. Louis, "Independence and Mapping in Model-based Decision Support Systems," Decision Information Systems Working paper, Arizona State University, 1991.

[39] Raghunathan, S.,"An Artificial Intelligence Approach to the Formulation and Maintenanace of Models", Unpublished Ph. D. thesis, University of Pittsburgh, Pittsburgh, PA, 1990.

[40] Raghunathan, S., R. Krishnan, and J. May, "Computer-Assisted Model Development: A Belief Maintenance Approach", Working Paper, Decision Systems Research Institute,SUPA, Carnegie Mellon University, Pittsburgh, PA, 1991

[41] Shipman, D.W. "The Functional Data Model and the Data Language DAPLEX," *ACM Trans. on Database Systems,* **6: 1** March 1981.

[42] Sprague, R.H. and E.D. Carlson, 'Building effective Decision Support Systems," Prentice Hall, 1982.

[43] Su, Y.W., "SAM*, A Semantic Association Model for Corporate and Scientific-Statistical Databases," *Information Science*, 29, 1983.

[44] Dalen, D. van, "Logic and Structure," *Springer-Verlag*, New-York, 1980.

Distribution List

Agency	No. of copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943	1
Library, Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Department of Administrative Sciences Library Code AS Naval Postgraduate School Monterey, CA 93943	10
Ramayya Krishnan SUPA Carnegie-mellon University Pittsburgh, PA 15213	1
Sumitra Mukherjee SUPA Carnegie-mellon University Pittsburgh, PA 15213	1
Hemant K. Bhargava Code AS/Bh Naval Postgraduate School Monterey, CA 93943	5

List of Recent Working Papers

1991

91-01 **Shu S. Liao**, **Thomas P. Moore**, and Andrew G. Mackel
"Modeling the Transportation and Logistic Support System for the Aviation Assets Aboard a Navy Aircraft Carrier", October 1990.

91-02 **Nancy Roberts**
"Case Development Program Naval Postgraduate School: An Overview," May 1991.

91-03 **Thomas P. Moore**
"Collection of Wartime Data-are Reserves the Solution?," August 1991.

1990

90-01 **Hemant K. Bhargava**
"A Simple and Fast Numerical Method for Dimensional Arithmetic," March 1990.

90-02 **Thomas P. Moore**
"A Multiple Repairable Equipment and Logistics-Maintenance System (REALMS) Model," April 1990.

90-03 **Daniel R. Dolk**
"Structured Modeling and Discrete Event Simulation," August 1990.

90-04 **Daniel R. Dolk**
"Model Integration and Modeling Languages," March 1990.

90-05 **William R. Gates** and **Katsuaki L. Terasawa**
"The Economics of Defense Alliances," June 1990.

90-06 **Katsuaki L. Terasawa** and **William R. Gates**
"Allies, Adversaries and Commitment in Defense Alliances," September 1990.