1990

ISSUES IN DISTRIBUTED MODEL MANAGEMENT SYSTEMS

Waleed A. Muhanna
The Ohio State University

ISSUES IN DISTRIBUTED MODEL MANAGEMENT SYSTEMS

Waleed A. Muhanna
Faculty of Accounting and Management Information Systems
The Ohio State University

ABSTRACT

Within the decision support system area of research, there is a growing body of work focusing on the topic of Model Management (MM). This paper proposes extending the concept of a Model Management System (MMS) to function within and exploit the capabilities of a distributed computing environment. A framework for research in Distributed MMS (DMMS) is discussed. In this framework, distribution of MMS functions can occur along two principal dimensions: distributed model-base management and distributed model building and execution. A variety of design and implementation issues which are relevant to each dimension are examined. It is also argued that the structuring principles and concepts of the systems framework for model management (as implemented in a prototype system called SYMMS) are particularly appropriate and ideally suited for a distributed environment. An architecture for a prototype distributed MMS is also presented.

1. INTRODUCTION

Within the field of decision support systems, Model Management Systems (MMS) (Will 1975) have emerged as an important area of research. An MMS is a software system which provides for the creation, storage, manipulation, use, and control of models. The primary goal of an MMS is to facilitate the development, maintenance, and utilization of quantitative models, forming the foundation of an integrated environment for the support of modeling-related activities in an organization. Various MMS issues have been investigated in the literature, including model formulation, model representation, model selection, model integration, and prototype development. (For a survey and analysis of the MM literature, the reader is referred to the work by Applegate et al. [1985], Holsapple and Whinston [1988], and Muhanna [1987].)

Meanwhile, interest in the development and implementation of distributed computing systems has been growing at a remarkable rate. The advent of VLSI technology and low-cost microprocessors combined with major advances in networking technology have made distributed computing an economic reality in today's computing environments. A typical configuration, which is becoming increasingly popular, consists of a local network of high performance workstations and server machines. The server machines provide specific services to the network community, such as printing or file storage. The workstations are autonomous processing units, consisting of a powerful microprocessor, reasonable amount of semiconductor memory (two to ten megabyte), graphics display with frame buffer, local hard disk (ten to seventy megabyte), and a network interface (ten megabit Ethernet). These workstations run multi-tasking operating systems and are used for wordprocessing, office automation, engineering design, and decision support.

The advantages of distributed computing systems are well-known and widely acclaimed (Enslow 1978), the most important of which are enhanced system performance through parallel processing, high availability and increased reliability, ease of modular and incremental growth, and automatic load and resource sharing. While there seems to be considerable experience in the development of distributed applications and the design of distributed database systems (DDBMS), it remains unclear how Model Management Systems (MMS) could be designed to operate in and benefit from a distributed computing environment.

As distributed computing systems become even more popular, we should rethink the model management approaches that were developed for the mainframe-based modeling environment and develop Distributed MMSs which allow modelers and decision makers to work freely in a networked distributed computing environment and reap its full advantages whenever possible. This paper investigates some of the challenging issues and problems involved in the design and implementation of DMMS. It outlines an agenda for research in DMMS and describes the progress we have made thus far.

One of the frameworks proposed for model management is the systems framework (Muhanna 1987; Muhanna and Pick 1988). Inspired by concepts from systems theory, the framework provides a rich and intuitively satisfying view of models, together with a collection of structuring principles that are fundamental for capturing the semantics and structural relationships in a modeling environment. The framework compares favorably with other proposals for model management, since it addresses important MM problems, such as model-model linkage, model-data linkage, and model reusability. In this framework, models are defined and constructed to present a well-defined interface, consisting of a set of input ports and a set of output ports. Instances of these models can be used in

isolation or interconnected to configure a new, more complex, composite model. This view suggests a graph-oriented, nonprocedural, and hierarchical approach for model composition.

We begin by illustrating some of the systems framework concepts in Section 2 in order to provide context and to show how these concepts are particularly appropriate and ideally suited for a distributed environment. Two principal dimensions along which MMS distribution can occur are identified: model-base (centralized/distributed) and model compilation, loading, and execution (local/distributed). Sections 3 and 4 examine the design and implementation issues that are relevant to each dimension. An architecture for a prototype DMMS, which is being implemented, is also presented. Finally, the status of our current research and conclusions are presented in Section 5.

2. SYSTEMS FRAMEWORK CONCEPTS: A SUMMARY

Underlying the Systems Framework for Model Management (Muhanna 1987) is a simple premise suggested by the very definition of the term "model." We define a *model* of a reference system as being itself a *system* expressed in a formal language and synthesized from representations of selected elements of the reference system and the inter-relationships between them. Indeed, the essence of a model is that it is an analogue that can be used to reason about the reference system it represents.

In the systems framework, a model is defined and constructed to mirror (as much as possible) the reference system it represents. A model presents to the environment a well-defined interface, consisting of a set of input ports and a set of output ports, corresponding to its input variables and output variables, respectively. There is a clear separation between model interface specifications and its implementation details. The former defines an abstraction called a *model type*, while the latter is captured in the concept of a *model version*. Both model types and model versions may have specific assumptions associated with them. A single model type may be realized by more than one *model version*, each implementing a different relationship, employing a different solution algorithm, or making different assumptions.

Decomposition and the use of hierarchy are two strategies that humans use to cope with complexity. Both are fundamental to the analysis, design, and implementation of complex systems. The systems view of models brings these structuring principles to bear on the problem of model construction. It suggests a modular and hierarchical view of models which reflect and fully exploit the parallel structures in the systems being modeled. This permits top-down model construction by step-wise refinement as well as bottom-up construction by coupling existing reusable model components.

The notions of model types, versions, and coupling endow an MMS with the qualities of modularity and extensibility. In particular, models can be composed hierarchically in terms of (independently developed) component models by interconnecting the output ports of one model with the input ports of another. Model versions built in this way are called *composite*, while models without components are called *atomic*. An atomic model-version explicitly describes relationships between inputs and outputs and specifies the solution (transformation) process algorithmically. A composite model-version, on the other hand, is specified by means of a *configuration scheme*.

A configuration scheme specifies how *instances* (copies) of component models are coupled to form a higher-level composite model. *Instantiation* allows multiple copies of the same model to be used in the construction of a new higher-level model. Each instance is a distinct replica that has its own set of input ports, output ports, and (possibly) internal state variables associated with its implementation. Both model types and model versions can be instantiated. Figure 1 shows a specific production-mix model which was constructed by coupling a generic Prod-Mix-LP model solver with a forecasting model (Forecast) and other components to get the remaining data from a database or the user. The model solves the following LP problem:

$$\underset{\mathbf{x}}{MAX}\ z = \mathbf{cx}, \text{ subject to } \mathbf{Ax} \leq \mathbf{b}, \mathbf{Ix} \leq \mathbf{s}, \text{ and } \mathbf{x} \geq \mathbf{0}.$$

The vector of product demand, **s**, is computed by the forecast model.

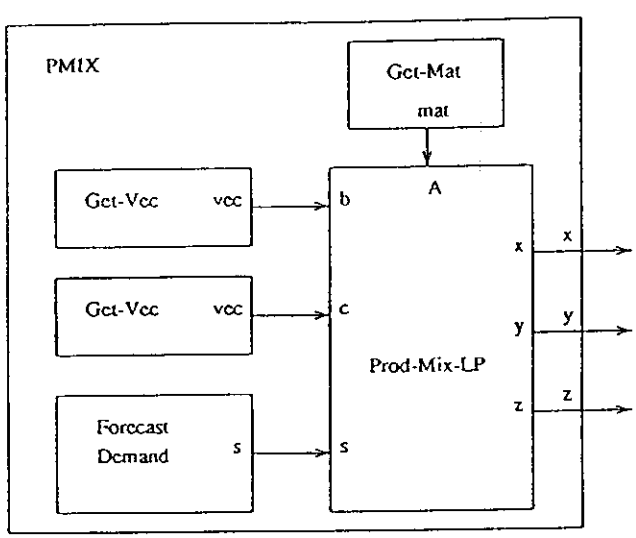

Figure 1. A Composite Model Configuration

We have defined a formalism for specifying composite models and developed a Model Description Language (MDL) which embodies this formalism (Muhanna and Pick 1988). MDL provides a high-level, non-procedural textual

means that facilitates both top-down and bottom-up hierarchical construction of models. Much of MDL's utility, as implemented under a prototype model management system, called SYMMS (Muhanna 1989c), derives from the ability to piece together individual models to construct new composite models which can be executed without writing a single line of program code.

A prototype model management system, called SYMMS, was developed in order to demonstrate the feasibility and utility of the concepts and structuring principles suggested by the systems framework for model management. Consisting of approximately 12,000 lines of code (mostly in Modula-2 with some routines written in C), the prototype runs under the Berkeley UNIX 4.3 operating system. SYMMS furnishes an integrated and coherent environment for model management. Users submit MDL modules for registration. The system verifies the completeness and correctness of these modules and provides facilities for their management and subsequent usage.

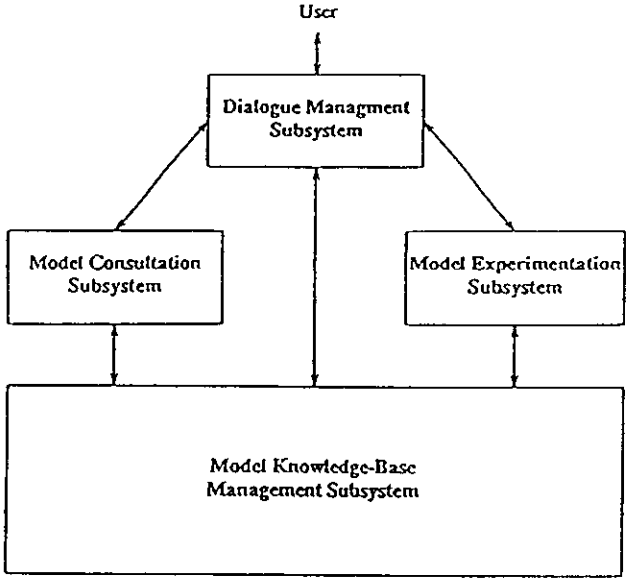

Figure 2. SYMMS Architecture

Figure 2 shows the overall architecture for the MMS SYMMS. SYMMS has four major subsystems: the Model Knowledge Base Management Subsystem (MKBMS), Model Consultation Subsystem (MCS), Model Experimentation Subsystem (MES), and a Dialogue Management Subsystem (DS). The MKBMS has overall custodial functions over model-related information and it provides facilities for use by two other subsystems - the MCS and the MES. A Dialogue Management Subsystem mediates interactions between the user and the system; it provides the user with an integrated view of the system and hides the complicated internal structure.

Central to this architecture is the idea that an integrated Model Knowledge-Base (MKB) serves as the basis by which model-related information can be shared and maintained. The MKB is managed by the MKBMS just as a database is managed by a DBMS. The MKBMS primarily consists of two layers of software. The bottom layer is the *storage management components* (SMC), which provides for reliable storage, retrieval, and update of all model-related information. It provides for version storage management at two levels: the level of model source code and the level of MDL version modules. Built on top of the SMC is the *Model Librarian*, which serves as a directory/dictionary for available models, maintains integrity and inter-module consistency, and furnishes a model-oriented interface (through check-out and check-in operations) to other MM subsystems (Muhanna 1989b).

The Model Consultation Subsystem (MCS) is comprised of two major components: a model selection component and a model construction and maintenance component. The *model selection component* provides high-level query facilities to help various MMS users in locating and selecting models that could meet their needs. The second component of the MCS is called the *model construction and maintenance component* and is intended for use by both model developers and model implementors for defining new model types, creating new composite or atomic model-versions, and for updating existing types and versions.

The Model Experimentation Subsystem (MES) is focused on supporting decision makers, who utilize models to improve their decision-making process. Once an appropriate model has been selected (perhaps by using the MCS), the decision maker can utilize the MES to run the model. The MES provides for the loading, instantiation, parameterization, sequencing, and execution of models. It furnishes a run-time environment to instantiate models and provides for their synchronization (scheduling) and communication via links.

In the next two sections, we study the major issues involved in extending SYMMS architecture and implementation to function within and exploit the capabilities of a distributed computing environment. Distributing the MKB component is first examined in Section 3. Section 4 then looks at the issue of extending the MES component to support distributed compilation and execution of (composite) models.

3. DISTRIBUTED MODEL BASE ARCHITECTURE

Central to the MMS architecture is the MKBMS, which provides for the storage and manipulation of model-related knowledge. The MKBMS is perhaps the most obvious area where distribution can occur. Three principal related issues are involved here: the physical location, the logical organization, and managerial control and administration of the MKB. Before presenting our proposed approach, it is useful to examine the alternatives available to us.

3.1 Physical Location Issues

The MKBMS (the MKB and supporting software) could be located centrally on one machine but made available to all through remote manipulation over the network. The idea is to implement the MKBMS as a *server* rendering services to multiple *clients*. The clients in this case are instances of other MMS components, namely, the MCS and MES. A client requests that a service (e.g., check-ins, check-outs, retrieval) be performed by sending a message to the server. The MKB server receives a request for service from a client, performs that service, and returns a completion message if necessary. Needless to say, the clients and the server may be running on the same or different machines in the network. Figure 3 depicts this physical arrangement.

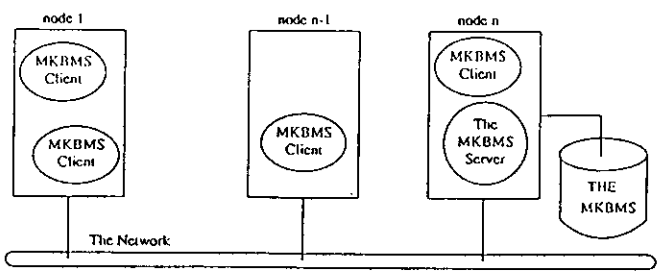

Figure 3. Centrally Located MKBMS

Many distributed systems have been successfully implemented using this client/server paradigm. For example, distributed file systems such as Sun's NFS (Walsh, Lyon, and Sager 1985) supports transparent access to shared files among heterogeneous workstations, allowing application programs to operate on remote files exactly as if they were stored on a local disk.

Compared to a single MKB server arrangement, a more advantageous but more expensive solution is to have (at the extreme) a fully partitioned and replicated distributed MKB. This arrangement, illustrated in Figure 4, involves having multiple, closely coordinated, MKB servers, each running on (possibly) a different host and managing one or more partition replica. There are complex design and implementation issues to consider here. Important design questions include (1) how the MKB should be partitioned, (2) how many copies of each partition should be maintained, and (3) on which network node should these copies be stored.

For management and control purposes, a good way to partition the MKB might be along the lines of organizational function, by organizational level, model class, or a combination of these. For performance purposes, one would have to consider different criteria, including access patterns, size, and network topology. This is clearly a hard design-optimization problem, not unlike the distributed database design and file location problems, which are known to be generally NP-complete (Eswaran 1974). Heuristic techniques have been developed for solving special cases of the file assignment problem (Morgan and Levin 1978; Dowdy and Foster 1982; Wah 1984) and they can be adapted to the context of model bases.

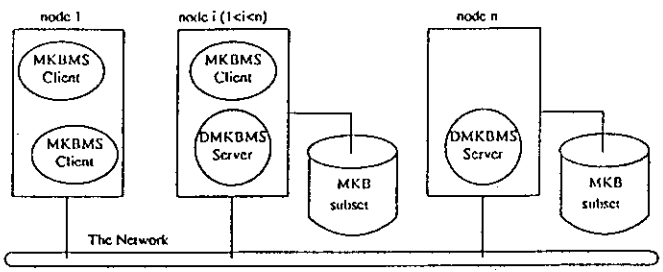

Figure 4. A Physically Distributed MKBMS

Beyond these design problems, there are also difficult implementation issues that must be addressed. To achieve a reliable and robust distributed MKBMS would require addressing problems similar to those encountered in a distributed database context, e.g., maintaining the integrity of replicated data and performing consistent atomic updates (Bernstein and Goodman 1981; Reed 1983). In a distributed system, an update operation may affect data stored at several nodes in the network. Thus, it is necessary to provide some mechanism for coordinating such updates so that an action performed at one node is successfully committed if and only if the related actions in other nodes can also be committed. The atomic update problem is part of the larger problem of transaction management in distributed systems, which has been the focus of much research. Solutions to these problems exist and have been successfully used in distributed database systems such as SDD-1 (Hammer and Shipman 1980), distributed INGRES (Stonebraker 1979), and R^* (Lindsay et al. 1984).

In deciding on an appropriate physical arrangement, one must consider the trade-offs involved. Centralizing the MKB under the control of a single server simplifies the design and implementation and helps keep the data consistent, but when the MKB is large, there may not be enough disk space on a particular host to store all the models together. Even if there is enough disk space, this arrangement tends to induce network and disk contention problems when many client users access the single server.

Distributing the MKB over several machines in the network has the advantage of significantly reducing the disk and (if set up correctly) the network contention problems. In addition, partitioning the MKB enhances the overall efficiency by bringing information closer to the person requiring it, thus making queries and updates proceed more rapidly. The replication of the entire MKB (or partitions thereof) on separate machines on the network

overcomes the problem of single points of failure, thereby enhancing the overall availability and reliability of the system. Moreover, since the load could now be spread among the machines where copies are stored, replication could also result in enhancing efficiency of MKB access operations. However, when there are considerable update activities, the high overhead involved in the complex protocols required to maintain the mutual consistency of multiple copies could thwart the efficiency gains promised by such a scheme.

3.2 Logical Organizations of the MKB

Orthogonal to the issue of physical location is the question of the logical organization of the MKB: Should the MKB be viewed as a single logical unit or as multiple logical units? If the latter, should these multiple logical units be maintained independently? Regardless of the alternative chosen for the physical location, the choice of a particular logical view impacts on the nature of the implementation and is influenced by the decisions relating to the division of administrative and managerial responsibilities.

When viewed as a single logical unit, a MKB could be spread over many machines in a network or stored centrally on one large server, but this remains transparent to the user, i.e., as far as he/she is concerned, all the information appears to be part of a single unit resident locally. This view is consistent with the philosophy underlying the concept of MM, namely, that models constitute an organizational resource that should be managed as such and that models should be viewed from an organization-wide perspective and not within the context of a specific function or activity. Despite its conceptual simplicity, however, supporting the single logical MKB view is difficult when the MKB is physically distributed through partitioning, replication, or both. The problems of transaction management and integrity maintenance become extremely complicated. A central or fully replicated catalog is needed to locate desired information, greatly complicating the query processing operations. The cost of data communication now becomes a new factor in determining an optimal query processing strategy. Furthermore, as indicated in the previous section, complex protocols are required to perform atomic updates and maintain the consistency and referential integrity of replicated data.

One alternative is to view the MKB as an integrated collection of logical units. This view permits the logical partitioning of the MKB (e.g., by level, function, project, model class) in a manner independent of its physical organization, thereby providing a way to cope with the complexity of the MKB and to provide a more focused search. However, if by integrating this collection of logical units we mean enforcing referential integrity constraints between models across the logical units, then the physical implementation issues are just as difficult as they would be had the MKB been viewed as a single logical unit. Much of this implementation complexity is removed when this logical partitioning defines independent MKBs or domains of integrity management.

3.3 Control and Administration Issues

Beyond the issues of physical location and logical view, there are also control and administrative concerns that must be resolved. Regardless of the logical and physical arrangement adopted, a central question remains. The question revolves around the issue of control over the modeling activities in an organization: who does modeling, how they do it, and what they do with it? An important role, which is critical for the success of an MMS system, is that of a MKB administrator. This role is analogous to that of a database administrator, and it entails broad responsibilities that include setting and enforcing standards for model documentation, coding, validation, security, and integrity. Given the complexity of the MM domain, however, it is more likely that the role of model-base administrator be played by a committee or a group of individuals as opposed to a single person. The issue here is whether the *function* of model-base administration ought to be centralized or decentralized.

These fundamental managerial considerations persist, irrespective of the physical location decision or the logical view chosen. Centralization of modeling control and the MKB administration function allows close monitoring and enforcement of protocols and standards to preserve organizational integrity of modeling activities. The major drawback of this arrangement is that a bureaucratic barrier could emerge stifling innovation and discouraging experimentation. A decentralized arrangement may therefore prove better if the desire to preserve the autonomy of organizational units is more important than maintaining organization-wide integrity of modeling activities. There does not seem to be a clear cut choice that is appropriate for every situation. The decision whether to centralize or decentralize control must be based not only on the choice of the logical view, but also on the organization's size, norms, and objectives.

The question of where control ought to reside and whether the administration function should be centralized or decentralized are not unique to the MM context. Indeed, these questions are aspects of the broader, classic MIS issue and long-standing debate of whether to centralize or decentralize computing in an organization. In the context of general computing, King (1983) suggested that while the economics of the deployment decisions are important, the driving force behind this ongoing debate centers around the issue of control. In the MM context, the arguments are similar but there are clear differences. One such difference is that we assume an environment in which a networked, as opposed to stand-alone, decentralization decision with regard to the hardware infrastructure has already been made. Hence, the economics of deployment

decisions revolve around the one-time cost of the implementation needed to support a given physical and logical view. Moreover, compared to a stand-alone decentralization, this networked arrangement permits organization-wide control and model administration, thereby preserving some of the benefits of centralization.

3.4 Design of Distributed MKBMS

The previous sections have outlined the pros and cons of alternative arrangements with regard to the physical location, logical view, and control and administration of a MKBMS in a networked computing environment. In this section, an architecture for a distributed MKBMS capable of delivering the desired modeling support is presented and the rationale behind its design decisions are discussed.

The challenge is to find an arrangement for MKBMS that benefits from the networked computing environment, meets user needs and provides them with an opportunity to experiment, all without creating problems for management control and modeling operations. Since these seem to be conflicting objectives, there is no universal "best" choice among the combination of alternatives outlined above. Some alternatives, however, appear more attractive than others. Since the actual physical location of the MKB could be made transparent to the user, the choice among alternative physical location arrangements could be changed as needed based on performance and implementation cost considerations. On the other hand, the choice of a logical view and decisions on the control and administration of the MKB must be informed by an understanding of the nature of the modeling environment, users' needs, and management requirements. The issue of control and administration must be recognized as the most important one because of its influence over other issues; hence it will be examined first.

Centralized administration of an MKB works fine for a single user or a small group of users. Such an arrangement, however, does not seem to be appropriate for large, diverse groups of users. The reason stems from the fact that the process of modeling tends to be iterative and tentative in nature (Muhanna 1989b). This stands in sharp contrast with database management, where only stable historical data are stored and later updated in response to events. Furthermore, modeling tends to be an individual or team activity that is function or project specific.
Centralized administration of the MKB may, because of lack of flexibility or responsiveness, place a premium on modeling innovation and experimentation. Also, for political reasons or security concerns, users may not be inclined to share some of their models, particularly when they are experimental in nature. These conditions may cause users to avoid using the system at all or at least to circumvent it by copying all of the required information to their local machines and then making the required changes. Extreme decentralization of control is no panacea either, since it fails to address the need for some discipline in the storage, sharing, and management of models as an organizational resource.

An intermediate arrangement appears to be a promising compromise solution that takes into account organization-wide management concerns while allowing individuals and groups sufficient autonomy in their modeling activities. The basic idea is to partition the MKB into a collection of loosely-coupled logical MKBs, each defining a distinct administrative domain. There are three types of logical MKBs: *main*, *group*, and *private*. These types differ in terms of their access and manipulation rights and the extent to which integrity constraints are enforced. An organization's MKB would consist of a single instant of the *main* logical MKB type and one or more instances of the *group* and *private* logical MKB types.

The *main* MKB is a centrally administered, organization-wide repository containing a relatively stable collection of validated and verified models. While access limitations might be required, the main MKB has the least restrictive retrieval and execution rights and the most restrictive and elaborate storage and update procedures. Instances of models in the main MKB may be used as components in new composite models or executed by virtually any authorized user. A set of privileged users, perhaps through the main MKB librarian, may check out model versions for update. Before the new version is checked back in, it would undergo a rigorous sequence of validation and verification tests. Integrity constraints, particularly referential integrity, are strictly enforced by the system at all times.

A *group* MKB provides a mechanism through which a collection of users (e.g., members of a functional area or a project team) can share highly specialized, experimental, partially verified, or incomplete models. A group member may deposit such models in the MKB, which can be executed or checked out by other members for browsing or update as authorized. An individual user may belong to more than one group, each of which has a distinguished member, called the group owner, who plays the role of the administrator for that group's MKB. Referential integrity constraints are optionally enforced in group MKBs.

A *private* MKB is created for an individual user who assumes complete administrative and control responsibilities over it. A private MKB contains private experimental models and/or copies of models checked out from the main or a group MKB. The contents of a private MKB are accessible only to its owner. Model types or versions deposited in a private MKB need not be verified and may reference instances that do not exist in the private MKB.

Each user may define an ordered sequence of logical MKBs to be searched to satisfy his/her requests (e.g., select models, resolve references in composite models to their components). This sequence defines an aspect of the

run-time environment, called *search path*, for that user. For example, a default search path might specify looking up the user's *private* MKB first and, if the desired instance of a model type or version is not found, then an attempt is made to find it in the *main* MKB. A user can customize his search path to be any desired sequence of MKBs he/she is authorized to access. The search path can specify the physical location of the various MKBs so that client programs running on behalf of the user can connect to the servers at those locations. Alternatively, the path could specify only the globally unique names of the various MKBs, whose location can be determined dynamically using a variety of means (e.g., global name server, broadcast). Figure 5 illustrates the proposed architecture.

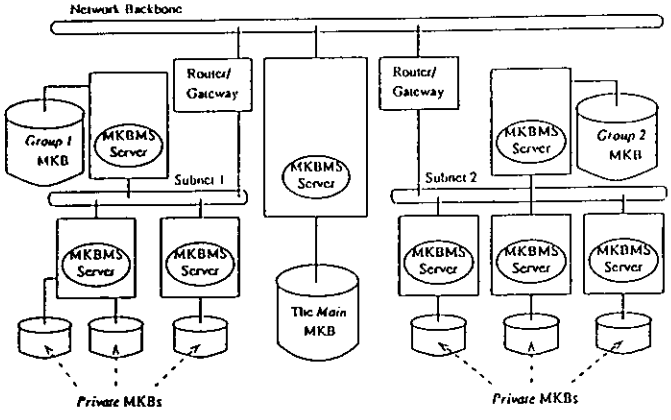

Figure 5. Distributed MKBMS Architecture

The above arrangement attempts to strike a balance between innovation and autonomy on the one hand and discipline and control on the other. Central control and administration are maintained over the *main* MKB, which contains models that are used and depended upon by the organization as a whole. Meanwhile, users are allowed to fully utilize the system facilities to manage their own sets of experimental models. The proliferation of private MKBs could be controlled by assigning the privilege of creating a private MKB to the administrator of the main MKB. That administrator should also take an active role in establishing codes for good modeling practices, promulgating these codes, and coordinating the modeling activities throughout the organization.

Having decided on an appropriate arrangement for control and on a logical view of the MKB, the choice of a physical location can (and should) remain flexible since it can be made transparent. Initially, the entire collection of logical MKBs and the MKBMS server could both reside on some central machine. As the size of this collection grows or when disk and network contention reach a certain threshold, the collection could be spread physically on multiple disks or partitioned and/or replicated across machines on the network.

As described in Section 3.1, physical distribution of the MKB greatly complicates the implementation since it must provide support for distributed transaction management and distributed version management. Our decision to view the MKB as a collection of independently managed logical MKBs may actually simplify these problems, particularly when we avoid replication. The reason is that the partitions of the physically distributed MKB are now loosely-coupled, since each of these partitions corresponds to a logical MKB.

The physical distribution of the various logical MKBs could parallel the physical design of the underlying network. Many local-area networks today are designed around a high-bandwidth backbone to which departmental subnets are attached using bridges, gateways, or routers. Such design is intended to provide for better traffic and failure isolation. It also suggests a simple way for distributing the MKB (Figure 5). The main logical MKB could be stored physically on a machine directly attached to the network backbone, while servers for group and private MKBs could be placed on machines on the appropriate group subnets.

3.5 Other Issues

Problems which are not particularly unique to distributed MMSs include security and query optimization. These problems can be addressed in a manner similar to those employed in distributed DBMSs. One problem, which is perhaps more unique to model management, concerns change notification. Changes made to an MKB may have a ripple effect, requiring many actions. For example, updating a model version may affect the validity of other models that use an instance of it as a component. Such changes must be monitored and propagated in a controlled manner. The creation of a new model version or the deletion of an existing one are actions that could be deemed worthy of announcement to interested parties.

In small groups where members are in close proximity, notification can occur during regular verbal communication. In large groups whose members are spread out throughout the organization, some notification mechanism must be put in place to facilitate this process. A rule-based notification mechanism has been proposed as part of the activity manager in SYMMS and is described elsewhere (Muhanna 1989c).

In the next section, we examine another dimension along which MMS distribution can occur.

4. DISTRIBUTED MODEL BUILDING AND EXECUTION

A principal objective of an MMS is to increase the productivity in organizational modeling and decision-making activities. It must therefore provide concepts and techniques to enhance the efficiency of both model development and model experimentation processes. As described in Section 2, the systems framework for model manage-

ment supports the notions of reusable components and suggests a graphical, non-procedural, hierarchical approach to the construction of composite models, both of which can significantly reduce the time it takes to develop a model.

Once developed and validated, however, a model is then used to support decision making. This process of model experimentation is invariably time-consuming because it is often repetitive (when there are multiple scenarios to be tested) and computationally intensive, particularly when large simulation or combinatorial (e.g., integer or non-linear programming) models are involved. Speeding up the model experimentation process can contribute to meeting an important objective of MMSs, namely, increasing the productivity of model builders and users. A distributed computing environment has the potential of making such speedups a reality.

One obvious virtue of networking in a distributed computing environment is that applications are no longer restricted to using only those resources located on the local system. Now they can, in principle, transparently use resources located throughout the network. These resources can be centralized MKBMS servers, as described in Section 3, as well as raw computing power available on the various machines on the network. This means that computationally intensive tasks, such as model compilation, linking, and execution, could be done in parallel by distributing the work across multiple machines on the network.

A common computing environment consists of many workstations connected together by a high-speed local area network. When viewed as an aggregate, these workstations represent a significant computing resource. This resource, however, is often underutilized because the workstations usually serve as personal computers and are frequently idle. The ability to distribute computational tasks could greatly enhance the utilization of such systems. An effective DMMS should allow users to tap into the enormous computing power available on the network. Users should be able to develop models locally and compile and execute them on remote machines (whether they are workstations, mainframes, or supercomputers) that offer the optimal performance and characteristic for the models. To make this transparent to the user, the DMMS should hide the details of the network communication and remote execution and offer seamless integration of the (heterogeneous) distributed computing environment. Model developers would therefore concentrate on the task of model conceptualization and formulation, not on writing distributed programs or keeping track of the static and dynamic characteristics of the machines and the network. This translates into better utilization of the network-wide resources, increased portability of models' codes, and enhanced modeler/decision-maker productivity.

A major goal in developing DMMS is therefore to benefit from the availability of multiple processors on the network by first identifying and then exploiting inherent parallelism (i.e, potential for concurrent execution) in the activities of model compilation and execution. To this end, the systems framework for model management as implemented in SYMMS is particularly appropriate for a distributed computing environment. There are two basic reasons behind this.

First, the framework draws a clear distinction between "modeling in the small" and "modeling in the large," (Muhanna 1987), which allows both small-grain as well as large-grain parallelism to be identified and exploited. Atomic models are implemented using a conventional programming language (e.g., Modula-2) augmented with message passing primitives. Special programming languages and/or parallelizing compilers can also be used to exploit small-grain parallelism (i.e., at the level of the instruction or function) in atomic models or their solvers whenever SIMD (vector) or shared-memory MIMD processors are available. The configuration component of MDL is used to hierarchically structure instances of atomic models into composite models. When expressed directly in terms of its constituent instances of atomic models (i.e., when the composition hierarchy is flattened), the configuration of a composite model is basically a graph, whose nodes represent the instances of the atomic models and arcs represent the inter-communication (dependency) structure between them. This graph defines a partial ordering of the atomic instances (assuming it is acyclic) and displays large-grain parallelism (i.e., at the level of one or more component instances of atomic models) that can be exploited. These grains can be downloaded to multiple remote machines for concurrent execution, providing higher performance for users and better utilization of computing resources throughout the network.

A second reason why the systems framework for MM is particularly suited to a distributed environment lies in the data-flow paradigm it embodies at the macro level. Instances of atomic models are highly encapsulated entities which communicate by making references to local ports and not to other instances, machines, or communication channels. Furthermore, communication between instances running on the same or on different machines is effected *uniformly* via message passing. Message passing is the natural mechanism for communication across a network. Techniques for exchanging data (e.g., common blocks and standard procedure calls) which have been developed for single or multiprocessor systems are not suitable for this environment because they are based on the notion of shared-memory, which is not readily available in a distributed computing system.

The notions of encapsulation, composition via coupling, and message passing are very useful concepts to capitalize on in achieving distributed model execution. A *task* is an execution of an instance of atomic model, i.e., it is the run-time realization of that instance. When a request to execute a composite model is submitted to SYMMS, the

system first flattens the model's composition hierarchy. The flattened composition represents the composite model in terms of its ultimate components, i.e., as a network of interacting instances of atomic models. This network is realized by means of an isomorphic network (called a task graph) of communicating tasks, where one-way communication channels interconnect those tasks and buffer their output.

In a distributed implementation of SYMMS, a critical issue is how to *effectively* exploit inherent parallelism in a given task graph and benefit from the availability of parallel hardware. In particular, four broad problems must be addressed:

(1) *Task Graph Characterization* – determining execution times of each task as well as the frequency and amount of communication between tasks in a task graph.

(2) *Task Graph Scheduling* – assignment of tasks to (logical) processors and local CPU scheduling of individual tasks for maximum global performance.

(3) *Implementation* – building mechanisms to (a) locate and allocate physical processors, (b) build and load executable images, and (c) effect communication and synchronization between tasks executing on the same or on different machines.

(4) *Support for heterogeneous machines* – a new dimension of complexity to the above problems.

Of the above four problems, the last two will be examined next. The first two issues are investigated in Muhanna (1989a) and Muhanna and Pirkul (1990), respectively, and will not be addressed here. Suffice it to say, however, that one can obtain estimates for parameters of the task graph through sample executions and under certain assumptions through basic syntactic analysis. Also, the problem of task graph scheduling turns out to be NP-hard, if one insists on an optimal assignment. Surprisingly, very little research has been done to solve this particular class of scheduling problems. A heuristic scheduling algorithm has been developed, a description of which is presented in Muhanna and Pirkul.

4.1 Implementation Issues

An effective distributed MMS must exploit the availability of parallel hardware to speed up the compilation, linking, and execution of composite models. To this end, a number of implementation issues must be addressed. First, a decision must be made as to whether scheduling task graphs should be done statically (perhaps when the model is stored) or dynamically (on the fly just before execution). Early static scheduling is more efficient but less flexible. Late dynamic scheduling can yield schedules that reflect better the dynamic nature of the computing environment (e.g., processors' availability and load).

A two-phased scheduling heuristic proposed in Muhanna and Pirkul suggests an intermediate approach in which the first phase, which depends only on the characteristics of the task graph and not on those of underlying computing systems, can be done statically, while the second phase is performed dynamically on demand. In addition, a dynamically generated schedule can be cached and discarded when the system conditions assumed during its generation change.

Executing a given task graph schedule requires the implementation of mechanisms to (1) locate and allocate physical processors, (2) compile, link, and load executable images, and (3) effect communication and synchronization between tasks executing on the same or different machines. SYMMS already has these mechanisms implemented for a computing system consisting of a single machine. A description of these mechanisms can be found in Muhanna (1989c). What is needed is an extension on these mechanisms to permit distributed compilation and execution. Such extensions should be made in such a way so as to insulate the user from distributed application design. From the user's point of view, the distributed environment should be transparent, appearing as uniform as a traditional uniprocessor.

It turns out that making the extensions necessary to realize the above objective is, while complicated and laborious, made technically possible by (1) the latest advances in networking standards, tools, and utilities and (2) the features inherent in the systems framework for model management. A brief description of these extensions is provided next.

To keep track of the status of machines on the network, coordinate the allocation of processors, and enforce access privileges, a reservation executive is needed. This reservation agent is not necessarily part of the distributed MMS, since applications other than the MMS compete for the same resources. Various facts about machines on the network are kept by the reservation executive. Machines could then be chosen on the basis of a number of criteria such as idle time percentage over a certain time interval, hardware and software configuration, and even administrative factors such as ownership and cost accounting. Periodically, machines broadcast their status to update the dynamic portion of the information maintained about them by the reservation executive. This part should be relatively easy to implement. The difficulty lies in providing reliable and efficient coordinated allocation of processors across the network. For a lightly loaded environment, coordination is not critical and independent allocation could be made at each node on the basis of recent status information broadcasted by other nodes. We have opted for this latter approach for the time being, since a network-wide coordinated reservations mechanism may not be sufficiently efficient for it to be useful in this context.

Also planned is a mechanism to compile, link, and load executable images to run on the assigned processors. Under the current implementation of SYMMS, when given a task graph (generated by an automatic flattening process of the composition scheme) to run, the system first searches an Object Code Cache for object modules corresponding to each atomic component. If none is found, the relevant source code modules are extracted from the MKB, compiled, stored in the cache, linked, and dynamically loaded as coroutines in a single UNIX process. Finally, each task is instantiated from its corresponding coroutine template as a lightweight process. Run-time scheduling and inter-task communication are supported by a run-time kernel.

In a distributed environment where multiple machines are available, parallelism can occur not just at execution time but also at the compilation and loading time. Concurrent compilation and loading of modules can dramatically speed up the start of model execution and thus greatly improve the user's productivity. To support this capability, the MES of SYMMS is being modified to run as a server on each of the nodes on the network. An MES server running on one machine becomes a client to other MES servers running on remote machines, accepting requests to compile, link, and load subsets of task graphs on their behalf. The basic idea is that once a schedule for a task graph is generated by the local MES, requests to load (and compile if necessary) subsets of that task graph are sent to the machines to which those subsets were assigned. The task graph subset is loaded as a single process with distinct threads of control for each task. Each process (task graph subset) registers its unconnected ports with a *connection server*. The connection server allows two independently started processes to connect their ports (i.e., rendezvous) using mutually agreed upon names. The run-time kernel in each process establishes connections to remote counterparts with which it needs to communicate.

Finally, the SYMMS run-time kernel is being extended to support inter-task communication between tasks running (1) on the same machine and (2) on different machines. The former is already supported in SYMMS through shared-memory and semaphores. Messages destined for remote tasks are trapped by the kernel, framed, and passed to the operating system's IPC mechanism (e.g., Berkeley UNIX sockets) for network delivery. No modification to the task source code is needed for communication with remote tasks to occur. The reason is that tasks perform read and write operations to their own ports; there is no direct naming to communication channels or reference to other tasks. The complexity of the communication mechanism is hidden by the software in the kernel layer, which furnishes a uniform interface for communication between local as well as remote tasks.

4.2 Support for a Heterogeneous Environment

Most networked computing environments contain a variety of machines (with different hardware architecture or speed) and operating systems types. It is also often the case that certain *model solvers* are constrained to run on a particular hardware and software configuration for technical, licensing, or availability reasons. A distributed MMS must therefore provide support for such heterogeneous environments in order to maximize the network-wide sharing and utilization of resources.

Invariably, the problems described earlier are intensified by the presence of dissimilar machines on the network. Both the formulation and solution of the task graph scheduling problem are more difficult in this context. Also, the difficulty of the system implementation is increased by at least an order of magnitude. In fact, the implementation would be prohibitively expensive had it not been for the availability of a common set of standard communication protocols, tools, and utilities which can be used to build a messaging platform that transparently integrates the myriad of multi-vendor machines.

However, a common communication protocol is not enough to support inter-task communication across different machines. The reason is that different machine families (e.g., VAX versus Sun) have different byte ordering and data representation schemes. Hence, one problem, which is unique to a heterogeneous environment, is that messages between machines must conform to the way data is represented on the destination machine. To do this, we plan to use the External Data Representation (XDR) (Sun Microsystems 1987) as a common scheme. Before sending a message across the network, the send routines, which are part of the run-time kernel, encode the message data using XDR as the common data representation standard. The destination run-time kernel then transforms the message to its local data representation scheme. These transformations are to be made only if the send and receive machines use different data representation schemes.

5. CONCLUSION AND IMPLEMENTATION STATUS

In this paper, we proposed extending the concept of an MMS to function within and exploit the capabilities of a networked distributed computing environment. A number of MMS research issues are either peculiar to or intensified by the presence of multiple machines. To facilitate their examination, a framework for research in Distributed MMS (DMMS) was discussed. In this framework, distribution of MMS functions can occur along two principal dimensions: distributed model-base management and distributed model building and execution.

Along the first dimension, we examined issues related to the physical location, logical view, control, and administration of the MKB in a distributed computing environment. An architecture which centers around the issue of administration and control was presented. In it, the distributed MKB consists of one *main* MKB and a collection of *group* and *private* logical MKBs. This arrangement attempts to strike a balance between innovation and autonomy on the one hand and discipline and control on the other. Actual physical distribution (through partitioning, replication, or both) of each logical MKB could occur by carefully balancing the cost of implementation and the benefit of increased performance and availability. Each logical MKB is managed by an MKB server, which provides access and manipulation services to local and remote MCS and MES clients.

The second dimension of research concerns the development of techniques to exploit the networked environment by using multiple processors to speed up the compilation, loading, and execution of models. To this end, we have found that the systems framework for MM is particularly appropriate, making our prototype MMS, SYMMS, ideally suited for extension to support distributed model building and execution. Modularity, loose coupling, and message passing are salient characteristics of a distributed system. These characteristics are basic to the systems framework through the notions of model types, composite and atomic versions, and coupling ports through links.

In supporting distributed execution, a critical issue is how to effectively exploit inherent parallelism in a given task graph representing a configuration of a composite model. Three problems were identified: task graph characterization, task graph scheduling, and implementation. We found that we can obtain estimates for parameters of the task graph through sample executions and, under certain assumptions, through basic syntactic analysis. The problem of task graph scheduling turns out to be NP-hard, if one insists on an optimal assignment. It was noted that very little research has been done to solve this particular class of scheduling problems. A heuristic scheduling algorithm has been developed, details of which are presented in Muhanna and Pirkul (1990). Finally, a number of implementation problems and solutions were described for supporting distributed build and execution in both homogeneous and heterogeneous computing environments.

The list of problems described in this paper is by no means exhaustive. Much work remains to be done in identifying new issues and enhancing proposed solutions to existing ones. Much insight could also be obtained through further analysis and user input to see how a DMMS could be designed to more effectively support modeling activities in an organization. A prototype system provides the means to test these ideas in practice. To this end, we are implementing the ideas by extending SYMMS to run in an environment consisting of a number of VAX and SUN workstations. All machines run derivatives of Berkeley BSD UNIX, whose basic IPC and networking primitives are being extensively used in the implementation. Concurrently being developed is a graphical interface for browsing and composite-model construction. The goal is to obtain a distributed MMS which forms a coherent MM environment in which the entire network and its resources act as a simple and powerful extension of the user's personal workstation. This should translate into better utilization of the network-wide resources and enhanced modeler/decision-maker productivity.

6. REFERENCES

Applegate, L. M.; Klien, G.; Konsynski, B. R.; and Nunamaker, J. F. "Model Management Systems: Proposed Model Representations and Future Designs." *Proceedings of the Sixth International Conference on Information Systems*, San Diego, December 1985, pp. 1-16.

Bernstein, P. A., and Goodman, N. "Concurrency Control in Distributed Database Systems." *ACM Computing Surveys*, Volume 13, Number 2, June 1981, pp. 185-221.

Dowdy, L. W., and Foster, D. V. "Comparative Models of the File Assignment Problem." *ACM Computing Surveys*, Volume 14, Number 2, June 1982, pp. 287-313.

Enslow, P. H. "What is a Distributed Data Processing System?" *IEEE Computer*, Volume 11, Number 1, January 1978, pp. 13-21.

Eswaran, K. P. "Placement of Records in a File and File Allocation in a Computer Network." *Information Processing 74*, IFIPS, August 1974, pp. 304-307.

Hammer, M. M., and Shipman, D. W. "Reliability Mechanism for SDD-1: A System for Distributed Databases." *ACM Transactions on Database Systems*, Volume 5, Number 4, December 1980, pp. 431-466.

Holsapple, C. W., and Whinston, A. B. "Model Management Issues and Directions." Working Paper No. 7, Department of Decision Science and Information Systems, University of Kentucky, November 1988.

King, J. L. "Centralized Versus Decentralized Computing: Organizational Considerations and Management Options." *ACM Computing Surveys*, Volume 15, Number 4, December 1983, pp. 319-349.

Lindsay, B. G.; Hass, L. M.; Mohan, C.; Wilms; P.F.; and Yost, R. A. "Computation and Communication in R^*: A Distributed Database Manager." *ACM Transactions on Computer Systems*, Volume 2, Number 1, February 1984.

Morgan, H. L., and Levin, K. D. "A Dynamic Optimization Model for Distributed Databases." *Operations Research*, Volume 26, Number 5, September-October 1978, pp. 824-835.

Muhanna, W. A. *A Systems Framework for Model Management in Organizations*. Unpublished Ph.D. Thesis, University of Wisconsin-Madison, December 1987.

Muhanna, W. A. "Composite Programs: Hierarchical Construction, Circularity, and Deadlocks." *IEEE Transactions on Software Engineering*, 1989a.

Muhanna, W. A. "On the Organization of Large Shared Model Bases." Working Paper, The Ohio State University, 1989b.

Muhanna, W. A. "SYMMS: Design and Implementation Notes." Working Paper, The Ohio State University, 1989c.

Muhanna, W. A., and Pick, R. A. "Composite Models in SYMMS." *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, January 1988, pp. 418-427.

Muhanna, W. A., and Pirkul, H. "Heuristic Algorithms for Scheduling Task Graphs in Distributed Systems." Working Paper (in preparation), The Ohio State University, 1990.

Reed, D. P. "Implementing Atomic Actions on Decentralized Data." *ACM Transactions on Computer Systems*, Volume 1, Number 1, February 1983, pp. 3-23.

Stonebraker, M. "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES." *IEEE Transactions on Software Engineering*, Volume SE-5, Number 3, May 1979, pp. 188-194.

Sun Microsystems, Inc. "XDR: External Data Representation Standard." RFC-1014, June 1987.

Wah, B. J. "File Placement on Distributed Computer Systems." *IEEE Computer*, Volume 17, Number 1, January 1984, pp. 23-32.

Walsh, D.; Lyon, R.; and Sager, G. "Overview of the Sun Network File System." *Proceedings of the Winter Usenix Conference*, 1985, pp. 117-124.

Will, H. J. "Model Management Systems." In E. Grochla and N. Szyperski (eds.), *Information Systems and Organization Structure*, Berlin: Walter de Gruyter, 1975, pp.467-482.