Flexible Integration of Manufacturing Databases Using a Model-Based Architecture Capable of Wide-Area Networking

Yicheng Tao[1]
Gilbert Babin[2]
Cheng Hsu[3]

July 1996

[1] Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY, 12180-3590.
[2] Assistant Professor, Départment of d'informatique, Université Laval, Québec City, Québec, Canada, G1K 7P4.
[3] Associate Professor, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY, 12180-3590.

ABSTRACT

Manufacturing enterprise systems entail both vertical and horizontal integration. Vertical integration involves different time frames of three basic cycles: the work order-oriented real time processes, the customer/demand-oriented aggregate systems, and the business strategy-oriented enterprise functions. Horizontal integration, on the other hand, is concerned with geographically distributed organizational systems, which typically require heterogeneous computing environments distributed not only over local-area networks but likely also over wide-area/global networks. Both integration require scalable building and adaptive configuration of systems, where interoperability must be accomplished without jeopardizing these systems' autonomy and concurrent processing. Furthermore, these systems may require active capabilities such as event-based information handling and fault-tolerant coordination among conventional database servers or data warehouses.

A model-based architecture providing flexible integration of manufacturing databases beyond local area networks is proposed in this paper. The concept of metadata independence for multiple systems is presented as the basis for the conceptual design of the architecture, while the ROPE (Rule-Oriented Programming Environment) method is developed to execute the design. Particular designs are developed to incorporate client-server technology (including CORBA) wherever appropriate.

1 AN ENTERPRISE INFORMATION MANAGER: THE OBJECTIVE

Multiple (heterogeneous, distributed, and autonomous) systems characterize distributed manufacturing enterprises. The enterprise information manager that integrates these systems (in part or in whole) needs to provide interoperability at three levels: file transfer (work flows), global query, and data and knowledge/processes management. At each level, the systems involved may require fault-tolerant, real-time, or even active capabilities in addition to the traditional database and file system operations. Furthermore, these multiple systems may have to operate over wide-area or even global networks where reliable and controllable enterprise-wide network transactions management capabilities are non-existent. To compound the complexity even further, distributed enterprises always involve dynamic changes that not only update the contents (data and operating rules) of these multiple systems, but also add or delete member systems and modify the configurations associating them. The enterprise information manager (EIM) technology must be assessed in this context. Integration which is done on a lesser scale is only accomplished by constructing special software programs which translate between individual proprietary systems. These translators are typically complex and expensive to modify in order to respond to changing business needs or improved information technologies.

The objective of an EIM is to enable manufacturing enterprises to have not only the disparate information integrated within each factory, but also to have access to relevant information across all factories, design centers, and other functional divisions. Further, the enterprises would be capable of making changes to data and have those changes automatically disseminate throughout the entire distributed organizations, which will result in having all related processes brought instantly into line. This objective can be demonstrated by using an Army logistics example which has the metadatabase installed into the configuration of Army depots and arsenals.

In the scenario, the EIM (which may be distributed) would provide a comprehensive linkage between each Army locale, thereby giving personnel the ability to search the global Army environment for particular pieces of data, and control of global events. The strength of global query capability would be matched by the ability to control events from a distance. Personnel may modify data on a global scale, such as changing the deadline in the research prototype and manufacture of a weapon system. Normally, any such scheduling change would begin a cascading effect; one Army arsenal would inform another by telephone until all were informed. Each in turn would have to determine which information systems required updating, locate the affected databases and implement the changes piecemeal, until al pertinent processes are realigned to ensure consistency. In contrast, the EIM would perform all of the above repetitious and error-prone tasks on-line. Ideally, installation of the EIM does not have to replace existing Army data structures but rather be expandable; its structure provides for computing growth within all Arsenals and Depots. This objectivity can only be achieved in steps, of course. An immediate step that this research attempts is to develop a flexible integration method applicable to EIM's under wide-area-networking (WAN) conditions. In particular, it proposes a model-based architecture with fault-tolerance to support global query, data management, and event-based processes.

The work is based on the authors' previous results [1,2,13,14,15,17]. New results which constitute the focus of this paper include a communication requirements model and a new design of the model-based architecture. These results respond to the specific needs of real-time process control [23] as well as the emerging client-server tecnologies, such as CORBA (Common Object Request Broker Architecture) and ANSI/ISO SQL3, so as to provide sufficient capabilities for active event-based processing. In addition, methods and algorithms are developed to execute the architecture. Fault tolerance is a built-in feature of the methods. Data replication is supported as a mechanism for both fault tolerance and real time process control. A prototype is implemented to verify the new execution methods. We now discuss the research problem in Section 2, and present the solution in Sections 3, 4 and 5. Conclusions are provided in Section 6.

2 AN ARCHITECTURE FOR FLEXIBLE INTEGRATION: THE PROBLEM

Industrial solutions tend to either be broad-based open systems, or be proprietary "one vendor" technology. The broad-based DAE (Distributed Application Environments)-class technology deals with API (Application Program Interface) and other inter-system interfacing tasks to provide file transfer-level of integration. A promising example is CORBA [21], which supports direct interoperability among various object-oriented systems and some gateway services for connecting relational or other traditional systems into an object environment. This class of enterprise information managers lacks a global model for the enterprise and typically does not include online knowledge bases (e.g., intelligent interfacing agents or rule-based shells) in its inter-system interface configuration design; both of which are strong suits of technologies that feature database-class capabilities at the expense of flexibility and openness. In between the DAE-class and the database-class technologies there exists data warehouse-class enterprise information managers. Prime examples include IBM's Information Warehouse and Hewlett-Packard's WorkManager.

Some global query functionalities (typically with build-in latency on the contents of information and limited flexibility on the types of queries supported) are provided in addition to file transfers.

The database-class technology - in particular, the various HDDBMS (Heterogeneous Distributed Database Management Systems) and multidatabases [3,6,8,9,11,22,24] - is somewhat antithetic to the other two classes of enterprise information managers in philosophy and practice. While being more analytically elegant and complete, available HDDBMS's tend to be limited to local-area networked environments and are not as robust as the other classes in dealing with tasks such as large-scale file transfers (e.g., engineering drawings) and work flow management. Moreover, due to the basic control methods employed, they tend to focus on the integration of information systems at a fixed point of time (the development time) and leave out such non-static aspects as (1) dynamically changing the (inter-system) processes and (intra-system) structures and (2) incrementally building or grouping systems (e.g., adding systems to an HDDBMS after the development time or connecting an HDDBMS to another). The two properties are referred to as **flexibility** in this work.

There are a few major concerns facing the database-class technology striving to extend flexibility for manufacturing over WAN's. We recognize the following problems from the perspective of this research:

1. Global serialization. Serialization across multiple databases has long been recognized as impractical if there is a higher probability of deadlock due to the network involved. The network problem of serialization in LAN environments might be solvable under certain conditions, but the same reliability issues become intractable in WAN and global network environments, where overall network management is totally lacking and instantaneous processing (for instantaneous correctness) is a physical impossibility due to delays.
2. Fault-tolerance. This is an acute concern for WAN-based interoperations. Since the global information processing among distributed systems might be interdependent, disruptions at a system would take place when any of those processings at other related systems fail or fail

to send results back and reach the system. Many systems could therefore be rendered inoperable frequently.

3. Real time process control. Manufacturing databases must support real time control processes to be implemented on shop floor. The key concern for this requirement is two-fold: performance (time frame) and storage. The architecture should be amenable to work with systems that do not possess mass storage capabilities. To satisfy stringent time constraints, local structure should always maximize parallel processing using self-reliant information.

A model-based architecture was provided by the metadatabase model to effect flexible integration of multiple systems. The previous results [13,14,15] do not include sufficient details to solve all of the problems discussed above. This research extends these results.

The metadatabase model proposes that (1) the enterprise metadata include contextual knowledge and be structured and managed as an independent full-fledged database - thus, the term metadatabase, (2) the inter-system operating knowledge be distributed into a concurrent shell system to effect interoperability at all three levels (file transfer, global query, and data and processes management), (3) a new criterion of correctness be employed without requiring a global synchronizer (which is optionally employed only for some mission-critical data that are designated dynamically and controlled by rules), (4) the metadatabase be commissioned as the enterprise information manager which directly manages the metadata instances at the enterprise level (global models and configuration logic), but only indirectly (through metadata) facilitates raw data processing at the local systems level, and (5) the metadatabase-defined data replication is used as a mechanism for both fault tolerant and real time process control over WAN/LAN environments. The metadatabase employs a distributed, rule-oriented shell system to interoperate with local systems, embed necessary knowledge of integration (at all three levels) into local environments, and let these local systems operate thereafter with this knowledge. It intervenes only when the enterprise metadata are changed, so as to update the localized knowledge. Based on these extended characteristics, the notion of **metadata independence** is defined to refer to the capability of an enterprise information manager being structurally neutral to multiple databases yet functionally supportive of their adaptive and scalable integration. This notion is directly comparable to the classical concept of data independence in single database with respect to multiple files.

The research extends the previous rule-oriented shell system [1,13] to support the overall architecture integrating WAN and LAN with reasonable performance by incorporating fault-tolerant capability. The major challenges and tasks include: (1) defining the rule-oriented shell system based on WAN and LAN technologies such as client-server architecture and CORBA and (2) developing execution methods with fault-tolerance and real time processing to support the inter-operation of shells across LANs and WANs.

3 THE MODEL-BASED ARCHITECTURE

3.1 The Concept

The structural model of the architecture is depicted in Figure 1 [13,14,16]. Note that the metadatabase can be distributed both horizontally and vertically [12]. Thus, the structural model shown in Figure 1 may be expanded recursively into a system of clusters of metadatabases and their local application systems. Each clusters might be based on an LAN, and the clustering of these clusters constitute the WAN environment. The major points from these citations can be summarized as follows. The metadatabase itself (a rigorously constructed collection of enterprise metadata) provides an integrated enterprise model for the multiple information systems, their databases, and the interactions among the different systems; i.e. the information contents and their contextual knowledge. The metadatabase approach through the architecture (1) uses the enterprise model to assist end-users performing global queries free of both technical details and a hierarchy of integrated schemata; (2) distributes the contextual knowledge to empower these local systems to update data and communicate with each other without central database control; and (3) incorporates legacy, new or changed local models into its generic structure of metadata to support evolution without system redesign or recompilation. The shells in the architecture, therefore, implements the distributed (localized) knowledge which, in turn, is managed by the metadatabase.

Figure 2 illustrates the laboratory environments at Rensselaer in which the model-based architecture prototype was developed.

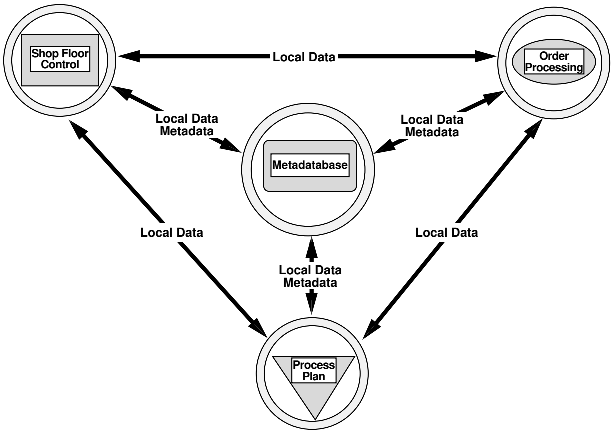


Figure 1 : The Model-Based Architecture Using a Metadatabase

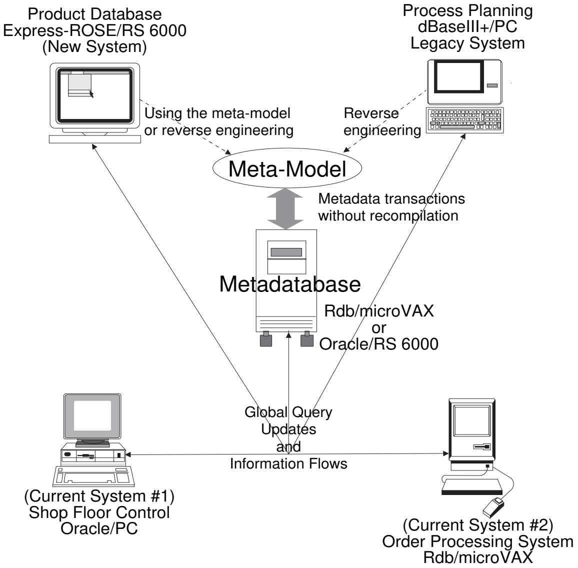


Figure 2 : The Operation of the Model-Based Architecture

The metadatabase model employs a new criterion, event/usage correctness [13], for control and thereby removes the need for global serialization, which is infeasible in WAN environments, and fosters full concurrent processing among local systems. The event/usage correctness criterion is defined as follows: a data value only needs to be updated when it is needed by an application according to its contextual knowledge. Therefore, the consistency is established on a need-to-know basis, using both operating and integrity rules (also referred to as data management rules) defined in the enterprise model. This criterion eliminates the *requirement* of serialization across multiple systems but entails knowledge methods in its place. One important point to note is that the event/usage correctness criterion defines generally how to synchronize data across different platforms and databases; but does not prohibits serialization from being applied as the operating rules for certain (or all) data items in the enterprise. Concurrency control at the local level remains the responsibility of the local DBMS or the local database server at an LAN.

The new criterion reduces or even eliminates the need for a central controller, depending on the rules used. The rules can be tailored to differentially provide a variable level of consistency for different data items; while, in contrast, serialization presumes all data be equally critical and need complete synchronization at any point in time. Moreover, the rule-based schema allows for a traditional mechanism to be employed at a particular application site of the multiple systems. For data items that require traditional correctness, operating rules can be developed to route them to a serialization controller which can be included in the system on an optional basis. The global consistency as measured against the new criteria is therefore achieved in two stages: the correct modeling of the control rules to represent the logic correctness, and then the execution model to ensure the accurate implementation of these rules and hence the logical correctness.

The architecture facilitates flexibility. Instead of having the applications changed to fit a control method (e.g., serialization), we use the knowledge about the model (i.e., operating rules and integrity rules) in the metadatabase to build customized control shells around each application. These shells can then operate independently and concurrently with their own control knowledge according to the event/usage criteria; the control knowledge itself is coordinated at the metadatabase. The functionality of each shell depends on the specific knowledge of particular applications, therefore, each shell is in essence different. However, we can create these shells in a manner that allows their basic structure to be identical, with the differences only coming from the specific knowledge they process. Application Systems will be added, deleted or altered through the metadatabsae and the corresponding additions, deletions, or modifications of shells.

The shells must also be able to (1) communicate with each other in a concerted global behavior, (2) react to changes in the local applications that have global repercussions, and (3) process the knowledge they receive from the metadatabase. In addition, the shells should be efficient to implement, change and execute; otherwise, why should one have shells if it is easier to modify the application system? ROPE - Rule-Oriented Programming Environment - is the method developed to satisfy these requirements.

3.2 The Model and Functionality

The ROPE method develops the shells technology needed for the model-based architecture of multiple systems. It defines: (1) how the rules are stored in the local shells or database servers (representation and format used), (2) how the rules are processed, distributed, and managed, (3) how each shell interacts with its corresponding application system, (4) how the different shells interact with each other, and (5) how the shells are designed (i.e., their architecture). Furthermore, the ROPE approach prescribes three principles: (1) rules representing processing logic are separated from the program code by placing them in a distinct rulebase section for easy modifiability, (2) communications among shells are conducted through a message system (typically in an WAN environment) or a database server (typically in an LAN environment), and (3) the rule processing and the communicating methods are combined into the shells. As such, the local shells are invisible to the users, but control the "external" behavior of the local systems. The functionality of ROPE is described below, which serves as the design goals for the ROPE structure discussed in the next two sections.

System Integration

In ROPE, the information management capabilities are achieved in three ways. First, ROPE processes global queries at local nodes. Figure 3 shows how data and queries flow across the networked shells. The global query system of the MDBMS (metadatabase management system shell; circle in the center) sends requests for local queries to be executed by the different shells. The resulting data are assembled by the global query system. The different local shells can also initiate a request for a global query to be processed at the metadatabase management system shell. The general process is as follows: a user initiates a global query, either from MDBMS or from a global query interface located in the local application environment, using the Metadatabase Query Language (MQL) (see [7] for a full documentation of MQL). This query is sent to MDBMS, where it is processed; its syntax and semantics are validated, and local queries and result integration information are generated. The local queries are then sent to their respective shells, where they will be executed by the local DBMS. The results are then sent back to MDBMS for assembly and post-processing.

The second class of information management capabilities through ROPE is the global updates and event-based information flows among local databases; which involves direct use of the rulebase sections of ROPE shells but does not require initiation, nor control, from the MDBMS. In this case, the local shells or database servers monitor the events happening in an application and the database(s) they are concerned with. Rules are triggered based on the status of the application or the database(s). These rules may generate automatic update commands to be performed by the other shells or database servers and query other databases to update their local databases.

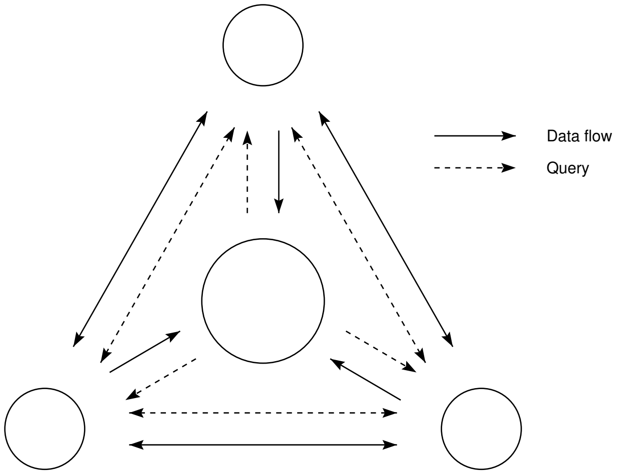


Figure 3 : Global Information Processing Figure 4 : Rule Updating Process

Adaptability and Flexibility

ROPE assures adaptability by (1) fully modularizing the global behavior of local systems into Metadatabase-supported individual shells, by (2) storing its knowledge in the form of rules in the shells or database servers, by (3) directly processing these rules in local languages or server languages, and by (4) automatically updating these rules whenever their logic is modified at the metadatabase. As such, the information processing logic (semantic integrity rules and contextual knowledge) of the multiple systems is managed via rulebase processing. This approach allows the shell to be easily adapted to new situations and provides a flexible interface between thedifferent applications and the metadatabase management system. Any change in the global behavior of local systems as represented in the contents of the metadatabase will trigger the generation of updated rules or new rules by the metadatabase management system in a neutral form, which is then propagated to all pertinent shells and implemented into the rulebase section of the local shells or database servers in local forms by ROPE (see Figure 4). Only metadata processing is involved in this management of rules.

Fault Tolerance and Real Time Process Control

The metadatabase-defined *data replication* is used as a mechanism by ROPE for both fault tolerance and real time process control. We discuss fault tolerance first. The basic objective of fault tolerance is to prevent the operation of local systems from being disrupted by the failure either at other systems (belonging to the same metadatabase cluster or other WAN-/LAN-based clusters in the distributed metadatabases environment) or at the network. Since both the contents and locations of these rules that are interdependent are known to the metadatabases, the scope and maintenance policies of data to be replicated in either synchronous or asynchronous (or periodic) way for a metadatabase cluster can be determined at any time. They constitute a reference table(s) to be stored and managed under an MDBMS for interdependent rules. A metadatabase cluster may not have such a reference table if its rules do not depend on rules in other metadatabase clusters. This mechanism of select data replication is useful for enhancing data availabilities and can also be used to improve performance. There are a few results in the literature that provide ready fault tolerance techniques to be used to implement the design [4,19]. The basic process of the creation, maintenance, and use of the replicated data is shown in Figure 5.

As for real time process control, data replication is used to convert rules that are not self-contained into ones that are. These replicated data will be created and updated on a batch basis as a part of the set-up process of the real time system (e.g., machining workstation). Furthermore, dynamic priorities of event processing (*preemptive scheduling*) are incorporated for sufficiently supporting the needs of real time process control. A real time ROPE shell design based on the execution model has been reported in [23].

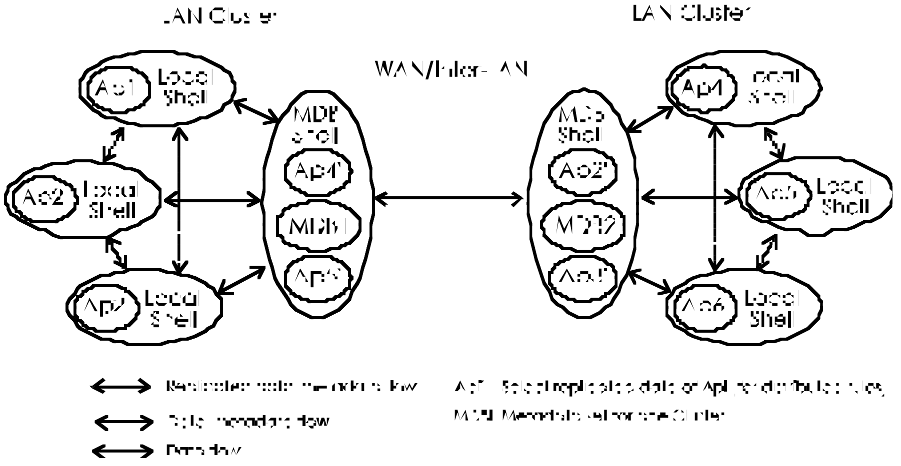


Figure 5 : Data Replication for Fault Tolerance between Metadatabases

Finally, we might also note that the scope and contents of data replication are dynamically defined and managed through the MDBMS. In other words, if a rule that requires interoperation with other rules in other systems/clusters is added or deleted at a metadatabase, the reference table(s) will be updated accordingly.

4 THE DESGIN OF ROPE

4.1 The Static Structure of ROPE

The static structure (see Figure 6) of ROPE shells presented in this paper is developed on the basis of client-server technology (including CORBA) and WAN communications. The major elements of the structure are discussed below.

4.1.1 Rulebase Segment

A key element of the shell is the possession of rules. Logically, the Rule Segment implements the intersection of the global rulebase model and the local application logic; physically, it employs a data structure amenable to local software environments. All the rules are originated from and globally managed by the metadatabase. Whenever an operating rule is changed, the change is propagated to all local systems that will be affected by it. Also, when changes occur in the global data model of the enterprise, new data management rules are generated to replace the existing rules in the Rule Segment of different shells. The Rule Segment is presently implemented using

ANSI/ISO trigger/procedure definitions in ROPE. Since the Rule Segment is separate from the source code, this allows for easy implementation on different platforms.

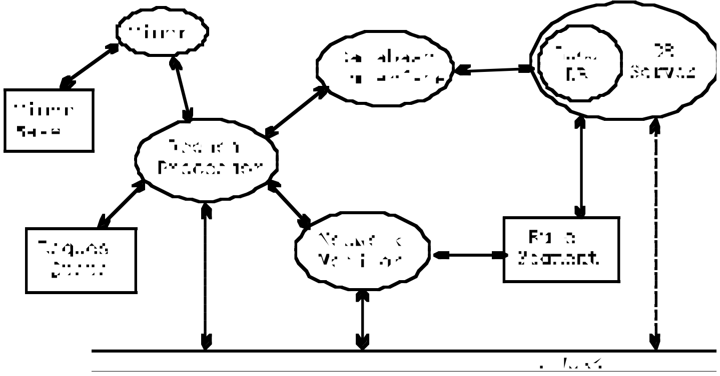


Figure 6 : Local Shell Structure of ROPE

4.1.2 Network Monitor

Network Monitor is responsible for monitoring and processing messages as well as signaling the Request Processor (see below) to take appropriate steps. The Network Monitor is also responsible for sending messages to other nodes. An electronic mailing utility is used by ROPE in WAN environments for message delivery; otherwise, an explicit client-server messaging method (e.g., FTP) is used. This object is tailored to the particular network employed for the local application system, and requires access to the routing information needed to reach the other application systems. ROPE uses another element, Request Processor, to directly interoperate among shells that are located within an LAN environment, to take advantage of the interfacing capabilities provided by database servers.

4.1.3 Timer

The Timer manages a list of time events. When a time event is due for processing, the Timer signals the Request Processor to start processing. In this research, there are three sources of time event: (1) time-triggered rules, (2) predetermined time-out conditions for remote subqueries/ messages processing, and (3) periodic (or asynchronous) data replications. This element further mitigates the impact of specificity in local software and hardware environments for the rest of the shell.

4.1.4 Database Interface

ROPE uses the object to help the local database load Rule Segment into its server, since data conversion is done by the database server. In addition, Database Interface processes the requests coming from both Request Processor and database server. Since rules reside at the database server, the shell will determine whether the client-server messaging method (i.e., CORBA/RPC) or the email method should be used for remote data access, depending on whether the remote site is located within a LAN or a WAN environment, respectively.

4.1.5 Request Processor

This object serves as a request receiver/dispatcher of the shell. All requests from Timer, Database Interface, or Network Monitor will be sent to Request Processor, which then determines the object to which each request should be sent. In LAN environments, CORBA/RPC will be used to allow the Request Processor in one shell to directly signal others in remote shells. Otherwise, signals (or requests) are sent/received as messages by the Network Monitor through WAN communications, before they are processed by this object.

4.2 The Dynamic Structure of ROPE

Based on the static structure, specific algorithms are developed for defining the shell's behavior. Just like the structure itself, these algorithms include both a generic nucleus and an implementation-specific interface to enact appropriately for different application systems. These algorithms suggest how the knowledge needs to be structured and what knowledge is needed for the shells to perform their tasks, hence defining the language requirements. The algorithms and languages constitute the dynamic structure of ROPE.

There are three major areas where new languages are needed to bring about the application system's global behavior through ROPE: (1) creating shells for, e.g., new application systems in the integrated environment, (2) defining the global behavior of the local application systems, and (3) communicating across the different shells.

4.2.1 Shell Definitional Language

Shells are created by using this language, which is completely callable by the metadatabase management system or a user of ROPE. The language first defines a shell in generic terms in a global environment, then uses a code constructor to implement the generic structure in the target local environment. Shell definitional constructs include (1) the system functions defining the elements of the static structure and their attendant algorithms, (2) the system specifications defining the interfaces with the local application, and (3) system parameters defining the requirements of the interfaces and the algorithms. These constructs give rise to a generic shell which is system-independent. This result is then mapped to specific software environments through the code constructor, to allow for maximum portability.

4.2.2 Modeling and Rule Language

The Modeling and Rule Language helps describing the contextual knowledge contained in the metadatabase and the shells. It is an extension of the rule language proposed in [5].

There are two types of actions in the rules: system actions and user actions. The system actions are functions and procedures that are included in every application's shell, as integration tools. They include the routines used by the shell to process and generate the messages that it sends to or receives from other applications' shells. The user actions are all other functions and procedures involved in the rules. The Modeling and Rule Language provides constructs to define: (1) the firing conditions of the rule, (2) the actions to be performed when the rule is fired, (3) the globally significant events (e.g., time event, database event), and (4) how to access the data needed to execute the rule.

4.2.3 Message Protocol and Language

The messages are used to enable communications across the different applications. They are also used to link the metadatabase management system shell with each local system shell.

There is some minimal information that the Message Language should express in order to completely specify a message: the message identifier, the function to be performed upon reception of the message with the necessary parameters, and the origin and destination of the message. The functions can be further classified into metadata and data functions. The metadata functions are used to manage the Rule Segment of the shells. The data functions are used to enable cooperation among shells and with the metadatabase. The message types are determined from the functional tasks of a message.

4.3 Decomposition of Knowledge

The decomposition method is a key for solving the remaining flexible integration problem. It determines which shell or database server will execute a specific action of the global rulebase model so that the distribution of rules into local shells/database servers retains the global consistency and synergism as the original model. With this technical grounding, the seemingly independent processing of each shell can then amount to a consistent global processing without needing the use of a global controller.

The central issue underlying the decomposition is the fact that the condition and action clauses of a rule may contain events and use routines both of which are distributed in different application systems. Thus a rule must granularize its events and routines along with their associated logic in the rule such that each group will be executed in a particular application system, and all groups can be distributed to different systems. Specifically, the process of decomposition takes a rule and produces an equivalent set of subrules that satisfies the following criteria/conditions: (1) the serial behavior of the subrules must be equivalent to the original rule, (2) each element of a subrule is uniquely identifiable, (3) intra-rule data must be separated from inter-rule data to minimize coupling between rules, (4) data items' usage must be exactly determined, and (5) the set of subrules produced must be minimal.

The full details of the decomposition algorithms can be found in [1,2].

5 THE OPERATION AND IMPLEMENTATION OF ROPE

5.1 ROPE Operations

As discussed above, ROPE shells contain rules distributed from the metadatabase. These rules assist in a number of operations belonging to three classes: (1) global query processing operations, (2) rule processing operations, and (3) knowledge management operations. When rules are distributed, the metadatabase shell first examines the global rules and their decomposed subrules to determine whether or not each global rule is entirely distributed within an LAN - that is, its operation involves only the local shells that are located within an LAN cluster. If the local shells involved belong to different LAN clusters, then the WAN convention of data replication will take place and reference tables will be created accroding to the data item types involved in the rules (this information is contained in the metadatabase). Then, it distributes subrules with metadata definitons into local shells. After table/trigger/procedure definitions are generated in each local shell according to the metadata definitions of these subrules, the database server loads and compiles them for future rules execution. The reference tables are updated periodically, while additional maintenance policies are also included on an optional basis.

5.1.1 Global Query Processing Operations

During rules distribution, data conversion rules will first be translated into table and trigger definitions (see Figure 7(a)) for each data item that needs to be converted during operations. Both table and trigger definitions will be loaded into the database server. The trigger definitons are associated to the tables that are used for converting a data to another format. One basic requirement is that the user-defined routines for converting data be created and stored as procedures in the database server, so when data are inserted into the tables, the associated triggers will be fired to run the correspondent procedures that execute the data conversion. Similar preparation is also required for rule processing operations.

During actual operation, each global query is optimized as a set of subqueries [7] by the metadatabase shell. If a subquery involves a shell distributed over an WAN (i.e., in a different LAN cluster), it will then be sent to the shell through electronic messaging method for further processing. Otherwise, a procedure (see the following example) will be created for the subquery in the database server to allow direct LAN data access through client-server architecture (see Section 4.1). In either case, the database server will do the results integration task using a pre-stored procedure created by the metadatabase shell for all subqueries.

Example: A Set of Generated Procedures for LAN-Based Global Query

```
create procedure subquery_1 as
 begin
   create view ops_result as
     select i1,i2,i3
    from ops.order@june.cmp.rpi.edu a,
           ops.item@june.cmp.rpi.edu b
    where a.key1 = b.key2;
   delete from ops.i1_l_to_g@june.cmp.rpi.edu;
   insert into ops.i1_l_to_g (local) @june.cmp.rpi.edu
// trigger will convert data from local to global format
   select i1 from ops_result;
   update ops_result a set i1 =
      (select global
       from ops.i1_l_to_g@june.cmp.rpi.edu b
       where a.i1 = b.local)
end;
```

```
create procedure subquery_2 as
   begin
    ...
   end;
         …
         …
create procedure subquery_n as
  begin
    ...
  end;

// results integration
create procedure global_query as
  create view query_result as
  ... // join of subqueries
  select * from query_result;
```

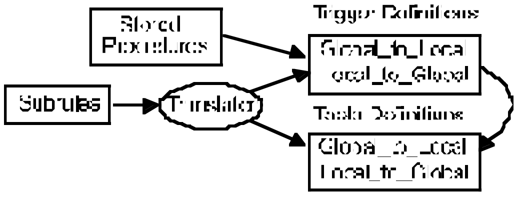
If a subquery fails, a global query processing operation could take one of the following actions for immediate response: (1) show no result, (2) show the partial result, or (3) re-send the subquery. These operating models are selectable by the global query users.

5.1.2 Rule Processing Operations

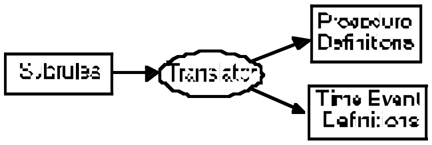
Processing time-triggered rules and data-triggered rules is the most common type of operations in this class. Data conversion rules that are depicted in Figure 7(a) for global query processing operations are also needed for rules processing to achieve consistency of data formats. For time-triggered rules, they will be translated into procedure and time definitions (see Figure 7(b)) during creation. Only the procedure definitions will be loaded into the database server. The time definitons will be stored in Timer. During operation, it is the Timer's task to manage the list of time events that impact on the shell processing. When Timer wakes up, it retrieves all the time events due for processing and notifies Request Processor. Typically, Request Processor will then signal Database Interface to invoke the stored procedures (i.e., rules) in the database server.

As for data-triggered rules, they will be translated into procedure and trigger definitions (see Figure 7(c)). Both definitions will also be loaded into the database server and rules (stored as procedures in the database server) will be fired by triggers when a change in the associated table of the local application database is detected. The basic events are simply the unit operations of database processing: insertion, deletion, and update. Additional events may be defined for particular applications. This event-driven approach uses triggers that are embedded in the system to monitor the local database. For local systems that satisfy certain requirements, there are particular event-driven methods that can be employed for data monitoring [20,25].

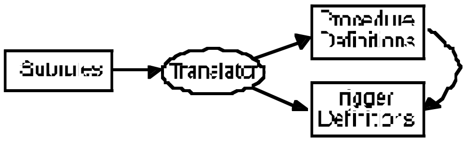
The execution of a global rule is typically completed by executing several subrules which might be distributed within or without an LAN. For the case of LAN, the original shell's Request Processor will directly signal the Request Processor in other shells for susequent subrule execution. Otherwise, the signal(s) will be converted and sent as messages to WAN shells. Again, depending on how metadata are defined, the subrules and related data could be replicated at the LAN site of a metadatabase for the sake of performance and reliability.



(a) Rule Segment Creation for Conversion Rules



(b) Rule Segment Creation for Time-Triggered Rules



(c) Rule Segment Creation for Data-Triggered Rules

Figure 7 : Rules Distribution for ROPE

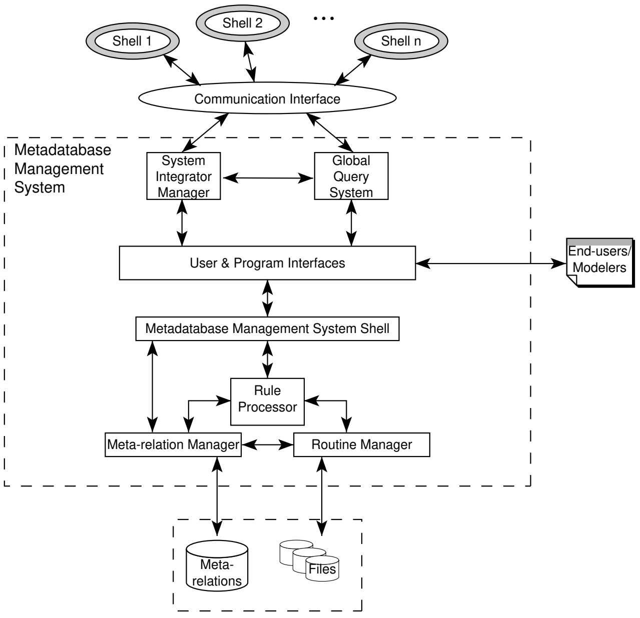


Figure 8 : The Metadatabase Management Shell Architecture

5.1.3 Knowledge Management Operations

We consider two typical scenarios here: (1) structural model modification and (2) Rule modification. Structural model modification includes adding or removing fields or entities/relationships. Once a change is committed to a local database and the metadatabase, a structural model modification message is sent to the local shell. The local shell updates the trigger/procedure definitions of all the rules being triggered by the modified entity/relationship. The updated trigger/procedure definitions will be loaded and re-compiled in the database server that the local shell uses.

Rule modification is supported in ROPE as an (old) rule deletion followed by a (new) rule insertion. The System Integrator Manager of the metadatabase shell (see Figure 8) does not need to keep track of where a rule is located. A rule insertion message contains all the details needed to update the different data of the application shells. The shell will remove any reference, particularly the trigger/procedure definitions, to a rule when receiving a rule deletion message for that rule, enforcing the integrity constraints of the ROPE structural model.

5.2 Laboratory Implementation

A prototype is developed for the model-based architecture. At the core is a metadatabase management system. The major elements of this system are depicted in Figure 8. It performs three major tasks: (1) management of the metadatabase itself, including metadata queries, analyses, and rulebase processing [5]; (2) formulation and processing of global queries, utilizing the metadatabase to provide on-line intelligent support to users [7]; and (3) generation of data management rules as well as operating rules and interfacing with ROPE for event-based information flows and data management [1]. The elements performing these tasks are self explanatory in Figure 8.

The model-based architecture shown in Figure 1 has been implemented along with the metadatabase management system, resulting in a prototype that verifies the design objectives and methods of the proposed model. The performance, scalability, and generality of the results are also studied through the prototype. Much of the prototyping work has been completed at Rensselaer as part of the industry-sponsored Computer Integrated Manufacturing Program (through June, 1992) and the Adaptive Integrated Manufacturing Enterprises (AIME) Program (July, 1992 to June, 1995) [16]. Both programs are sponsored by Alcoa, Digital, GE, General Motors, and IBM. The AIME Program is also funded by the National Science Foundation. At present, an industrial pilot system is being constructed for Samsung and U.S. Army.

The metadatabase management system was initially implemented on a micro VAX platform using Rdb as the database engine for the metadatabase and has been demonstrated publicly in early ‘90. A new version using RS/6000, AIX, and Oracle has later been developed to provide a multi-platform and distributed metadatabase environment [14,16].

At this time, the new extensions to ROPE that manage fault tolerance over WAN and utilize client-server technology to enhance performance are currently under development. Although some objects still need to be implemented, most concepts put forward by the model-based approach have been empirically tested using the metadatabase model. The laboratory environment consists of (1) a product design database using ROSE (an object-oriented database management system) and EXPRESS (the STEP data definition language) on an RS/6000, (2) a shop floor control system using Oracle on a PC, (3) an order processing system using Rdb on a microVAX, and (4) a process planning system using dBase on a PC. ROPE shells were then created and incorporated into these local systems as additional application programs against their own database management systems.

6 CONCLUSIONS

In this paper, we presented (1) the concept of model-based architecture for flexible integration of multiple manufacturing systems based on the principle of metadata independence and (2) its execution model, the Rule-Oriented Programming Environment method, for operations in LAN and WAN environments. This solution is based on the metadatabase model, developed and reported in the field over the past few years. We review below the new extensions of the architecture over the previous results, with respect to the goals of flexibility.

How does the model-based architecture satisfy open system requirements? This issue encompasses interoperability and portability, and underlines a technical basis to flexibility. The open system capacity defined by the architecture is composed of three layers: (1) the local environment layer, (2) the shell layer, and (3) the network layer (see Figure 6). The link between the local environment and the shell layer is achieved by (1) the Database Interface, (2) the Definitional Language, and (3) the Message Language. Because there are but a few connection points, the shells are poised to be implemented in diverse environments. The ROPE shell design assumes some basic and generic database and network capabilities of the local application systems, but are otherwise independent of the implementation environment.

ROPE deploys the model integration knowledge contained in the metadatabase to effect interoperability among the heterogeneous application systems it integrates. Model integration rules are acquired through modeling process. ROPE treats these rules the same as others and hence implements these event/usage criteria faithfully up to the correctness of their modeling as rules.

The concurrent processing method and the event/usage concurrency control approach make it possible to fulfill the need for local system independence. The ROPE shell provides an interface between the enterprise knowledge and the application systems. It is used to translate any result from the global knowledge processing into a format suitable for the local application systems, and vice versa. The distributed knowledge processing and management method enables the shells to communicate partial rule processing results as well as the metadatabase to transmit new or updated knowledge to the shells. The decomposition algorithm is designed in a holistic view of knowledge processing in a distributed environment, meeting the requirements of such a task. Furthermore, the shell's linkage to the local application and the network is minimized by ROPE. However, after that linkage is accomplished, the application has the option of tapping into the ROPE capabilities as an extension to its own environment.

Furthermore, how does the model-based architecture addresses the problems discussed in Section 2 that are critical to WAN-based interoperations?

1. Global serialization. The model-based architecture employs a new criteria, event/usage correctness [13], for control and thereby removes the need for global serialization and fosters full concurrent processing among local systems resides at either WAN or LAN cluster.
2. Fault-tolerant processing and real time process control. The model-based architecture uses metadatabase-defined data replication as a mechanism for both purposes. Since both contents and locations of rules that are interdependent are known to the metadatabase, the

scope and maintenance policies of data to be replicated in either synchronous or asynchronous (or periodic) way for a metadatabase cluster can be determined at any time. Therefore, data availability can be enhanced and reasonable performance can be achieved for both real time and non-real time systems.

Finally, since the new ROPE design presented in this paper is based on client-server technologies such as ANSI/ISO SQL3 and CORBA, its portability is supported by these industrial standards. As a result, the model-based architcture promises a reasonable scalability for both LAN- and WAN-scale systems.

REFERENCES

1. Babin, G., *Adaptiveness in Information Systems Integration*, Unpublished PhD Thesis - Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, N.Y., August 1993.
2. Babin, G. and C. Hsu, “Decomposition of Knowledge for Concurrent Processing,” *IEEE Transactions on Data and Knowledge Engineering*, forthcoming.
3. Barber, K. and M.T. Özsu, “Concurrent Transaction Execution in Multidatabase Systems,” *COMPSAC 1990*, IEEE Computer Society Press, Los Alamitos, CA, pp. 282-294, October 1990.
4. Birman, K.P., “The Process Group Approach to Reliable Distributed Computing,” *Communications of the ACM*, vol. 36, no. 12, pp. 36-53, 1993.
5. Bouziane, M., and C. Hsu, “Rulebase Modeling for Data and Knowledge Integration in Multiple System Environments”, *International Journal of Artificial Intelligence Tools*, 2(4), pp. 485-509, 1993.
6. Buchmann, A.P., “Modeling Heterogeneous Systems as an Active Object Space,” *Fourth International Workshop on Persistant Object Systems*, A. Dearle, G.M. Shaw, S.B. Zdonik (Eds), Morgan Kaufmann Publishers, San Mateo, CA, pp. 279-290, 1990.
7. Cheung, W. and C. Hsu, “The Model-Assisted Global Query System for Multiple Databases in Distributed Enterprises,” *ACM Transactions on Information Systems*, forthcoming.
8. Chung, C.-W., “DATAPLEX: An Access to Heterogeneous Distributed Databases,” *Communications of the ACM*, 33(1), pp. 70-80, January 1990 (with corrigendum in *Communications of the ACM*, 33(4), p. 459, April 1990).
9. Desai, B.C. and R. Pollock, “MDAS: heterogeneous distributed database management system,” *Information and Software Technology*, 34(1), pp. 28-42, January 1992.
10. Holtkamp, B., “Preserving Autonomy in a Heterogeneous Multidatabase System,” *COMPSAC ‘88*, IEEE Computer Society Press, Washington, DC, pp. 259-266, October 1988.
11. Horowitz, J.R. and A.F. Cardenas, “Decomposing Heterogeneous Inter-Entity Relationship Updates,” *IEEE Transactions on Knowledge and Data Engineering*, 4(4), pp. 360-370, August 1992.
12. Hsu, C., “Enterprise Integration and Modeling: the Metadatabase Approach,” Kluwer Academic Publishers, Boston, 1996.

13. __________ and _______, “A Rule-Oriented Concurrent Architecture to Effect Adaptiveness for Integrated Manufacturing Enterprises,” *International Conference on Industrial Engineering and Production Management*, Mons, Belgium, pp. 868-877, June 1993.
14. __________ , _______, M. Bouziane, W. Cheung, L. Rattner, A. Rubenstein, and L. Yee, “The Metadatabase Approach to Integrating and Managing Manufacturing Systems,” *Journal of Intelligent Manufacturing*, 5(3), pp. 333-349, 1994.
15. __________ , M. Bouziane, L. Rattner, and L. Yee, “Information Resources Management in Heterogeneous, Distributed Environments: A Metadatabase Approach,” *IEEE Transactions on Software Engineering*, 17(6), pp. 604-625, June 1991.
16. __________ , L. Gerhardt, D. Spooner, and A. Rubenstein, “Adaptive Integrated Manufacturing Enterprises: Information Technology for the Next Decade,” *IEEE Transactions on Systems, Man, and Cybernetics*, 24(5), pp. 828-837, 1994.
17. __________ , Y.-C. Tao, M. Bouziane and G. Babin, “Paradigm Translations in Integrating Manufacturing Information Using a Meta-model: The TSER Approach,” *Information Systems Engineering*, France, 1(3), pp.325-352, 1993.
18. Kim, W., editor. “Modern Database Systems,” ACM Press, 1995.
19. Ladin, R., B. Liskov, L. Shrira and S. Ghemawat, “Providing Availability Using Lazy Replication,” *ACM Transactions on Database Systems*, vol. 6, no. 2, pp. 213-226, 1992.
20. McCarthy, D.R. and U. Dayal, “The Architecture of An Active Data Base Management System,” *Proceedings of the 1989 ACM SIGMOD*, ACM Press, New York, NY, pp. 215-224, 1989.
21. Object Management Group, “The Common Object Request Broker: Architecture and Specification,” ACM Press, 1995.
22. Perrizo, W., J. Rajkumar, and P. Ram, “HYDRO: A Heterogeneous Distributed Database System,” *ACM SIGMOD 1991*, pp. 32-39, May 1991.
23. Shaefer, O. and C. Hsu, “Distributed Rulebases for Real Time Process Control on the Shop Floor: The Metadatabase Approach,” *ASME MH-vol.2, 75 Anniversary Commemorative Volume on the Materials Handling Engineering Division,* pp 49-59, November 1994 .
24. Shan, M.-C., “Pegasus architecture and design principles,” *ACM Sigmod,* pp. 415-423, October 1992.
25. Stonebraker, M., “The Integration of Rule Systems and Database Systems,” *IEEE Transactions on Knowledge and Data Engineering*, 4(4), pp. 415-423, October 1992.