# A Model-Based Architecture for Adaptive and Scalable Multiple Systems:

Gilbert Babin[1]
Cheng Hsu[2]

September 1994

Submitted to *ACM* Transactions on Database Systems

**[1] Assistant Professor, Department of Computer Science, Université Laval, Quebec City, Quebec, Canada, G1K 7P4.**
**[2] Associate Professor, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY, 12180-3590.**

ABSTRACT

The Integration and management of multiple data and knowledge systems entail not only interoperability, local autonomy, and concurrent processing, but also the new capabilities of adaptiveness allowing scalable (or incremental) integration and non-standard or legacy systems, while avoiding the well-known performance problems caused by the traditional approaches to global control and management. The past decade has seen many good solutions developed from vast worldwide efforts for many aspects of the problem; yet certain needs remain unsatisfied — especially the new capabilities of adaptiveness. Most of the previous efforts have focused either on providing an architecture for direct interoperation among different systems (such as CORBA, Common Object Request Broker Architecture), or on developing a global model to manage these systems in the tradition of databases (such as heterogeneous and distributed DBMS's). It seems that a promising approach to the problem is making the interoperable architecture adaptive and scalable by virtue of defining it throught the global model; or, simply, the model-based architecture.

Such an architecture is proposed in this paper, using the metadatabase model. The concept of metadata independence for multiple systems is presented as the basis for the conceptual design of the architecture, while the Rule-Oriented Programming Environment (ROPE) method is developed to execute the design. ROPE implements global (integration) knowledge into localized rule-oriented shells which constitute the control backbone of the interoperating architecture, and manages these shells. The metadatabase enables the shells to grow in number as well as to change its control knowledge contents. A prototype is developed to test the concept and design.

Category and Subject Descriptors: H.2.4 [**Database Management**]: Systems — *concurrency*, H.2.5 [**Database Management**]: Heterogeneous Databases, H.2.7 [**Database Management**]: Database Administration, J.1 [**Administrative Data Processing**]: Manufacturing.

General Terms: Design, Management.

Additional Key Words and Phrases: Heterogeneous and Distributed Databases, Information Integration, Concurrent Architecture, Metadatabase.

ACKNOWLEDGEMENTS

This research is supported in part through Grant #DDM-9015277 and Grant #DDM-9215620, National Science Foundation, and the industrial sponsors of Rensselaer's CIM and AIME programs, which include Alcoa, Digital Equipment Corp., GE, General Motors, and IBM.

# 1. THE MANAGEMENT OF MULTIPLE SYSTEMS IN DISTRIBUTED ENTERPRISES

Enterprise information management in significant organizations is characterized with multiple data and knowledge systems operating over wide-area networks. The hallmark of these multiple systems is their defiance of traditional control models: they do not abide by any single standards, are not amenable to centralized administration, and cannot be comprehensively modelled for the entire enterprise at once. A prime case would be a modern manufacturing enterprise, where a single factory could easily include over a thousand of databases running at a scale of over a million transactions a day. The complexity is overwhelming: in addition to such well-known issues of heterogeneous databases as interoperability and distributed updates, this environment also faces rapid changes in its underlying (data and knowledge) technology, business processes, and applications. Thus, the unpleasant reality of legacy systems and heterogeneity will always stay. Progress on standards would hardly make the problem fade away, since today's standards will become tomorrow's legacies in the face of new cutting edge technologies, which tend to transcend any standards that require a long time to develop or to take effect. On top of this technical reality, organizational reality such as budget, politics, and user acceptance also tends to make the introduction of information systems an ever-changing and never-stopping process. These systems cannot be developed, nor standardized, once and for all.

Therefore, the nature of enterprise information systems is inherently dynamic; which clearly requires an adaptive and scalable architecture for their integration and management. In this context we present our work.

## 1.1. Basic Challenges in Previous Research

Most of the previous efforts reported in the literature are naturally concerned with the first order of business in integration; namely, the solution for heterogeneity, distributedness, and autonomy. Many of these results offer a solution for a fixed point in time (e.g., all requirements are known). The need for the solution's integration architecture to be adaptive and scalable seems to be slated as beyond the immediate concern. Although this focus on a fixed architecture is sensible when considering the urgent needs in practice for industrial-grade solutions, it nevertheless also leads to a need today of extending this scope into developing architectures which support integration over time, especially in the face of some well-known challenges to these solutions.

We might first mention interoperability solutions such as Common Object Request Broker Architecture (CORBA) of the Object Management Group and the Distributed Applications Environment of IBM and the Open Software Foundation. Indeed, the natural approach to integrating multiple systems in open architecture environments is to develop a common protocol acceptable to most applications. CORBA and other industrial solutions available today, however, are not sufficiently open as to allow change and growth to a particular architecture once constructed; at least not without serious disruption, recompilation, or restructuring. As a matter of fact, well-known difficulties with the present technology of CORBA (in a broad sense) include the lack of a global view of the knowledge encapsulated into objects and the inflexibility in changing the methods and triggers which represent the knowledge. Without such a global model of knowledge (and objects distributed in individual local systems) and the ability to alter the knowledge, on which the interoperating architecture is ultimately based, an adaptive and scalable

architecture can hardly result. CORBA and other results can be improved and enhanced towards this end, of course. The new approach attempted in this work might be helpful in dealing with the knowledge problem, as well as stands as a metadatabase solution of its own..

The global modeling and management of multiple systems in the tradition of databases is another fertile area in the literature. Philosophically beyond the concept of interoperability, the area is characterized with a strict vision of multiple databases which are globally controlled under a tight regime and perceived by users as a whole (or even a single system). A global schema (or its variants) developed from the global modeling effort is a defining element of integrating architectures resulting from efforts in this area. Many hard problems in databases and distributed computing have been elegantly addresses under a variety of labels, including multidatabases [1, 3, 7, 18 21, 22] or simply heterogeneous and distributed databases [4, 5, 8, 9, 12, 14, 16, 23, 35, 37, 40, 41].

From the perspective of this work, we might notice a few salient points which shed light on the expansion of previous scope. A basic task of all these systems is global query — in fact, many of the previous efforts focus exclusively on this functionality — examples include the venerable Multibase from the Computer Corporation of America [14, 37], the Carnot system from Microelectronics and Computer Technology Corporation [13], the SIRIUS-DELTA system [19], the IMDAS system from University of Florida [39], the HD-DBMS system from UCLA [9, 23], Mermaid from UNISYS [40], the EDDS system from the University of Ulster [5], The PRECI* system [15], DATAPLEX from General Motors Research Labs [12], the InterBase system from Purdue University [18], the MDAS system from Concordia University [16], and DQS from CRAI (Italy) [4]. Active data management tasks, especially updates, pose tremendous challenges to distributed computing technology, and systems that suffice these tasks in large scale and dynamic environments (such as manufacturing) in actuality are hard to come by. The second key observation is the development of shell-based architecture to manage the global and sometimes local queries (or even the active tasks) [17, 42]. Among them are the concept of using rule-based technology to develop these shells (first proposed by Hsu and Rattner [25]) and the employment of active objects (by [8]) and agent (by HP's Pegasus[1]) to execute autonomously events for information flows and transactions. The incorporation of knowledge into the architecture used is a valuable revelation to this work. A new execution model would turn the shell into an intelligent agent of interoperability. The knowledge method needed is one that, on the one hand, implements the concept of concurrent rule-oriented shells to manage the events and, on the other hand, manages these shells themselves according to a global model. This kind of execution model seems to be particularly interesting when the shell-based architecture is also envisioned to support active data management tasks.

The above observations can be summarized into this framework: there are three natural categories of work for adaptive and scalable integration of mutiple systems. The first is interoperability among different systems as exemplied in CORBA and other industrial protocols and standards. The second is global administration of these systems to provide the classical functionalities of databases. Both have enjoyed the benefits of outstanding results from worldwide research efforts. The third, however, is the less-aware feedback from (the global model that enables) the administration to (the architecture that facilitates) the interoperability. In particular, the model can or even should be used directly in the definition and construction of the architecture and thereby bring about the ability to change the architecture through changing the model. The feedback loop would close up the process of integration as any adaptive system requires. We refer to this concept as the model-based architecture. The Metadatabase Model [24, 26, 27, 28] is

among the works that have been concentrating on the third category. As will be reviewed in the next section, the metadatabase results provide a model-based architecture for adaptive and scalable multiple systems based on the feedback principle. There have also been new technologies developed in the past several years to support the modeling, the management of models, the use of models for global query, and other key tasks in the approach. However, a full fledge execution model based on new knowledge methods accountable for responding to the challenges mentioned above is lacking previously. Developing such an execution model and completing the proposed architecture, thus, are the objectives of this work.

## 1.2. The Research Problem

The model-based architecture calls clearly for certain specific requirements to be satisfied by the new methods. First is system autonomy: what is demanded of all local systems in order to evolve the architecture (in particular, the integrated schema, the processing knowledge base, and the shells) when some systems are changed, deleted, or added? From our perspective, a system is fully autonomous when it does *not* need to comply to any of the following: (1) conforming to an integrated schema (e.g., converting a legacy schema to some global standards), (2) directly cooperating with any global controller (e.g., serialization program or any direct supervision of transactions and their communications), and (3) requiring users to possess specific knowledge about the global environment in which the local systems operate (e.g., cannot use the local system in the same way with the integration as it would be without). Complying with these requirements will obviously impede on the adaptability and scalability.

Second is performance, which is directly related to the lack of concurrent processing of local systems. Regardless of how much pipelining they offer for local systems, virtually all of the previous results which manage data updates call for global serialization at a certain point. Factors compounding this bottleneck include network scaling problems and distributed query and transaction processing. Since these results are mostly concerned with small scale networks with a limited number of transactions, there are questions concerning whether or not they work in larger and widely dispersed environments. At the heart of the distributed processing issue is the most accepted criterion for achieving consistency among multiple concurrent jobs, the von-Neumann model of serialization. This criterion is driven by the objective of assuring instantaneous correctness for all data in the entire system [20], which is appropriate (or even necessary) for single-database systems. However, when applied to multiple systems in distributed enterprises, this objective is at least overzealous and may be outright unattainable. With a greater demand for current processing and real-time data availability in adaptive and scalable architecture, a new criterion for data correctness needs to be employed [25, 34].

Accordingly, the proposed architecture has three major elements: (1) a metadatabase (and its management system) serving as the knowledge base for enterprise integration, (2) a concurrent shell system using the metadatabase for integration, and (3) a rule-oriented programming environment (ROPE) for implementing and managing the concurrent shell system [2, 29, 30]. The third element constitutes the execution model which is the research problem for this paper. The basic objective of ROPE is to decompose and distribute the pertinent contextual knowledge (for both event-based information flows and data management) contained in the metadatabase into localized shells, and thereby effect concurrent processing and facilitate architecture adaptability and scalability under distributed and autonomous control. Therefore, the research problem is technically concerned with *concurrent knowledge representation and processing*. This knowledge

method must be able to (1) control the application logic of local systems without the use of a central controller, (2) enhance the local applications with a distributed knowledge capability for global behavior without changing them, (3) transfer the needed knowledge both between the metadatabase and the local applications, and among the local applications themselves, and (4) allow implementation of global knowledge in a clustering of shells which share a common information model and can be added or deleted independently.

### 1.3. The Organization of the Paper

The concept, background, and the problem for the proposed architecture are discussed above in this section. An original analysis for the notion of metadata independence which provides the philosophical basis for the solution approach is presented in Section 2. On this basis, the metadatabase model is reviewed to provide the conceptual design of the model-based architecture in the same section. New contributions, namely the ROPE methods, are presented in Section 3; which delineates the shells system, static and dynamic structures, and the decomposition and distribution of contextual knowledge of ROPE. Section 4, then, illustrates the operation of ROPE for such basic tasks as concurrent rules processing, knowledge management, changes to enterprise models, and addition of new or legacy systems. Together, these two sections (3 and 4) present the execution model. A prototype verifying the architecture in a computer-integrated manufacturing enterprise laboratory at Rensselaer is discussed in Section 5, along with an analysis of its performance, functionality, and implementation requirements. The last section concludes the paper. We might mention that this paper reports original work centered around ROPE, which completes the architecture, and presents the entire solution for the first time. ROPE in its own right adds a new rule-based method for interoperability (in the same category as, e.g., CORBA). This potential is commented in Section 6. Specifically, all of Sections 1, 3, 4, and 6 and much of Sections 2 and 5 represent new contributions.

## 2. THE METADATABASE APPROACH TO ADAPTIVE AND SCALABLE INTEGRATION

The metadatabase model is intended to effect a solution approach that converts the integration problem from one that deals directly with data instances to one that controls through metadata, and thereby seeks to simplify the complexity of the problem and avail adaptability and scalability for multiple systems. The key ideas are analyzed below.

### 2.1. The Notion of Enterprise-Level Metadata Independence

Succinctly, the unique requirements of multiple systems in enterprise information management may be stated as follows:

- Scalability. The total enterprise information integration environment must allow incremental development and be expandable, such that the integration can start with a small part of the enterprise and gradually extend to the rest (even to other organizations) over time, without loosing operational continuity and structural integrity.
- Adaptability. Systems that use either standard or non-standard technologies, as well as new and legacy systems, can be incorporated into the integrated environment in a seemless way without causing any disruption to any existing systems.

- Parallelism. The multiple systems must be able to operate concurrently while achieving synergism for the enterprise, without requiring global serialization or similar synchronization mechanisms imposed on any instance-level transactions.
- Autonomy. Local systems in the integration need to have the flexibility to be designed, constructed, and administered independently by the local management alone, without having to conform, nor later convert, to a global schema.

An interesting observation of the above requirements is the revelation that they are fundamentally identical to the classical concept of data independence using the three-schema architecture; the difference is the "primitive", or primary concern, in each: At the (single-site) database level, the primary concern is multiple applications processing data instances; whereas at the enterprise level, the primitive is multiple (database and knowledge-based) systems processing applications. Consider the fact that systems are modeled and hence substantiated with metadata, then it is evident that the enterprise-level problem can be formulated as a metadata problem whose conceptual complexity, in metadata terms, is similar to the traditional database-level problem. Therefore, in the spirit of data independence for scalable, adaptable, parallel, and autonomous applications against a database, we refer to the enterprise-level requirements of scalable, adaptable, parallel, and autonomous databases as *metadata independence*. The search for a solution to the enterprise information management problem is, it follows, focused on transforming the data problem into a metadata problem and bringing the proven model of databases to the enterprise level, thereby effecting a metadata independent architecture to simplify the complexity [2, 6, 11, 25, 26, 27, 28, 31].

Where should the search begin? Traditionally, database researchers all cherished three principles: the use of a (global) data model, the reliance on an integrated schema, and the enforcement of global serialization. This tradition has been carried on throughout the myriad of efforts in developing distributed databases, federated databases, and multidatabases; and is still dominating in many the latest endeavors of integrating multiple data and knowledge systems operating in heterogeneous environments across wide-area or even worldwide networks. Although a great deal of progress has been accomplished in the past decade on, e.g., interoperability, local autonomy, and open system architecture, a great deal more still remains to be accomplished; which centers especially around the issues of *concurrent processing* and *architecture adaptability*. Other aspects of metadata independence mentioned above are also based on these two technical issues. Consider the previous example of a modern manufacturing enterprise (see Section 1), multiplying its thousand-databases factory by a factor of ten and linking them through a global network, and the significance of these two issues becomes immediately evident. Scalability as defined above follows directly from these two key capabilities. Together, they are referred to in this paper as the adaptive (and scalable) integration problem. We propose a solution to this problem entailing the following basic elements transforming the three database principles into enterprise-level to achieve metadata independence:

(1) An enterprise information model: this model globally represents all local data models and their contextual knowledge in the enterprise with a metadata-independent structure which, when put online, allows all local models and other metadata contained in it to be added, deleted, or modified through ordinary metadata transactions (as opposed to a fixed global data model);

(2) An online (independent, but sharable) metadatabase: this metadatabase implements the enterprise information model, and comprises a scalable hierarchy of mini-

metadatabases for any scope of local functions in a (expandable) client-server manner (as opposed to schema integration); and

(3) A concurrent shell-based system for execution: this execution model supports concurrent processing of local systems with localized distributed control knowledge (as opposed to global serialization).

Together, they define an adaptive and scalable architecture as a solution to the problem. The major elements of the metadatabase model that pertain to the structure of the architecture are reviewed next.

## 2.2. The Basic Structure: A Metadatabase-Supported, Rule-Oriented, Concurrent Shell System

The structural model of the architecture is depicted in Figure 1 [25, 28, 30, 31]. 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.

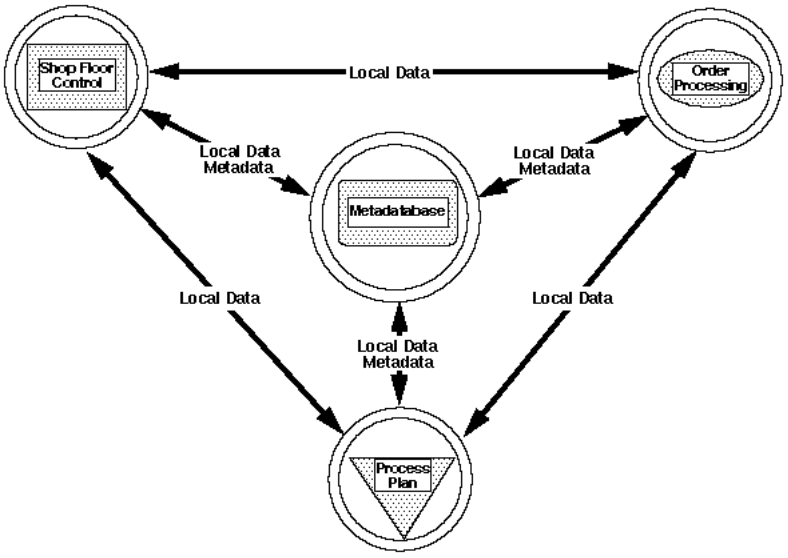



Figure 1 : The Concurrent Architecture Using a Metadatabase

The metadatabse itself employs a generic meta-structure, the Global Information Resource Dictionary (GIRD) model (see Figure 2), abstracting the enterprise metadata resources (i.e., models). As such, each and every local model is represented (but not duplicated, nor removed) into the metadatabase as ordinary metadata “tuples” populating the structure. The metadata independence at the model integration level is achieved since any change in enterprise models

would involve only ordinary metadata transactions similar to the usual relational processing, and do not require any change to the structure itself, nor reloading/recompilation. The structure itself, as well as the representation of local models, is developed according to the Two-Stage Entity Relationship (TSER) method, which serves for this purpose as a meta-model. TSER provides at the first, functional stage two basic representation constructs: SUBJECTS (similar to objects in object-oriented paradigm) and CONTEXT (rule-based description of process and other contextual knowledge). At the second, structural stage, ENTITY (characterized with singular keys) and RELATIONSHIP (characterized with composite keys) plus two more types of associations signifying special integrity rules (functional and mandatory) are defined. All these constructs, combined with hardware and software classes of metadata, are included in the GIRD model. (The above description suffice the purpose of this section; the full details are provided in [27].)

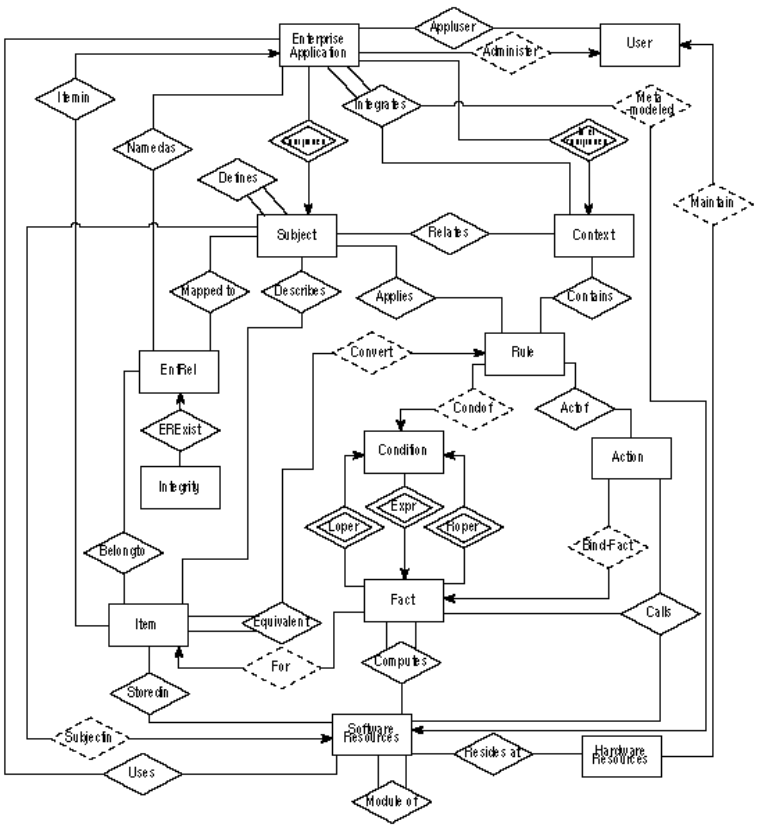



Figure 2 : The GIRD Meta-Model

A meta-model system is developed to support the modeling and creation of the metadatabase — see [32]. A full reference for TSER can be found therein.

## 2.3. The Metadatabase-Supported Scalable Integration Hierarchy

In large-scale systems, the metadatabase can be distributed in a recursive manner and constitutes a hierarchy of nodes [31], illustrated in Figure 3, where, for simplicity, only three levels are shown. At the leaf, several of the application systems are represented in a sub- or mini-metadatabase. This mini-metadatabase then can be represented in the main metadatabase system such that the meni-metadatabase becomes the gateway to the application systems it represents.

There can, of course, be as many levels of sub-/mini-metadatabases as needed. The real significance of this hierarchy, however, is not its top-down construction, which is the predicament of virtually all other integration models; but rather its ability to effect bottom-up, incremental development and expansion — i.e., scalable integration: the higher level nodes can be constructed from the ones immediately below it (vertical), and additional branches or nodes can be added to the same level and concatenated with the existing hierarchy (horizontal). A key element in this bottom-up, scale-up approach is the GIRD meta-model discussed above. This structure allows new application models to be incorporated into a metadatabase without disrupting the current systems. Thus, large scale, overwhelming integration environments can be achieved gradually, by first developing individual nodes in client (application systems) - server (metadatabase) type of clusters and then integrate them in a grand clustering of such clusters. In a similar way, the main metadatabase can be incorporated into a super or inter-enterprise metadatabase and become a mini-metadatabase of the latter. Both the clusters of metadatabases and the clusters of application systems at the leaf nodes are managed by the same execution model.

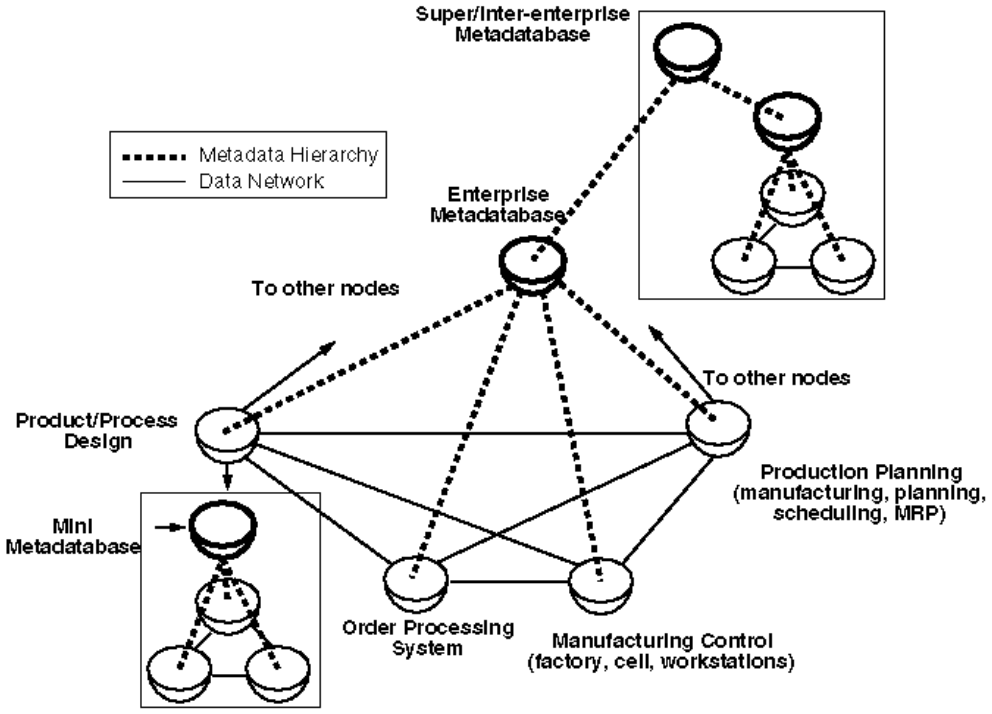



Figure 3 : Enterprise Integrated Through Distributed Metadatabase System

A complete management system is developed for the metadatabase (signified as the center shell in Figure 1). The major elements of this system are depicted in Figure 4. It performs three major tasks: (1) management of the metadatabase itself, including metadata queries, analyses, and rulebase processing [6]; (2) formulation and processing of global queries, utilizing the metadatabase to provide on-line intelligent support to users [11]; and (3) generation of data management rules as well as operating rules and interfacing with ROPE for event-based information flows and data management [2]. The elements performing these tasks are self explanatory in Figure 4.

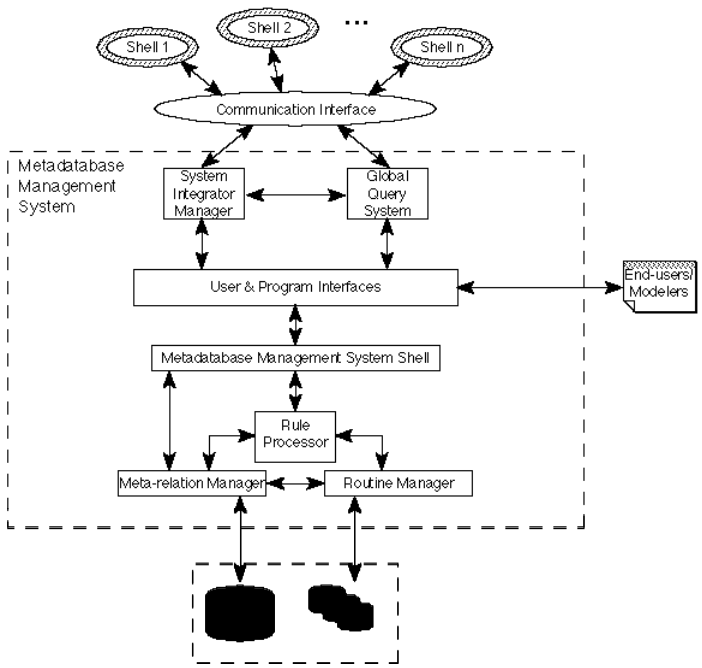



Figure 4 : The Metadatabase Management Shell Architecture

## 2.4. The New Method for Concurrent Knowledge Representation and Processing

The metadatabase model employs a new criterion, event/usage correctness [29, 30], and thereby removes the need for the global serialization and fosters full concurrent processing among local systems. This argument is closely related to some earlier observations [22], which argue that serialization is impractical if not impossible in multidatabase systems since there is a higher probability of deadlock due to the network.

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 *requirements* 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. From ROPE perspective, concurrency control at the local level remains the responsibility of the local DBMS.

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, whereas serialization presumes all data be equally and completely synchronized at any point in time. Moreover, a traditional mechanism can 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 metadatabase model (Figure 1) is employed as the basic structure to develop the new method required for the execution model. 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.

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? ROPE is the method developed to satisfy these requirements. The proposed architecture is completed with the development of ROPE.

## 3. ROPE: THE EXECUTION MODEL FOR THE ARCHITECTURE

The ROPE method develops the shells technology needed for the adaptive and scalable architecture of multiple systems. It defines: (1) how the rules are stored in the local shells (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 (e.i., 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, and (3) the rule processing and the message system are implemented into local environments and combined into the shells. As such, the local shells are invisible to the users, but control the “external” behavior of the local systems. These elements are fully described in this section.

We first provide a functional description of ROPE from the perspective of the enterprise information systems to illustrate the technical context of its elements.

### 3.1. The Model of ROPE

#### *3.1.1. Global Query Processing*

In ROPE, the information management capabilities are achieved in three ways. First, ROPE processes global queries at local nodes. Figure 5 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.

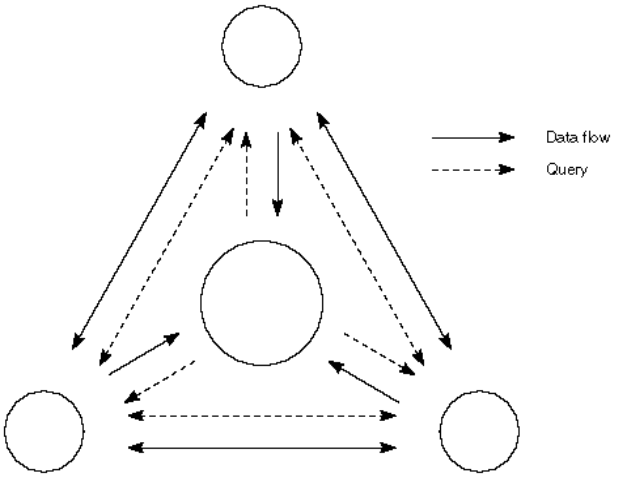



Figure 5 : Global Query Processing

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 [11] 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.

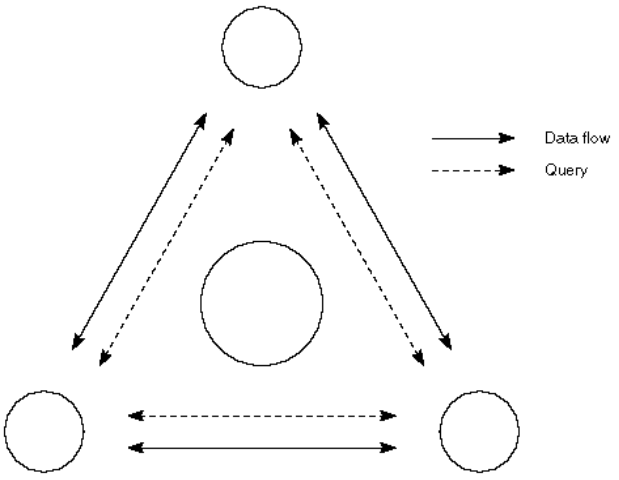



Figure 6 : Automatic Enterprise Knowledge Processing

*3.1.2. Global Updates and Events 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 (see Figure 6). In this case, the local shells monitor the events happening in an application and the database(s) they're 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 and query other databases to update their local databases.

*3.1.3. Adaptability and Flexibility.*

ROPE assures adaptability (1) by fully modularizing the global behavior of local systems into Metadatabase-supported individual shells, (2) by storing its knowledge in the form of rules in the shells, (3) by directly processing these rules in local languages, and (4) by 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 the different 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 in local forms by ROPE (see Figure 7). Only metadata processing is involved in this management of rules.

When a new system is added to the total environment or, for that matter, deleted therefrom, only three things need to be done to update ROPE in a modularized manner: (1) the metadatabase must be updated (through ordinary metadata transactions), (2) a new shell must be created (or deleted), and (3) pertinent rules in other shells must be updated as ordinary rule updates. ROPE, along with MDBMS, facilitates and automates all three things.

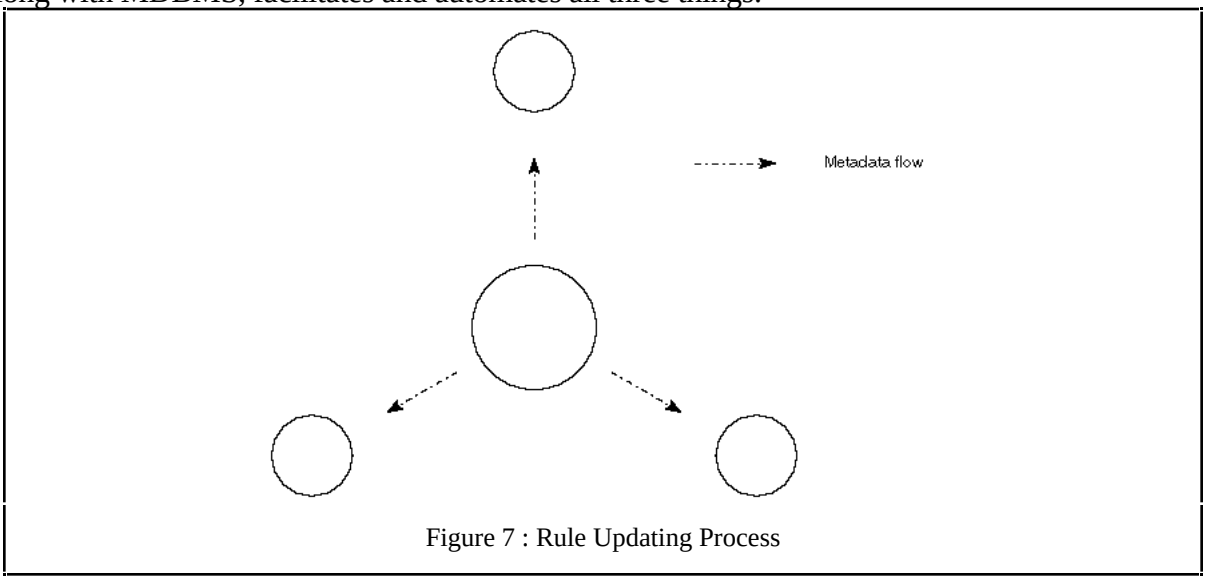



Figure 7 : Rule Updating Process

*3.1.4. Information Model*

The information model of the ROPE method sets the cornerstone for the structure of ROPE's other elements. Consider, based on the above functional description, the creation of shells and the subtantiation of their information contents. The pertinent knowledge in the metadatabase is distributed into these shells, when the global rules are decomposed for distribution. The decomposition of a rule itself is straightforward using the decomposition algorithm developed later in this section. This will generate a set of local queries and subrules to be sent to different shells. However, care must be taken when storing that information into the

local shells to minimize the processing required at each shell. This can be achieved by storing elements pertaining to one rule separately from elements pertaining to multiple rules. Furthermore, each shell must have sufficient information to perform properly its local tasks and to request processing from other systems.

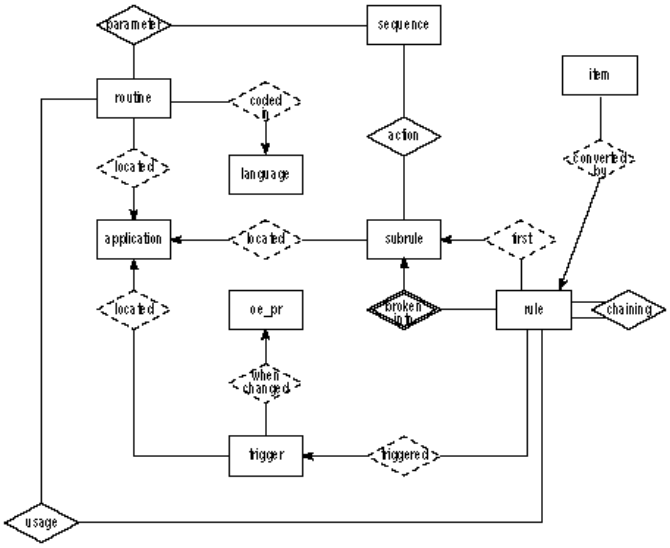



Figure 8 : Information Model of a ROPE Shell

The information model of ROPE can be subdivided in five functional areas: (1) the shell itself, (2) rule information, (3) trigger information, (4) the queries used to monitor the local application, to fetch data used by the rules, and to store the result from the query, and (5) the information regarding rule chaining, i.e., how the rules are interrelated. The resulting structural model can be used to store all the information needed by ROPE to execute the decomposed global rules. This is illustrated in Figure 8, using the TSER.

## 3.2. The Static Structure of ROPE

The static structure of ROPE defines the elements of ROPE that are identical from one shell to another. The local shell structure is shown in Figure 9.

### *3.2.1. 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 flat files, separate from the source code. This allows for easy implementation on different platforms.

### *3.2.2. Network Monitor*

The Network Monitor is the shell's interface with the communications network, thus it provides the window to the outside world for the local application and its shell. It receives incoming messages and passes them to the Message/Rule Processor (see below). It is also responsible for sending messages to other nodes. This module is tailored to the particular network employed for the different applications in implementation, and requires access to the routing information needed to reach the other applications.

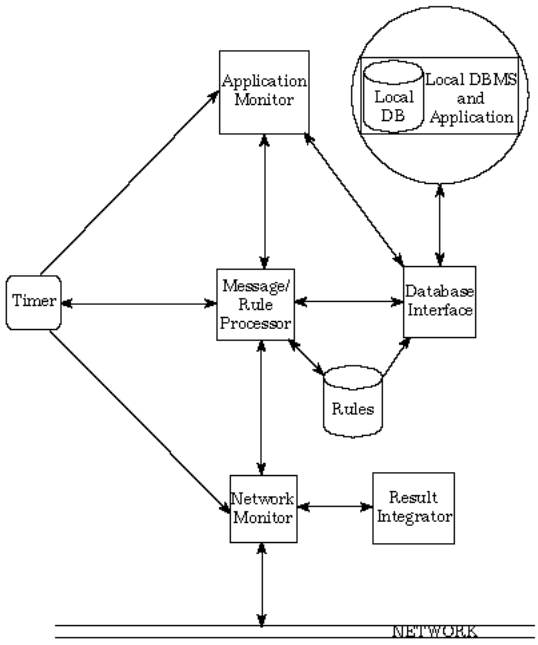



Figure 9 : Local Shell Structure

### *3.2.3. Application Monitor*

This module interfaces with the local application. It "spies" on the behavior of the application and reports any globally significant event to the Message/Rule Processor, such as changes to the local database that would result in the execution of a global behavior rule. The Application Monitor functionality can be specified in general terms, allowing implementations on very different platforms. Furthermore, the use of the Global Query System (GQS) facilities [11] actually removes the need to tailor the Application Monitor to the local database.

Different DBMS and file management systems provide vastly different capabilities for monitoring their states of processing. Some would allow an Application Monitor to be designed by tapping into the local system's own monitoring mechanisms, or even make possible real-time processing. Many, however, do not. Thus, we have developed a basic monitoring algorithm that can be used with any database to detect the unit events of database processing, namely, updates (changes in non-key items), deletions, and insertions. The algorithm creates and maintains a reference table for each of the tables to be monitored in the database, for each distinct rule, and compares it with the current table. Note that the tables to be monitored are determined directly from the rules; that is, only the tables used in the events/rules slated for the shell will be monitored. Thus, the reference tables are not a copy of the entire database, but might be a small fraction of it. The monitoring algorithm compares the current state of the table with the previous state represented

in the reference table to determine the changes that occurred. The algorithm has four basic steps: (1) detection of non-key fields modifications, (2) detection of insertions, (3) detection of deletions, and (4) recreation of the reference table. Table 1 contains the SQL statements corresponding to each step, where `h` is the reference table and `t` the monitored table.

Table 1 : Database Monitoring Algorithm Using a Reference Table

| Step | SQL Statements |
|---|---|
| Updates | **SELECT h.*, t.* FROM h,t WHERE h.key = t.key AND h.nonkey <> t.nonkey ;** |
| Insertions | **SELECT t.* FROM t WHERE NOT EXIST (SELECT h.* FROM h WHERE t.key = h.key) ;** |
| Deletions | **SELECT h.* FROM h WHERE NOT EXIST (SELECT t.* FROM t WHERE h.key = t.key) ;** |
| Recreation of Reference Table | **DELETE FROM h;** **INSERT INTO h (fields) SELECT t.* FROM t ;** |

The biggest challenge with the Application Monitor is how to implement this algorithm while minimizing its dependence to the local database to make it portable. This is achieved by implementing this algorithm in MQL which, in turn, is used to automatically generate the logic of Table 1 and code it in local languages. We only need one (fixed) local query for each shell, which retrieves the whole content of the table to be monitored. Steps detecting changes are accomplished by defining different joins on the current and previous states.

### *3.2.4. Message/Rule Processor*

This module is the inference engine of the shell. It is generic and neutral to particular implementations. On the basis of the events it receives from the Network Monitor, the Application Monitor, the Timer (see below), and the local application, it will trigger the appropriate rule(s) stored in the Rule Segment, or execute the appropriate functions of the shell.

In addition to rule execution, the Message/Rule Processor performs four other functions: (1) it serves as a dispatcher, transmitting local query requests to the Database Interface (see below); (2) it transmits any structural change notification it receives from MDBMS to the Application Monitor; (3) it updates the Rule Segment and notifies the Application Monitor and Timer, when a rule deletion or insertion notification is received; and (4) it updates the frequency at which events are processed.

The Message/Rule Processor consists actually of two processors, the Message Processor and the Rule Processor. The Message Processor deals with dispatching messages and updating the Rule Segment, and the Rule Processor executes the enterprise subrules. Since rules can contain user-defined routines that must be executed by the Message Processor or the Rule Processor, thus the latter must structurally link itself to user-defined routines and dynamically manage the linking to the shell. By separation, the Rule Processor can be linked to any new user-defined routine whenever a rule change occurs, without causing any processing difficulties.

### *3.2.5. Timer*

The Timer manages a list of time events. When a time event is due for processing, the Timer launches the command associated to that time event. This element further mitigates the impact of specificity in local software and hardware environments for the rest of the shell.

The design of the Timer itself is closely related to the local environment. When running in an environment allowing multiple processes (e.g. UNIX), the timer only needs to be an infinite-loop waking up periodically to execute the different events it manages. The single-process environment (e.g. MS-DOS) represents the biggest technical challenge to ROPE implementation. The Timer must be able to run while the regular operations on the hardware can continue. For this to be accomplished, the Timer has to be part of the operating system itself, so it can suspend temporarily regular operations to perform ROPE tasks, or at least, it must behave as if it is part of the operating system.

#### *3.2.6. Result Integrator*

The Result Integrator is used to assemble the results of local queries requested by the local system. In addition, it can perform operations on these queries, like producing a sum of the values on the different rows. The functions this module performs are based on MQL and GQS. The processing of the Result Integrator is straightforward. It receives a set of messages as input, one of which contains a script describing (1) the local queries, (2) the integration operations to be performed, and (3) the output file to be used. Then, these scripts are executed.

#### *3.2.7. Database Interface*

This module converts the data item values to a neutral format used by the shells. This is to remove the burden of processing conversion rules every time we need to use a data value.

### 3.3. 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 systems' 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.

#### *3.3.1. 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 five 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. See Appendix A for the language syntax.

### *3.3.2. 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 [6].

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. See Appendix B for the language syntax.

### *3.3.3. 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. See Appendix C for the language syntax.

## 3.4. Decomposition of Knowledge

The decomposition method is a key for solving the remaining adaptive integration problem: It determines which shell will execute a specific action of the global rulebase model so that the distribution of rules into local shells 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 decomposition algorithm we have developed uses the fact that a rule can be broken down into five stages of execution: (1) rule triggering, (2) data retrieval, (3) condition evaluation and actions execution, (4) result storage, and (5) rule chaining. The idea is to serialize the execution of the rule using these five stages and to assure that the condition and actions of a subrule produced are localized in the same application system. To determine the exact procedures for executing the decomposition, rules are categorized into five types based on their information

requirements: (1) time-triggered rules, (2) data-triggered rules, (3) program-triggered rules, (4) rule-triggered rules — or, simply, chained rules, and (5) user-triggered rules.

*3.4.1. The Basic Logic and Definitions of Decomposition*

The decomposition algorithm must determine where each of the following elements of a rule are to be located: (1) the trigger, (2) a global query template to create the initial rule's fact base, (3) a set of subrules satisfying the criteria mentioned above, and (4) a set of update query templates. When the rule is decomposed, every shell receives a rule insertion message, specifying the information that the shell needs for the particular type of rule. As mentioned earlier, the ROPE information model describes the information contents across the different shells; individual shells, however, do not need to acquire all the possible information about a rule. The decomposition algorithm proceeds to implement the above logic in the prototypical environment as follows:

**I. Generate Message Language constructs corresponding to the trigger definitions contained in the rule.**

**II. Generate a global query template to retrieve the data items used by the rule and update query templates to store the rule's results.**

- **II.a Identify the rule's global queries.**
- **II.b Determine the data items to retrieve and to update.**
- **II.c Generate the queries (can be predetermined).**
- **II.d Generate the query templates (used to update the local databases after the rule execution).**

**III. Generate a set of subrules corresponding to the rule's condition and actions.**

- **III.a Rearrange the condition and actions.**
- **III.b Obtain an optimal partitioning of the rearranged condition and actions.**
- **III.c Generate a subrule for each partition.**

**IV. Generate rule chaining information.**

This algorithm makes use of a set of theorems developed in [2]; the key one is concerned with the decomposition of global rules into localized subrules. The theorem and its supporting lemmas are given here.

Lemma 1: Trigger Localization

- Data triggers and program triggers should be located in particular application systems where their data or program resides.
- Time triggers and chaining triggers can be arbitrarily placed in any application systems.

Lemma 2: Global Query Template Localization

- The global query template for generating the temporary fact base of the rule should be located in the same system as the rule trigger.

Lemma 3: Subrules Localization

- If a rule does not use any user-defined routines nor modify any data items, the decomposition process should produce exactly one subrule (the rule itself) which can be placed in any application system.
- If the rule uses one or more user-defined routine or modifies one or more data items, the decomposition process should produce one subrule for each group in the optimal partitioning of the set of all such operations (user-defined routines and data item modification). Each of these subrules is located in a particular application system where the operation should take place.

Lemma 4: The Pair of First Subrule and Trigger Localization

- The trigger and the first subrule produced by the rule decomposition process should reside in the same application system whenever it is possible — i.e., when either one can be placed in any system.

The Decomposition Theorem: Rule Elements Localization

- Data-triggered and program-triggered rules: The global query template is placed in the same application system as the trigger. If the rule uses no user-defined routines or modifies no data item, the unique subrule produced is also located in the same application system as the trigger.
- Time-triggered and rule-triggered rules: The global query template and the trigger are in the same application system as the rule's first subrule. If the rule uses no user-defined routines or modifies no data item, all elements are located together, in any application systems.

### *3.4.2. The Execution of the Decomposition Logic*

Step I. Extract Trigger Definition

A trigger describes the event that forces the execution of the rule. The rule does not execute the trigger but rather is fired by it when the triggering event occurs. The trigger information is defined in the rule as part of the condition of the original rule, using special trigger operators defined for the Modeling Language. This allows the user to define rules in a standard format, while specifying the information needed to trigger a rule. The Decomposition Algorithm extracts the information from the rule condition, removes trigger functions from the rule text, and generates the appropriate rule insertion statements.

Step II. Generate Query Templates and Encapsulate Query Metadata for Autonomous Processing

One major characteristic of ROPE is the capability to process knowledge using persistent data items stored in heterogeneous distributed databases. For each rule, the ROPE environment creates a temporary fact base, retrieving relevant data items from the appropriate databases when the rule is triggered. Moreover, the shell is able to assign values to the appropriate data items in the local database and store into it the changes produced by the rule execution.

Rules represent stable, predefined knowledge. Hence, the query used by the rule to retrieve (create the temporary fact base) and store (update the global fact base) data items can also be predefined, or at least a template for that query can be predefined. The general approach is to

generate (1) an MQL query to retrieve the needed data items and create the fact base for the rule, and (2) a set of MQL queries to store the changes into the local databases for each rule. Because the query statements used by the rule remain the same from one execution to the other, the Decomposition Algorithm goes one step beyond the simple generation of the MQL queries: it uses GQS to preprocess the query statements, resulting in local query statements and an integration script that can be stored in the shells for repetitive use. This approach reduces the number of messages needed to construct the temporary fact base, while enabling the shells to execute the global query without communicating with MDBMS. Moreover, this approach not only allows the distribution of global query processing, hence minimizing bottlenecks that could occur if only one global query processor was available, but it also confines the use of GQS to the decomposition process. Therefore, by preprocessing the MQL queries, it removes the need to use the metadatabase in the regular operations of the shells (e.g., rule processing).

Step III. Generate a set of subrules corresponding to the rule's condition and actions.

As mentioned earlier, the condition and actions might contain user-defined routines. Furthermore, whenever new values are assigned to data items, there is a need to append an update directive instructing the shell to execute the update query template for the corresponding application system. The execution of these routines and directives of the rule (1) in the appropriate system and (2) at the appropriate time is properly serialized through the Rule Rearrangement Algorithm. To illustrate its logic without showing the full details, consider the rule:

```
if (is_completed(WO_ID) = FALSE) AND
    (due_date(CUST_ORDER_ID) > todays_date())
then
    status := "overdue";
    write_customer_notice(CUST_ORDER_ID);
    update application system Order Processing;
```

The user-defined routine *is_completed*() is located in system Shop Floor Control, while all the other user-defined routines — namely, *due_date*(), *todays_date*(), and *write_customer_notice*() — are located in the Order Processing System. Furthermore, the update directive "update application system Order Processing" is also located in the Order Processing System. Figure 10 illustrates the rule before and after rearrangement.

Once the condition and actions have been rearranged, we try to obtain an optimal partition of the condition and actions of the rule such that the overall logic of the rule is intact and that we have the smallest number of partitions. We preserve the overall rule logic by analyzing the precedence relationships between these condition and actions. In a first step, we look at the precedences between every operation that the rule must perform (an operation in this context is a condition, an action, or any sub-expression composinf a condition or action). The corresponding graph for our example is shown in Figure 11.

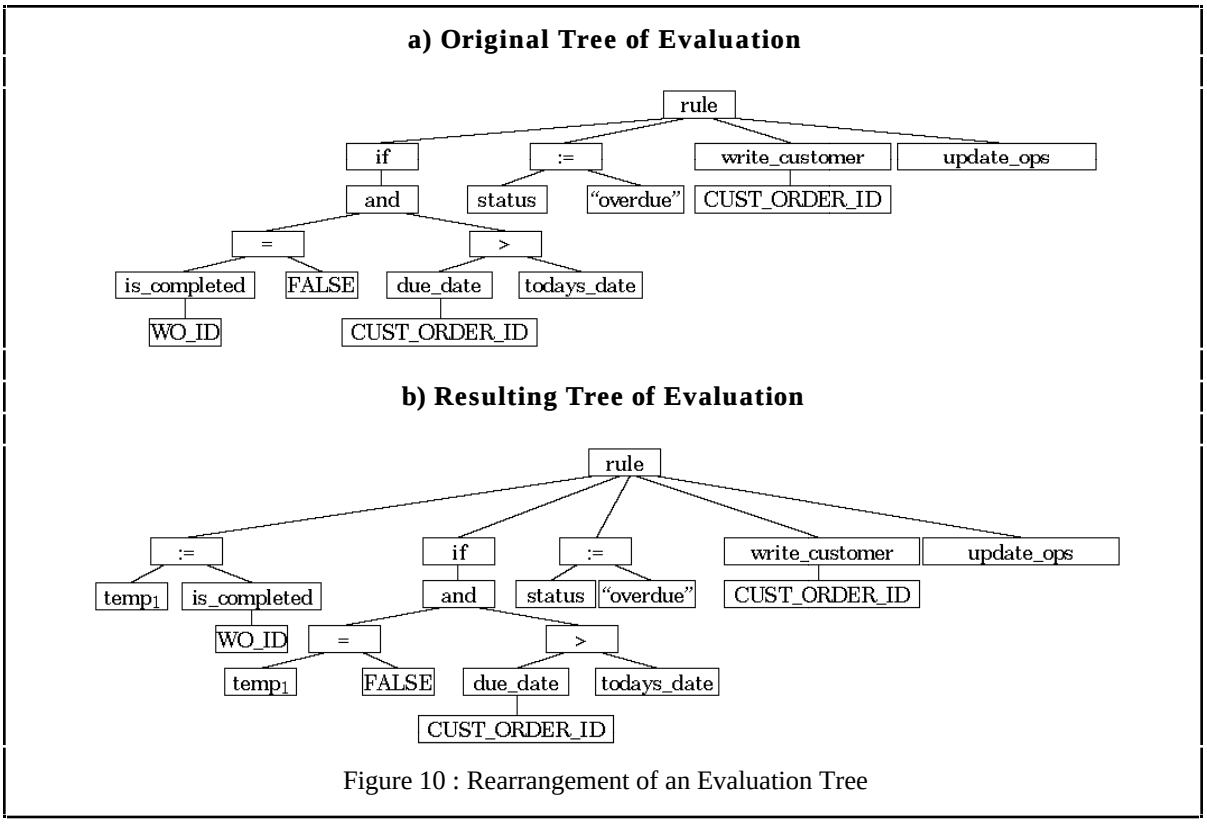



Figure 10 : Rearrangement of an Evaluation Tree

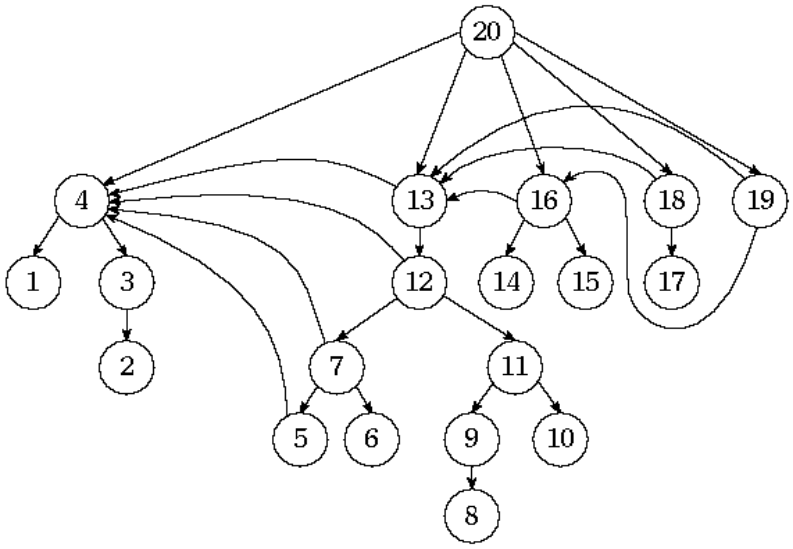
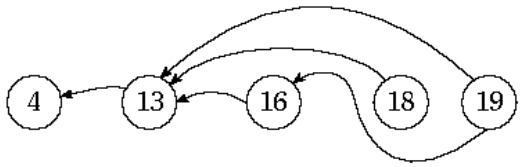



Figure 11 : Precedence Graph Between Operations

Only the nodes corresponding to those condition and actions in the partitioning need to be included to obtain the precedence graph (see Figure 12a). Finally, to simplify the use of the precedence graph, we remove transitive precedence relationships to obtain the strict precedence graph (see Figure 12b). The partitions are created by permuting the order of execution of the condition and actions, while preserving the relationships expressed by the strict precedence graph. Figure 13 shows the optimal partition for our example.

a) Precedence Graph




b) Strict Precedence Graph

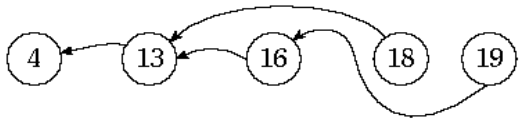



Figure 12 : Precedence Graph Between the Condition and the Actions

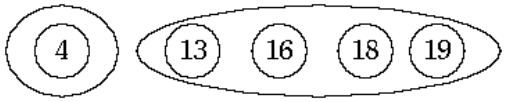



Figure 13 : Partition Optimization

## 4. THE OPERATION OF ROPE

The following scenarios describe how ROPE methods support the integration architecture for multiple systems. We group these scenarios into two classes: (1) rule processing operations (Scenarios 1 through 4) and (2) knowledge management operations (Scenarios 5 through 7). The rule processing operations are mostly concerned with data instances and rule execution; the knowledge management operations deal with the shell's adaptiveness and its ability to change its rulebase. Both will also illustrate how ROPE works.

### 4.1. Rule Processing Operations

In Section 3.4, we have introduced the five stages of rule execution. The first stage, rule-triggering, varies in the details of execution from one type of rules to another (see Section 3.4 for these types). Once a rule has been triggered and its temporary fact base constructed, the remaining stages of the execution are essentially the same for every type. We will now look more closely at the processing details associated with rule triggering. The following scenarios describe how the shells use the knowledge they store to effect the integration of different application systems illustrated in Figure 6.

*4.1.1. Scenario 1: Time-Triggered Rule (for event-based information flows)*

We define time-triggered rules as rules triggered by the occurrence of a time event, i.e., a specific time has been reached. It is the task of the Timer to manage the list of time events that impact on the shell processing. When the Timer wakes up, it retrieves all the time events due for processing and notifies the appropriate module in the shell.

*4.1.2. Scenario 2: Data-Triggered Rule (for event-based information flows and data management)*

Data-triggered rules are executed only when a change is detected in the local application database. The basic events are simply the unit events of database processing: insertions, deletions, and/or updates. Additional events may be defined for particular applications.

We might mention that there exists two approaches to achieve local application monitoring: (1) the event-driven approach and (2) the timer-driven approach. In the first approach, the triggers become part of the system monitored and the monitoring must be embedded either in the application system itself or in an interface that intercepts all pertinent transactions into the system. This is still an area of active research, especially the Active Databases field (e.g., see [10, 38, 33]). For systems that satisfy certain requirements, there are event-driven methods that can be employed for application monitoring in ROPE. The timer-driven approach is more of an external monitoring and hence more universally applicable. At specific time intervals, which are programmable as part of the trigger management by the metadatabase, the application is monitored to see the consequences of change. This approach is less system-dependent and fits the need of event/usage criterion defined in this research just well.

Therefore, to provide a basic method that can maximize the practicality and portability of ROPE, we offer a time-driven mechanism as the default. The shell monitors the local database periodically, using the Timer for launching the monitoring, based on the time event list. The Application Monitor retrieves: (1) the fields in the table to monitor, (2) the name of the reference table to use, and (3) a monitoring query template. Using this information, it creates a monitoring query and launches the Database Interface, where the query is executed by a call to the local DBMS. The result from the query is preprocessed by the Database Interface to convert local values to their global equivalent values. The result is then transmitted to the Application Monitor to compare the previous result (stored in the reference table) and detect changes, using the monitoring algorithm in Table 1. If any significant changes are detected, the Application Monitor launches the Message Processor with the changes. Finally, it creates a new event in the time event list to specify when the Application Monitor is to be launched next.

For simplicity, the present prototype uses only a timer-driven approach to monitor the databases of the different application systems. To implement the event-driven approach, the prototype would require that some tools exist to define triggers in the local application systems or DBMSs. The idea is for those triggers to perform the same kind of process as the Application Monitor. The events-synchronization would be undertaken on an optional (selective) basis using the same rule-oriented control that ROPE performs on other contextual knowledge.

*4.1.3. Scenario 3: Rule-Triggered Rule (for global query and event-based information flows)*

For rule-triggered rules (or chained rules), Stage 1 corresponds to Stage 5 of some other rule: a rule terminates its execution by firing the rules it chains to. Again, in Stage 2, we need to fetch the data needed by the rules to be fired. However, we also need to relate the fact base from the previous rule to the newly retrieved fact base. This is accomplished by (1) using the common

fields on which the chaining occurs and (2) expanding the integration script of the chained rule to join the two fact bases on these field values.

#### *4.1.4. Scenario 4: User-Triggered Rule (for global query, event-based information flows, and data management)*

The ROPE shell allows for a user to fire any rule, from any shell, at any time. There are two ways to fire an arbitrary rule: (1) by triggering a system call to the Message Processor or (2) by sending a message to any application shell requesting the execution of the rule, using the Message Language. The Message Monitor identifies the application shell where the rule execution must start; either where the data or program trigger is located or where the rule's first subrule is located, as stated by the Decomposition Theorem (see Section 3.4.1). The Message Processor dispatches the rule execution message to the appropriate shell and determines if the rule is triggered by data monitoring; in which case the Message Processor forces the monitoring of the entity/relationship associated to the rule.

### 4.2. Knowledge Management Operations

The following scenarios describe how the local shells will behave when changes are propagated from the metadatabase. They represent the different classes of knowledge modification supported by ROPE.

#### *4.2.1. Scenario 5: Structural Model Modification*

Structural model modification includes adding or removing fields or entities/relationships. Once the changes are committed to the local database and the metadatabase, a structural model modification message is sent to the local shell, where it is acquired by the Network Monitor. The message is then transmitted to the Message Processor where it is used to update the global query template of all the rules being triggered by changes in the modified entity/relationship. The next step is to notify the Application Monitor of the changes in the structure and reset the corresponding reference tables by launching the Database Interface to retrieve the current contents of the table after its structure modification.

#### *4.2.2. Scenario 6: Rule Insertion and Deletion*

Rule modification is supported in ROPE as an (old) rule deletion followed by a (new) rule insertion. The System Integrator Manager 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 files of the application shells. The Message Processor performs those changes. The changes could cause any number of the following: (1) updating the list of monitored tables and creating a new reference table for data-triggered rules, (2) inserting a new time event in the time event list for time-triggered and data-triggered rules, (3) updating the list of subrules in the shell, (4) updating the list of user-defined routines located in the shell and rebuilding the Rule Processor and Database Interface (only if adding a new user-defined routine), (5) updating the list of equivalences, and (6) creating query files for the rule. The Message Processor will remove any reference to a rule when receiving a rule deletion message, enforcing the integrity constraints of the ROPE structural model. Any subrule defined for that rule is also deleted. In addition to the constraints implicitly defined in the information model of ROPE, the deletion process includes two more integrity rules: (1) a routine not used by any rule is deleted from the entity "routine" and (2) an item is removed from the entity

"item" if there is no conversion rule associated to the item. Finally, any event referencing a deleted rule is removed from the event list. The Rule Processor and Database Interface are updated when a change occurs in the list of user-defined routines.

#### *4.2.3. Scenario 7: Monitoring Modification*

ROPE will allow the user to modify the time-string associated to a rule or to modify the frequency at which the Network Monitor will perform its tasks. These changes will only take effect the next time the rule is fired.

## 5. IMPLEMENTATION AND ANALYSIS: THE PROTOTYPING OF THE ARCHITECTURE

Rope 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 (since June, 1992). Current sponsorship include Alcoa, Digital, GE, General Motors, and IBM. The AIME Program is jointly funded by the same consortium and the National Science Foundation.

### 5.1. The Prototype

Figure 14 illustrates the laboratory environments at Rensselaer in which the prototype was developed. The metadatabase management system (MDBMS) has been implemented on a micro VAX platform using Rdb as the database engine for the metadatabase and has been demonstrated publicly. A new version using RS/6000, AIX, and Oracle has recently been developed to provide a multi-platform and distributed metadatabase environment [28, 31]. The enterprise environment currently consists of (1) a product design database using ROSE (an object-oriented database management system) and EXPRESS (the PDES/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. A particular test for adaptability and scalability is depicted in Figure 15. The test was concerned with adding the Product Database to the other three systems shown in Figure 14, which were integrated first. The product database was reverse engineered into the metadatabase (proving the metadata independence capability of the GIRD model), which in turn created a new shell for the new system and updated other shells. All major operating requirements — viz., global query processing, global updates and events processing, and adaptability — are both conceptually and empirically tested against this setting.

It is worthwhile to stress that the entire development process incidentally mimics an evolutionary enterprise that this research is aimed to facilitate. While the Order Processing system was developed by using TSER, which is the modeling technology in this case, all other systems had been developed independently before the integration commenced. Among them, the design of Shop Floor Control system could be considered as being influenced by the integration technology, while the Process Planning system was completely a legacy system and the Product Database a new technology — which was incorporated into the scope of integration after the MDBMS had

been developed. Thus, Figure 14 not only depicts a static environment, but also shows a dynamic process of the integration through the prototype.

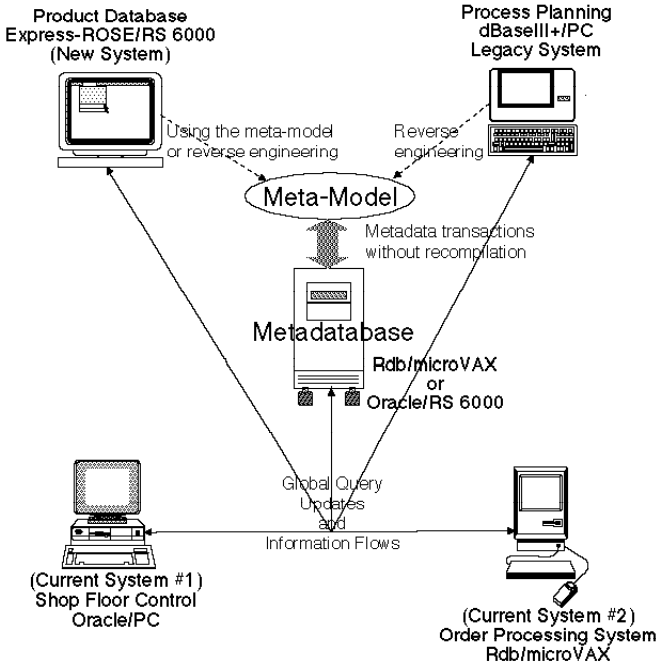



Figure 14 : The Operation of the Active HDDBMS Prototype

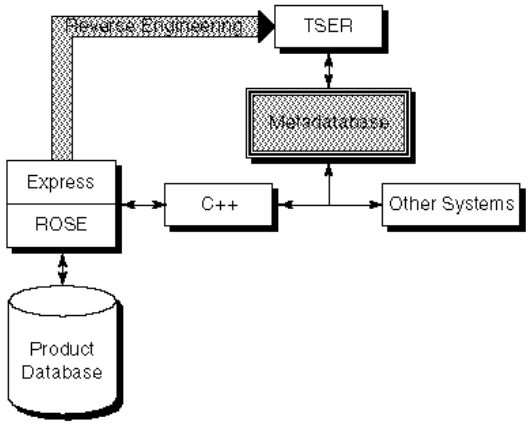



Figure 15 : Models Integration

## 5.2. Performance Assessment of ROPE

The execution model of the architecture hinges on the particular ROPE method, thus, is important to assess the performance of ROPE vis-a-vis concurrent knowledge processing. In order to assess the performance of executing (original) global rules in the distributed shells, the following assumptions are made: (1) it takes *s* steps to execute a ROPE job for a global rule, (2)

each step occurs on a different shell, (3) it takes on average $t$ units of time to execute an operation for a step (i.e., execute a subrule, a local query), (4) it takes $n$ units of time to transmit a message, and (5) messages are processed every $d$ units of time. In the best case, there are no delays, hence it takes $s \cdot t$ units of time to perform the operation. In the worst case, we have to wait $d+n$ units of time between each step, thus, the execution time is $s \cdot t + (s-1) \cdot (n+d)$.

The worst case equation provides some insight on how to optimize the performances of the system. The only theoretical variable in the formula is the number of steps; the more steps we have per rule, the longer it will take to execute the rule. This is why the decomposition algorithm minimizes the number of subrules to reduce the number of times a message is passed to another system. The rest are technological parameters: the performance can be improved by (1) reducing the processing time $t$ with a faster rule processing, (2) increasing the speed of the network, hence reducing $n$, and (3) increasing the frequency at which messages are processed ($1/d$).

## 5.3. Implementation Requirements of ROPE

The information model described in Section 3 is sufficient for ROPE to function as a standalone system. However, if ROPE is functioning as an independent entity without working together with the metadatabase, it will only operate at a degenerate mode where it serves as an enterprise interoperability tool, just like other tools available without providing any evolution management. In our prototype environment, this functionality of evolution is provided by the System Integrator Manager of MDBMS (see Figure 4).

The implementation of the MDBMS requires a relational-compatible database manager, a routine (file) manager, and a decent operating system; all of which are common and are described in [27]. ROPE, on the other hand, interfaces with networks and local application systems, and require more analysis. The information model of ROPE defines the basic information required. Added to this is all the information used to generate the shells and to manage the evolution of the shell: (1) the operational rules, (2) the application resources (hardware and software), and (3) the enterprise data model. These requirements are specified below, which would help an evaluation for possible application of ROPE to a particular interoperability tool, such as CORBA.

Required Information Regarding Rules

- functional model of the enterprise (defining operational rules across applications)
- condition, including the trigger information
- action(s)
- routines used
- data items to retrieve and store
- generation of database queries (retrieval and modification)

Required Information Regarding Application Resources

- description of the different machines (including network addressing)
- description of the software resources (coding and usage)
- location information of the software resources

Required Information Regarding the Enterprise Data Model

- data item description (including key and non-key information)
- equivalent data items information (includes conversion rules)
- integrated structural model of the enterprise (therein defining integrity constraints)
- mapping from the structural model to the actual implementation

In addition, in order for the ROPE shell to function properly, some information must be provided by the system administrator through the Definitional Language; together, the information characterizes these basic capabilities required.

Required Information Regarding the Network

- a command to fetch messages from a remote application system (if necessary), for every pair of application systems.
- a command to send messages to a remote application system, for every pair of application systems.

Required Information Regarding the Local Database

- a command to call the local DBMS (i.e., the DBMS provides a data manipulation language)

Required Information Regarding Environment Specific Commands

- a command to link modules already compiled
- a command to compile a program in C language, producing an object file
- a command to delete files
- a command to copy files
- a command to rename files
- a command to obtain the list of files in a predefined directory

Required Information Regarding Environment Specific Routines

- C language run-time routines to manipulate time and date variables
- a routine to execute operating system calls
- a routine to classify characters as visible and non-visible
- the list of modules used in defining the above functions

## 6. CONCLUSIONS AND REMARKS

In this paper, we presented (1) the concept of model-based architecture for adaptive and scalable multiple systems based on the principle of metadata independence and (2) its execution model, the Rule-Oriented Programming Environment method, for enterprise information

management. We review below the architecture with respect to the four requirements that we formulated in Section 1; namely, scalability, adaptability, parallelism, and autonomy.

There are two levels of analysis to this review: the conceptual model and the execution model. The former is provided through the notion of metadata independence discussed in Section 2, where scalability and parallelism were also explicitly illuminated in Figures 3 and 1, respectively. All four requirements were discussed as to how the proposed model accomplishes these goals. The execution model of the architecture is provided by ROPE, which was presented in Sections 3 and 4 above. How does ROPE satisfy open system requirements? This issue encompasses interoperability and portability, and underlines a technical basis to all four issues reviewed above.

The open system architecture defined by ROPE is composed of three layers: (1) the local environment layer, (2) the shell layer, and (3) the network layer (see Figure 9). The link between the local environment and the shell layer is achieved by (1) the Database Interface in charge of translating any local value into its global equivalent, (2) the Definitional Language that unifies the system calls performed by the shell, 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 (see Section 5), 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 knowledge is acquired from this modeling process: First, each application has its own data model, which is translated into TSER using two modeling primitives (functional dependencies and rules) and a mean to associate, or "package", them. Model discrepancies are solved by defining equivalent data items as contextual knowledge within an application or across applications. This knowledge also includes conversion rules to resolve data format discrepancies between equivalent data items are solved by the definition of conversion rules. Implementation discrepancies in database organization are addressed by mapping the model into the actual implementation. This way, the implementation can differ from the model, but the metadatabase can always be used to trace back and resolve these discrepancies.

The main category of the contextual knowledge is the operating, control, and decision rules pertaining to the interaction of multiple systems. Thus, the event/usage correctness rules are included in the category. 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. In instances where the local application system would

execute a ROPE command, it would do so by sending a message using the Message Language, or by directly calling the Message/Rule Processor.

A level of open system architecture is evident in the above review. In the prototype, as discussed in Section 5, this property is proven in a laboratory setting. On this basis, the four requirements are considered sufficed by the architecture proposed in this paper.

From a generic perspective of concurrent knowledge method, the knowledge decomposition theory developed in ROPE indicates an approach to developing distributed rulebase systems. More interesting to enterprise information manegement is the potential of applying ROPE to enhance CORBA and other interoperability architectures and tools. Based on the analysis of ROPE in Section 5, it seems that a knowledge base could be developed to globally model the key rules embedded in distributed objects, ROPE results would then help, say CORBA, to manage knowledge and close up the feedback loop for architectural evolution. ROPE could also be applied to implement global integrity rules and other contextual knowledge for otherwise disparate databases. A full-fledge metadatabase is not needed in these applications unless adaptiveness or global query capabilities are required.

We also recommend further research into this model, including its deepening and broadening. The former clearly indicates the many details and possibilities mentioned in the text that need further development, such as the incorporation of a serializer to complementing the pure event/usage correctness criterion for transactions that demand instaneous consistency. The latter could lead to new designs for Application Program Interchanger and a method for interoperating multiple repository systems. Finally, the proposed architecture has been extended into real-time physical process control [36], where systems that do not possess mass storage capabilities and must satisfy stringent time constraints are provided an alternate ROPE design based on the execution model herein. Real-time systems are another fertile area for research.

REFERENCES

1. Ahmed, R., P.D. Smedt, W.D. Du, W.K. Kent, M.A. Ketabchi, W.A. Litwin, A.R. Rafii, and M.-C.S. Shan, “The Pegasus Heterogeneous Multidatabase System,” Computer, 24(12), pp. 19-27, December 1991.

2. Babin, G., *Adaptiveness in Information Systems Integration*, Unpublished PhD Thesis - Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, N.Y., August 1993.

3. Barber, K. and M.T. Özsu, “Concurrent Transaction Execution in Multidatabase Systems,” in *COMPSAC 1990*, IEEE Computer Society Press, Los Alamitos, CA, pp. 282-294, October 1990.

4. Belcastro, V., A. Dutkowski, W. Kaminski, M. Kowalewski, C.L. Mallamaci, S. Mezyk, T. Mostardi, F.P. Scrocco, W. Staniszkis, and G. Turco, “An Overview of the Distributed Query System DQS,” in *Proceedings of the International Conferenceon Extending Database Technology*, Springler-Verlag, Berlin, pp. 170-189, 1988.

5. Bell, D.A., J.B. Grimson, and D.H.O Ling, "EDDS — a system to harmonize access to heterogeneous databases on distributed micros and mainframes," *Information and Software Technology*, 29(7), pp. 362-370, September 1987.

6. Bouziane, M., and C. Hsu, "Rulebase Modeling for Data and Knowledge Integration in Multiple System Environments", International Journal of Artificial Intelligence Tools, forthcoming.

7. Bright, M.W., A.R. Hurson, and S.H. Pakzad, "A Taxonomy and Current Issues in Multidatabase Systems," *Computer*, 25(3), pp. 50-60, March 1992.

8. Buchmann, A.P., "Modeling Heterogeneous Systems as an Active Object Space," in *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.

9. Cardenas, A.F., "Heterogeneous Distributed Database Management: The HD-DBMS," *Proceedings of the IEEE*, 75(5), pp. 588-600, May 1987.

10. Chakravarthy, S., B. Blaustein, A.P. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, R. Ladin, M. Livny, D. McCarthy, R. McKee, and A. Rosenthal, *HiPAC: A Research Project in Active, Time-Constrained Database Management*, Technical Report XAIT-89-02, XAIT Reference no. 187, Xerox Advanced Information Technology, Cambridge, MA, July 1989.

11. Cheung, W., *The Model-Assisted Global Query System*, Unpublished PhD Thesis - Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, November 1991.

12. 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).

13. Collet, C., M.N. Huhns, and W.-M.S. Shen, "Resource Integration Using a Large Knowledge Base in Carnot," *Computer*, 24(12), pp. 55-62, December 1991.

14. Dayal, U. and H. Hwang, "View Definition and Generalization for Database Integration in Multibase: A System of Heterogeneous Distributed Databases," *IEEE Transactions on Software Engineering*, SE-10(6), pp. 628-644, 1984.

15. Deen, S.M., R.R. Amin, and M.C. Taylor, "Implementation of a Prototype for PRECI*," *The Computer Journal*, 30(2), pp. 157-162, April 1987.

16. Desai, B.C. and R. Pollock, "MDAS: heterogeneous distributed database management system," *Information and Software Technology*, 34(1), pp. 28-42, January 1992.

17. Dilts, D.M. and W. Wu, "Using Knowledge-Based Technology to Integrate CIM Databases," *IEEE Expert*, 3(2), pp. 237-245, June 1991.

18. Elmagarmid, A.K. and M. Ruzinkiewicz, "Critical Issues in Multidatabase Systems," *Information Sciences*, 57-58, pp. 403-424, September-December 1991.

19. Esculier, C., "The SIRIUS-DELTA Architecture: A Framework for Co-Operating Database Systems," *Computer Networks*, pp. 43-48, February 1984.

20. Eswaran, K.P., J.N. Gray, R.A. Lorie, and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, 19(11), pp. 624-633, November 1976.

21. Georgakopoulos, D., M. Rusinkiewicz, and A. Sheth, "On Serializability of Multidatabase Transactions Through Forced Local Conflicts," in *Seventh International Conference on Data Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 314-323, April 1991.

22. Holtkamp, B., "Preserving Autonomy in a Heterogeneous Multidatabase System," in *COMPSAC '88*, IEEE Computer Society Press, Washington, DC, pp. 259-266, October 1988.

23. Horowitz, J.R. and A.F. Cardenas, "Decomposing Heterogeneous Inter-Entity Relationship Updates," *IEEEE Transactions on Knowledge and Data Engineering*, 4(4), pp. 360-370, August 1992.

24. Hsu, C. and C. Skevington, "Integration of Data and Knowledge in Manufacturing Enterprises: A Conceptual Framework," *Journal of Manufacturing Systems*, 6(4), pp. 277-285, December 1987.

25. _____ and L. Rattner, "Information Modeling for Computerized Manufacturing," *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4), pp. 758-776, July/August 1990.

26. _____ and_____, "Metadatabase Solutions to Enterprise Information Integration Problems," ACM Data Base, pp 23-35, Winter 1993.

27. _____ , 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.

28. _____ , G. Babin, M. Bouziane, W. Cheung, L. Rattner, and L. Yee, "Metadatabase Modeling for Enterprise Information Integration," *Journal of Systems Integration*, 2(1), pp. 5-39, January 1992.

29. _____ and G. Babin, "ROPE: A Rule-Oriented Programming Environment for Adaptive, Integrated Multiple Systems," in *ISMM First International Conference on Information and Knowledge Management*, Baltimore, MD, p. 663, November 1992.

30. _____ and G. Babin, "A Rule-Oriented Concurrent Architecture to Effect Adaptiveness for Integrated Manufacturing Enterprises," in *International Conference on Industrial Engineering and Production Management*, Mons, Belgium, pp. 868-877, June 1993.

31. _____ , 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.

32. _____ , 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.

33. McCarthy, D.R. and U. Dayal, "The Architecture of An Active Data Base Management System," in *Proceedings of the 1989 ACM SIGMOD*, ACM Press, New York, NY, pp. 215-224, 1989.

34. Özsu, M.T. and P. Valduriez, "Distributed Database Systems: Where Are We Now?," *Computer*, 24(8), pp. 68-78, August 1991.

35. Perrizo, W., J. Rajkumar, and P. Ram, "HYDRO: A Heterogeneous Distributed Database System," *ACM SIGMOD 1991*, pp. 32-39, May 1991.

36. Shaefer, O., *Metadatabase Integration in Shop Floor Real-Time Environment*, unpublished Diplom Ingenieur Thesis, Technical University of Munich, Munich, Germany, June 1994.

37. Smith, J.M., P.A. Berstein, U. Dayal, N. Goodman, T. Landers, K.W.T. Lin, and E. Wong, "Multibase — integrating heterogeneous distributed database systems," in *AFIPS Conference Proceedings of the 1981 National Computer Conference*, AFIPS Press, Arlington, VA, pp. 487-499, May 1981.

38. Stonebraker, M., "The Integration of Rule Systems and Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, 4(4), pp. 415-423, October 1992.

39. Su, S.Y.M., H. Lam, M. Khatib, V. Krishnamurthy, A. Kumar, S. Malik, M. Mitchell, and E. Barkmeyer, "The Architecture and Prototype Implementaiton of an Integrated Manufacturing Database Administration System," in *Spring Compcon '86*, 1986.

40. Templeton, M., D. Brill, S.K. Dao, E. Lund, P. Ward, A.L.P. Chen, and R. MacGregor, "Mermaid — A Front-End to Distributed Heterogeneous Databases," *Proceedings of the IEEE*, 75(5), pp. 695-708, May 1987.

41. Vidyasankar, K., “A Non-Two-Phase Locking Protocol for Global Concurrency Control in Distributed Heterogeneous Database Systems,” *IEEE Expert*, 3(2), pp. 256-261, June 1991.

42. Wu, W. and D.M. Dilts, “Integrating Diverse CIM Data Bases: The Role of Natural Language Interface,” *IEEE Transactions on Systems, Man and Cybernetics*, 22(6), pp. 1331-1347, December 1992.

## APPENDIX A: ROPE DEFINITIONAL LANGUAGE SYNTAX

```
<shell definition> ::= <shell specs> <network> <compilers> <functions> <object specs> <user
        calls> ;
<shell specs> ::= 'define' 'shell' 'for' 'application' applcode ',' 'located' 'in' 'directory' directory
        ';' ;
<network> ::= 'system' 'network' 'connections' ':' *[ <network links> ]* ;
<network links> ::= 'for' applcode ':' 'use' -[ +[ command ]+ 'to' 'get' 'messages' 'and' ]- +[
        command ]+ 'to' 'send' 'messages' ';' ;
<compilers> ::= 'language' 'specifications' ':' *[ <lang specs> ]* ;
<lang specs> ::= language 'is' 'invoked' 'by' command ':' *[ 'use' option 'to' [ 'specify' 'output'
        'file' | 'define' 'include' 'directories' ] ]* ';' ;
<functions> ::= 'function' 'specifications' ':' *[ <fct specs> ]* ;
<fct specs> ::= [ <time conversion> | <remove> | <copy> | <rename> | <dbms call> | <file list> ]
        ';' ;
<time conversion> ::= 'converts' [ 'time' 'to' 'integer' 'with' command | 'integer' 'to' 'time'
        'with' command ] ;
<remove> ::= 'removes' 'a' 'file' 'with' command ;
<copy> ::= 'copies' 'a' 'file' 'to' 'another' 'file' 'with' command ;
<rename> ::= 'renames' 'a' 'file' 'with' command ;
<dbms call> ::= 'calls' 'the' 'local' 'DBMS' 'with' command ;
<object specs> ::= 'object' 'modules' 'have' 'extension' extension  ';' 'place' separator 'between'
        'a' 'directory' 'and' 'a' 'filename' ';' ;
<user calls> ::= <return value> <parameters> <linked to> <system call definition> <visible
        characters> ;
<return value> ::= 'the' 'operating' 'system' 'returns' number 'on' 'valid' 'completion' ';' ;
<parameters> ::= 'parameters' 'are' 'separated' 'by' separator ';' ;
<linked to> ::= 'append' link_modules 'when' 'building' 'the' 'modules' ';' ;
<file list> ::= 'creates' 'the' 'list' 'of' 'files' 'with' command '(' number 'header' 'lines', number
        'trailer' 'lines' ')' ',' 'filenames' 'starting' 'at' 'column' number -[ ',' 'extension'
        'starting' 'at' 'column' number ]- ;
<system call definition> ::= 'the' 'function' 'system_call' 'is' 'defined' 'by' ':' '#start_function'
        C_function '#end_function' ';' ;
<visible characters> ::= 'and' 'function' 'is_visible' 'is' 'defined' 'by' ':' '#start_function'
        C_function '#end_function' '.' ;
```

APPENDIX B: RULE MODELING LANGUAGE

```
<rule> ::= -[ 'if' <expression> 'then' ]- +[ <action> ]+ ;
<action> ::= [ 'not' '(' <data item> word <data item>')' | <data item> [ ':=' <expression> | word
        <data item>| '(' -[ <expression>  *[ ',' <expression> ]* ]- ')' ]  ] ';' ;
<expression> ::= <simple expression> -[ [ '=' | '<=' | '>=' | '<>' | '>' | '<' | word ] <simple
        expression> ]- ;
<simple expression> ::= -[ [ '-' | '+' ] ]- <term> *[ [ '-' | '+' | 'or' ] <term> ]* ;
<variable or function> ::= <data item> -[ '(' -[ <expression> *[ ',' <expression> ]* ]- ')' ]- ;
<term> ::= <factor> *[ [ '*' | '/' | 'and' | '%' ] <factor> ]* ;
<factor> ::= string | integer | 'true' | 'false' | real | 'not' <factor> | <variable or function> | '('
        <expression> ')' ;
<data item> ::= -[ <view specifier> '.']- -[ <application> '.']- <item name> -[ <determinant> ]- ;
<view specifier> ::= word ;
<application> ::= word ;
<item name> ::= word ;
<determinant> ::= '<-' '(' +[ <item name> ]+ ')' ;
```

APPENDIX C: ROPE MESSAGE LANGUAGE

```
<message> ::= <subrule execution> | <rule chaining> | <rule firing> | <data query> |
        <structural model modification> | <rule modification> | <monitoring modification> |
        <result integration> | <local query result> | <GQS> ;
<data query> ::= <query> | <store_result> ;
<store_result> ::= 'update' 'database' 'using' file 'with' 'data' 'values' 'for' +[ item]+ ';' '\n'
        <local query result> ;
<subrule execution> ::= 'execute' subrule  [ 'with' 'data' 'values' 'for' +[ item ]+ ';' '\n'
        <local query result> | ';' ] ;
<rule chaining> ::= 'chain' rule 'to' rule [ 'with' 'data' 'values' 'for' +[ item ]+ ';' '\n' <local
        query result> | ';' ] ;
<rule firing> ::= 'fire' rule ';' ;
<query> ::= 'define' 'query' file <local_query> ';'  '\n' 'start_query' query 'end_query' '\n' ;
<local query result> ::= *[ +[ value '|' ]+ '\n' ]* ;
<structural model modification> ::= 'modify' 'oe/pr' oe/pr ':' <key> -[ <nonkey> ]- '\n' <query> ;
<key> ::= 'key' [ 'attribute'  'is' item | 'attributes' 'are' +[ item ]+ ] ';' ;
<nonkey> ::= 'non-key' [ 'attribute'  'is' item | 'attributes' 'are' +[ item ]+ ] ';' ;
<rule_desc> ::= <chained_by> <chain_to> <trig> <decomp> <routine> <convert> <rule_query>
        <rule_store> ';' ;
<rule modification> ::= 'delete' rule | 'insert' rule ':' 'starts' 'with' subrule <rule_desc> ;
<chained_by> ::= -[ 'chained' 'by' +[ rule '(' +[ item item ]+ ')' ]+ ]- ;
<chain_to> ::= -[ 'chains' 'to' +[ rule '(' +[ item item ]+ ')' ]+ ]- ;
<trig> ::= -[ 'triggered' 'by' [ <mon_trig> | 'time' 'using' time-string ] ]- ;
<mon_trig> ::= 'monitoring' <trig_location> -[ <local_trig> ]- ;
<trig_location> ::= 'oe/pr' oe/pr 'in' 'application' appl ;
<local_trig> ::= 'for' +[ [ 'updates' | 'deletions' | 'insertions' ] ]+ ':' <key> -[ <nonkey> ]- '\n'
        <query> 'using' time-string ;
```

```
<decomp> ::= -[ 'decomposed' 'into' +[ subrule ':' cond '(' +[ action ]+ ')' "" -[ subrule ]- "" ]+
        ]- ;
<routine> ::= -[ 'uses' +[ routname ':' type -[ 'with' 'parameters' +[ type ]+ ]- 'coded' 'in'
        language 'located' 'at' path ]+ ]- ;
<convert> ::= -[ 'converts' item 'to' [ 'global' | 'local' ] 'value' ]- ;
<rule_query> ::= -[ 'retrieves' 'data' 'using' ':' 'start_GQS' '\n' <GQS> '\n' 'end_GQS' ]- ;
<rule_store> ::= -[ 'stores' 'data' 'using' ':' 'start_query' '\n' query '\n' 'end_query' ]- ;
<monitoring modification> ::= 'change' 'frequency' 'for' [ 'rule' rule 'to' time-string | 'network'
        'monitoring' 'to' number [ 'minute' | 'minutes' ] ] ';' ;
<result integration> ::= 'resint' file subrule '\n' *[ 'define' [ 'view' view  <resint_q_and_f> |
        'query' file <local_query> ] ';' ]* <resint_q_and_f> ';' ;
<resint_val_fct> ::= 'count' | 'avg' | 'sum' | 'max' | 'exist' ;
<resint_q_and_f> ::= <resint_fct> '(' <resint_query> ')' | <resint_query> ;
<resint_fct> ::= <resint_val_fct> -[ 'group' +[ item ]+ ]- | <resint_disp_fct> ;
<resint_disp_fct> ::= 'distinct' | 'order' +[ -[ [ 'asc' | 'desc' ] ]- item ]+ | 'group' +[ item ]+ ;
<resint_query> ::= +[ [ 'get' | 'use' ] <resint_item> ]+ -[ 'for' <resint_select> ]- | view '(' +[ item
        ]+ ')'
        [ 'union' | 'excluding' ] view '(' +[ item ]+ ')' +[ 'get' <resint_item> ]+ ;
<local_query> ::= 'on' 'application' applcode 'get' *[ item ]* ;
<resint_src> ::= 'view' view | 'query' file ;
<resint_select> ::= <resint_cond> *[ <resint_conjoin> <resint_cond> ]* ;
<resint_conjoin> ::= 'and' | 'or' ;
<resint_cond> ::= <resint_item> <resint_bound> [ <resint_item> | value ] ;
<resint_item> ::= item 'of' <resint_src> ;
<resint_bound> ::= '=' | '<' | '>' | '<=' | '>=' | '<>' | 'EQ' | 'LT' | 'GT' | 'LE' | 'GE' | 'NE' ;
<GQS> ::= 'GQS' file '\n' <mql> ;
<mql> ::= *[ 'define' 'view' view <query_and_functions> ';' ]* <query_and_function> ';' ;
<function> ::= <value_fct> -[ 'group' 'by' +[ item ]+ ]- | <display_fct> ;
<value_fct> ::= 'count' | 'avg' | 'sum' | 'max' | 'exist' ;
<query_and_function> ::= <function> '(' <gqs_query> ')' | <gqs_query> ;
<display_fct> ::= 'distinct' | 'order' 'by' +[ -[ [ 'ascending' | 'descending' ] ]- item ]+ | 'group'
        'by'
        +[ item ]+ ;
<gqs_query> ::= +[ -[ 'from' <source> ]- 'get' [ +[ item ]+ | 'null' ] ]+ -[ 'for' <selection> ]- |
        view [ 'union' | 'excluding' ] view -[ 'on' +[ item ]+ ]- ;
<source> ::= 'view' view | 'application' application | 'subject' subject | 'oe/pr' oe/pr -[ 'in' [
        'application' application | 'subject' subject ] ]- ;
<selection> ::= <condition> *[ <conjoin> <condition> ]* ;
<conjoin> ::= 'and' | 'or' ;
<bound> ::= '=' | '<' | '>' | '<=' | '>=' | '<>' ;
<condition> ::= <item> <bound> [ enclosed_value  | <item> ] ;
<item> ::= item -[ 'of' <source> ]- ;
```