ANL/DIS/CP--86761
CONF-960274--1

# INFORMATION FLOW IN THE DAMA PROJECT BEYOND DATABASE MANAGERS: INFORMATION FLOW MANAGERS

Lucian Russell
Argonne National Laboratory; lrussell@anl.gov

Ouri Wolfson
University of Illinois at Chicago; wolfson@eecs.uic.edu

Clement Yu
University of Illinois at Chicago; yu@eecs.uic.edu

***Abstract***

*To meet the demands of commercial data traffic on the information highway, a new look at managing data is necessary. One projected activity, sharing of point-of-sale information, is being considered in the Demand Activated Manufacturing Project of the American Textile Partnership project. A scenario is examined in which 100,000 retail outlets communicate over a period of days. They provide the latest estimate of demand for sewn products across a chain of 26,000 suppliers through the use of bill-of-materials explosions at four levels of detail. A new paradigm, the information flow manager, is developed to handle this situation, including the case where members of the supply chain fail to communicate and go out of business. Techniques for approximation are introduced to keep estimates of demand as current as possible.**

## 1.0 OVERVIEW

In March 1993 the United States Department of Energy entered into a Cooperative Research and Development Agreement (CRADA) with the American Textile (AMTEX) Partnership. Participating companies range from retailers to producers of cotton and synthetic fiber. One project initiated that year was the Demand Activated Manufacturing Architecture (DAMA) Project. The 1994 Project Plan stated that the DAMA project aimed at "helping the

* The concepts presented in this paper along with others are still under investigation within the DAMA project. Drawing conclusions relative to any DAMA implementation would be misleading at this point in time.

U.S. Textile Industry be more competitive through Information Technology." The term "textile," as defined here, covers the entire range of commerce involving sewn products (Figure 1), and the term "Integrated Textile Complex" (ITC) is used specifically. As the project progressed, the March 1994 version of the five-year plan was more specific about how the U.S. textile industry could be more competitive and introduced the term "electronic marketplace" as part of a vision and objective. The project has continued and has a revised statement; however, the issue of the electronic marketplace initiated a technology exploration of how to manage the data of such a market. The specific question was whether technology for data management was up to the challenge. *This paper is probably the first look at the real commercial data management requirements of the projected information highway.*

In considering the specifics of the electronic marketplace, the most challenging data management problem is that of the "point-of-sale (POS) information backflow." Scanners in retail stores record items sold (at the POS), however, a product count is not helpful in determining product demand, except to the fabricator of the product (e.g., jeans, caps). Each manufacturer in the supply chain needs to know in its own terms (e.g., yards of cloth, thread, number of buttons) how much product was sold, i.e., the met demand. The traceback of the POS data through the bill-of-material (BOM) explosions at each supplier is the POS *information backflow.* It tracks the met market demand of all products from the view of their suppliers. This information flow could be established if all the information was stored and processed centrally, but such a system requires duplicating every company's disk storage for 500 gigabytes of data per year and transaction processing for up to 126,000 users. This paper examines whether a distributed database system could solve this problem. Transaction processing emphasizes transactions that last from seconds to minutes or hours; it seems inappropriate when the accepted time scale for processing is days or weeks (typical of electronic commerce today). Workflows are also considered, but, except for scheduling, they lack a unifying paradigm. The Information Flow Manager (IFM) is halfway between distributed database managers and workflow managers.

## 2.0 PRIOR RESEARCH

### 2.1 RELEVANT LITERATURE ON DISTRIBUTED DATABASE MODELS

Dozens of references on distributed database processing are available. Elmagarind's book describes a number of them [ELMA 91] . However, the most closely related research is the

Mariposa Project [STON 94a,b] and the Dayal et al.'s work [DAYA 90,91] on long-running transactions. More references are not provided here because Stonebraker et al. are forging new ground at the level of of a distributed database with 10,000 nodes. The DAMA project's electronic marketplace could have over100,000 nodes! The Mariposa Project's goals are to unify the approaches taken by four classes of system:

- Distributed database systems,
- Client-server distributed file systems,
- Deep store file systems, and
- Object-oriented database systems.

However, Mariposa focuses on the support of *query* on a collection of data seen as a logical unified entity, but one that is widely physically distributed. In the information flow model, the data are widely physically distributed, but the focus is on *update* to a network of distributed nodes. Unlike the Mariposa Project, the data are private at each node at a level >1, and only a subset is shared between two nodes at levels 1-2. Global queries are not permitted, so the issue of moving data and queries to sites is moot.

Dayal et al.'s work is similar to that of the Mariposa Project, except that it is too restrictive in extending the ACID transaction model. In [DAYA 91], the "model defines precisely the semantics of activities, including compensation and exception handling." Upon failure, we allow activities to be aborted." This model, however, remains one of a single transaction managing, albeit loosely, a group of other transactions. The IFM information contains no such thing. Information flows between level 1 and level 2 are not the start of a chain of nested transactions. Multiple flows may occur before a corresponding level 2 to level 3 flow occurs (e.g., if retail stores update daily, and fabricators update weekly). Thus, many transaction roots are possible Therefore, an aborted information flow does not occur as it does with a transaction.

Also, approximating an information flow works differently than compensating transactions. The goal of the former is to advance, however imperfectly, the state of the database; the goal of the latter is to return it to a prior state. The integrity condition is that *eventually* all data from level 1 nodes will be processed into data used at lower level nodes. The time limit for completion is set by a policy decision of all members of the network and is embodied in the rules of their Association.

## 2.2 RELEVANT LITERATURE ON WORKFLOW MODELS

The other related area of research is in workflows. Three key issues exist in workflows: task specification, task coordination, and execution (correctness) requirements [RUSI 93]. The key difference is that workflow seems to be whatever a particular software vendor or manufacturer chooses to make it, and information flows are not defined by single flows of activities, but rather by multiple flows. Many sets of independent flows of information will yield admissible results flowing between level 1 nodes (retail) and level 5 nodes (fiber producers). Workflows, therefore, are a framework; however, like a house frame, the differences are in the actual "windows and doors" (i.e., integrity conditions in the database) that give the framework a distinctive characteristic. Nevertheless, for completeness, some results are discussed below. Existing literature on workflow models and long transactions concentrates on three issues: recoverability, concurrency control, and programmer productivity.

### 2.2.1 Recoverability

The problem with recoverability is that workflow activities are normally long-lasting and composed of many (traditional) transactions. In some cases, it is necessary to back out transactions after they have committed because the long-term activity of which they are part has aborted. For example, a traveler could rent a car for a trip and then need to cancel the reservation. The work on Sagas [GARC 87] offers a possible solution to this problem. That work postulates that each transaction that is part of a Saga can be compensated for (rather than aborted), even after it has committed, by using a compensating transaction. The system tracks how far the Saga has progressed in its execution, and at the time of Saga abort, the system aborts the running transactions and compensates the committed transactions of the Saga. The user does not have to verify how far the Saga has progressed in its execution and then compensate or abort each transaction manually. In car rental example, the compensating transaction would enter a record of the reservation cancellation.

A Saga could be used in the DAMA project to define the transactions related to the sale of a single garment as a Saga. That is, the Saga would consist of five to six transactions that propagate the demand information down the client-supplier chain. The advantage is that when a garment is returned, the workflow manager (WFM) can automatically compensate by subtracting the demand information from the databases of the effected companies. The disadvantage is that a WFM would need to keep track of which component transactions have been completed in each Saga for millions of Sagas (one for each garment sale).

This overhead seems unreasonable, particularly because a simple solution to the garment-return problem obviates the advantages of Sagas in DAMA. The solution is to complete all the garment-sale chain of transactions (that propagate the demand information) and initiate an independent chain of transactions, unrelated to the sale chain, that either decrements the sales counter or increments the returns counter, depending on the method of handling returns. That is, the DAMA project deals with aggregates, where pairing of a compensation transaction to a sale (i.e., forward) transaction does not seem important. The Sagas' work aims at more stringent compensation requirements. For instance, in the earlier example, it is not possible to compensate for a rental reservation by increasing the inventory; the cancellation must be tied to a specific reservation.

A compensating transaction may be useful if it is important to relate a garment return to a particular purchase. For example, it may be important to know, for the return, when the purchase was made.

### 2.2.2 Concurrency Control

The problem with concurrency control is how to relax the serializability notion to allow long transactions, provide site autonomy, and allow a reasonable amount of concurrency. Concurrency control does not seem to be an important issue in the DAMA project.

### 2.2.3 Programmer Productivity

Workflow managers can specify precedence order and other dependencies and interactions among transactions that make up a workflow (e.g., if transaction T1 aborts, transaction T2 should run). The IFM enforces these dependencies at run time. Without this capability, the dependencies would have to be programmed into the transactions' logic and thus decrease programmer productivity. This capability does not seem important in the DAMA project because the dependencies among transactions are very loose. The main dependency seems to be that when the demand-computation transaction is run at a supplier, the input from its clients for the last period (say a week) must be available (at the Value Added Network). This dependency can be handled via missing data.

## 3.0 THE INFORMATION FLOW MANAGER

This section defines the functionality of an IFM. Such a software system monitors the transmission of information from and to each node in a network of interconnected nodes (companies). The communication is time-critical in that each node has to produce its output, which serves as the input to other nodes, by a certain deadline.

This section concentrates on the way that the IFM handles missing information at a node, when the information is needed to produce output by a certain deadline. The solution involves an architectural framework and a language for detecting missing data, their approximation, and subsequent adjustment of the error introduced by the approximation. The solution is demonstrated by using the DAMA POS information backflow.

## 3.1 OVERVIEW

Applications that involve information flow in a network of nodes are time-sensitive transactions in that (1) the output of one node must be posted as input to another node in the network, (2) the output must be posted by a certain deadline, and (3) the approximate output is tolerable when exact information is unavailable. This information flow is common in many companies that pass a document from department to department. For example, a purchase order may pass from the sales department to the finance department, the credit department, manufacturing, and finally to shipping. Furthermore, each department may have to complete its work on the order and forward it, even though some information in the order is approximate (e.g., the final price could still be under negotiation). Moreover, a customer's credit can be checked, although the final price is known to be *approximately*, for example, $100,000.

This section examines information flow of POS data in the DAMA project. It involves passing consumer-demand information among thousands of companies. A node *N* is obligated to post the demand information for its suppliers by a certain deadline every period (e.g., by Monday at noon, *N* must post demand information for the week ending the previous Friday). The deadline is important because suppliers must plan production, and it is better to provide approximate demand information than to delay posting exact information. Node *N* may be missing input demand information from some of its clients. This information is necessary for *N* to produce its output demand records on time. Missing data also occur, for example, when transaction *T* attempts to read the BOM record for item number *x*, but the BOM record for item *x* is missing.

The solution to the problem of missing data in time-sensitive applications involves three stages: detecting the missing data, approximating the missing data, and subsequently adjusting the error introduced by the approximation. Stage 1 detects that the data returned by the database management system (DBMS), as a result of a database query executed by transaction *T*, are incomplete. Why not perform this detection in transaction *T*? The problem in doing so is that the set of data that *must* be in the database, (i.e., the *core* data) changes over time. It would be desirable to avoid modifying the code of transactions each time; furthermore, thousands of transactions may be involved. In this approach, the user defines a set of keys that constitutes the keys of the core data, and the user changes this set of keys each time the core data change.

Stage 2 involves approximating the missing data by data present in the database. The reason for approximating missing data is that in a time-sensitive application, transaction *T* must complete the missing input and post its output by a certain deadline. Thus, the missing input must be approximated.

Finally, in Stage 3, when the missing data item arrives, possibly after the deadline, an adjustment record must be posted. This record indicates the amount by which the approximate output deviated from the actual output.

This paper proposes an architectural framework and a language by which missing data are detected, approximated, and later adjusted. Also proposed is a rule formalism in which missing data are specified as a condition, and the approximation and adjustment constitute the action executed when the condition is satisfied. Finally, a method of data approximation is proposed for marketing and manufacturing applications.

The problem of detecting missing data may arise in regular databases. For example, a user may want to define in a declarative fashion when the response to a query is considered incomplete and what to do about it. Possible actions that the user can take in response to missing data are approximation and adjustment; they are closely related to the time-sensitive aspect of the application.

The architectural framework consists of an IFM that runs between a DBMS and the application transactions. When a transaction issues a database query, the IFM is given control after the DBMS selects the set of data items that answers the query, but before these items are passed to the transaction. The IFM determines whether the answer set is complete; if not, it approximates the missing data. For example, in the DAMA POS case, transactions read BOM records. Each item has a BOM record that indicates the raw materials and quantities used to produce the item. For example, one unit of item 123 is produced by using 33 units of item 456 and 30 units of item 789.

In case of a missing BOM for item *x*, the IFM substitutes another BOM record, for example, that of item *y*; the transaction then resumes processing. Presumably *y* is an item similar to *x*, and this similarity between *x* and *y* is either predefined for the IFM or computed by the IFM when the BOM for item *x* is eliminated from the database. This mechanism enables *T* to complete the query on time and produce its demand data for suppliers down the chain. The same effect can be achieved by modifying transaction *T* to read BOM-*y* when BOM-*x* is unavailable. However, the proposed approach deals better with legacy systems and avoids the need to modify *T* to substitute BOM-*y* for BOM-*x*.

In addition to approximating the missing data, the IFM may also execute the actions necessary for late adjustment. For example, if the BOM-*x* record is available, although not on-line, the IFM can also invoke transaction *T'* to restore BOM-*x* from the archive. When *T'* completes an *adjustment*, the transaction is run. This transaction will correct the error introduced by the approximation of BOM-*x* by BOM-*y*.

## 3.2 DEMAND INFORMATION FLOW IN THE DAMA APPLICATION

The purpose of the DAMA project is to swiftly provide consumer-demand information to companies of the textile industry to increase their competitiveness and responsiveness to changes in demand. The model discussed here is also relevant to other industries, such as food, automotive and health care.

### 3.2.1 The Communication Structure

The communication structure in the DAMA application is modeled by a layered graph (i.e., a directed graph in which each node has a level and each arc goes from a node at level *i* [the client] to a node at level *i* + 1 [the supplier]. The structure has five levels: (1) retailers, (2) fabricators, (3) textile manufacturers, (4) yarn manufacturers, and (5) fiber manufacturers. Each node *N* at level *i* represents a company that produces items numbered, for example, $I_1,...,I_k$. The company has clients at the level above ($i$ - 1) and suppliers at the level below ($i$ + 1). Each client purchases some of the items produced by *N*, and each supplier supplies raw materials used in producing some $I_j$'s. The arcs connect each node to its clients and suppliers. Figure 2 depicts this communication structure.

The structure has four demand information streams, one for each level of the communication graph. The level *i* demand information stream represents the sales of level *i* to level *i* - 1. For example, the retail demand information stream represents the sales of retailers (level 1) to consumers, and the cut-and-sew demand information stream represents the sales of cut-and-sew manufacturers to retailers. The level *i* demand information stream flows to all levels below *i*, perhaps slipping a level. For example, the fabricator demand information stream flows to textile manufacturers, yarn manufacturers, and fiber manufacturers (Figure 1).

Each arc in the layered communication graph is implemented by a mailbox. The mailbox resides at the value-added-network provider or at a prespecified central site, depending on the system architecture. The client supplies demand information by using a *recoverable* transaction in the mailbox, and the supplier retrieves the information from the mailbox and empties the mailbox. The transaction has to be recoverable because failures that occur at the client or supplier should not lead to the loss of demand data.

### 3.2.2 Format of the Demand Information Data

This subsection specifies the data format in the retail demand information stream. The other streams have a similar format.

For each reporting period, each retailer constructs a set of *store records*, one for each of the retailer's stores. Individual store records are required because retailers (e.g., Marget) may have stores nationwide, and a supplier may request the demand at each location. At the lower levels of the client-supplier chain, each plant can be represented by a store record. Each store record consists of a *header* and a set of *item records*, one for each item number sold during the specified period. The header gives the store identification (ID) number, the period start date, and the period end date. The header also indicates that the item records pertain to the given store or the given period. This paper assumes that the supplier (1) has a relation that stores the location of each store ID and (2) may have another relation that provides temperature and precipitation information for a given location and period. This information may be necessary to explain and forecast the demand. Each item record is an object consisting of the following information:

*item number, price, special-promotion, units-sold, units-returned.*

The item number is unique to a supplier. That is, if two suppliers supply the same logical item, the item will have two different numbers, one for each supplier. (Otherwise, the item and supplier numbers can be linked together to achieve the same effect.)

Special promotion indicates unusual activity, such as extensive advertising. Special promotion can apply to the item or the whole store. If it applies to the whole store, it is turned on for all items sold in the store.

Units sold is a set of pairs (total units sold and discount). Each pair gives the number of units sold for each discount percentage. For example, the two pairs (20,0) and (40,25) indicate that during the period, the store sold 20 units without discount, and 40 units at 25% discount. Thefts can be identified as units "sold" at 100% discount.

Units returned is a set of pairs (total units returned and reason code). Each pair gives the total number of units returned for a particular reason code during the period at the store. For example, code 1 may be *defect,* and code 2 may be *inappropriate size*. Thus, the total number of returns for each reason is accumulated separately.

Each retailer *R* computes a *total item record* for each item by aggregating the item records in all stores. The format of the total item record is the same as that of an item record, except that the price is averaged over all the stores, and the special promotion indicator is a counter of the number of independent special promotions.

**Example 1:** Suppose that retailer *R* in stores *C* and *D* sells item 123. Let [] represent a null or absent value. Retailer *R* computes the total item record for 123 by aggregating the item records for 123 in stores *C* and *D*. Suppose that the item record for 123 in store *C* for December 12-19, 1993, is

123, $10.00, 1, [(10,0),(20,25)],[].

The item record for 123 in store D for the same period is

123, \$10.00, 0, [(11,0),(30,30)],[(2,1)].

Then, the total item record for 123 for the period is

123, \$10.00, 1, [(21,0),(20,25),(30,30)],[(2,1)].

The total item records are used to estimate future demand, to determine the number of units remaining in the inventories of the clients (by subtracting demand from shipments), and to determine the tail end of demand (when the total number of units sold is low). Each total item record is placed in the mailbox of the cut-and-sew supplier that produced it.

Individual store records are archived once they have been aggregated. They must be saved because they indicate more than total demand. They also indicate the location from which the demand originates, and this information can be important when a supplier decides where to locate a new plant. Therefore, node *N* at subsequent layers can initiate a special request for by-location demand information. Retailer *R* and all the nodes on the demand chain between *R* and *N* will then compute the demand for an item produced by *N* at each store. *R* will comply with this request by restoring the individual store records from the archive.

For example, textile manufacturer *M* may request the retail demand for item 456 at store *C* of retailer *R* for December 12-19, 1994. This demand request is passed to the fabricator customers of *M*, who first find the garments that have 456 in their BOM. They then request from *R* the demand information for these garments for *C* for the stated period.

### 3.2.3 Computing the Suppliers' Demand Information

A cut-and-sew node *N* computes the total item record for each item that *N* produces, by aggregating the item record in all its input mailboxes. For each produced item, node *N* may define threshold triggers. The trigger will fire if the number of units sold or returned exceeds or falls below the threshold. For example, a trigger may invoke an analysis program if the number of defect-returns of garment *I* produced by *N* exceeds the threshold. The analysis program provides early warning of production malfunction. The analysis program may need to consider the input item records that were aggregated in order to produce the total item record. The reason is that if a high volume of defect returns is concentrated with a particular client (retailer), the problem probably lies with the way in which the retailer handles item *I*. In this case, evidence of a malfunction in the production of *I* may be insufficient. On the other hand, if the returns are spread throughout the clients (i.e., all clients of *N* had a high number of defect returns for *I*), the problem probably lies with the way that *N* produces *I*. Therefore, this example indicates that the input item records cannot be discarded once they have been aggregated.

In addition to computing the total item records for the items that it produces, *N* also computes the demand information for its suppliers (i.e., the demand information for its raw-material items). These data are item records produced by using the BOM records. *N* has a BOM record for each item that *N* produces and sells to its clients. The output item records are posted in the mailboxes of *N*'s suppliers. Each item record is placed in the mailbox of the supplier that sold the item to *N*.

**Example 2:** Assume that for December 12-19, 1993, the record,

123, 1, [(21,0),(20,25),(30,30)],[(2,1)],

is in one of *N*'s input mailboxes, and the record,

123, 1, [(20,0),(30,30)],[],

in is another. *N* will then produce the following total item record:

123, 2, [(41,0),(20,25),(60,30)],[(2,1)].

Suppose that each unit of item 123 uses five units of item 456 supplied to *N* by textile manufacturer *M*. Suppose further that other items produced by *N* do not use item 456. *N* will then place the following item record in *M*'s mailbox:

456, 2, [(205,0),(100,25),(300,30)],[(10,1)].

The process of producing total item records and demand information for the next level is repeated at the next level recursively. Furthermore, the process is repeated for each demand information stream.

### 3.2.4 Out-of-Business Nodes

Participation of a company (node) in the DAMA project necessitates that the company post its BOM records in an escrow account and that it updates these records periodically. When company *C* goes out of business, a special out-of-business transaction is run. This transaction makes it possible for all suppliers of *C* to read the input mailboxes of *C*. The out-of-business transaction also provides *C*'s BOM records to the suppliers of *C*. Using *C*'s BOM information and its input mailboxes, the suppliers of *C* can compute *C*'s demand output (i.e., their input from *C*), even though *C* is out of the picture.

## 3.3 MISSING DATA STATEMENTS

This section discusses the software mechanisms for detecting missing data (MD), their approximation, and subsequent adjustment. The discussion assumes that the DBMS is relational, but it is easy to see the relational terms (e.g., structured query language [SQL], tuple) can be substituted by analog terms in other data models.

The IFM handles missing data by receiving as input and processing a set of MD statements. Each MD statement is a quadruple:

*(T, sql-stmt1, condition, action).*

The semantics of such a statement are as follows. When *sql-stmt1* is issued by the application transaction *T* to the DBMS, the IFM should "intercept" and examine the resulting set *S* of tuples returned by the DBMS before it is passed to *T* as the answer to the query. The condition is then checked, and, if it is satisfied, action is taken. The condition is satisfied when set *S* is incomplete in some sense. The action attempts to approximate the missing data. The approximate data are added to set *S*, which is returned to transaction *T*. Eventually, this approximation is adjusted when the exact data become available.

In an MD statement, *sql-stmt1* is the label in the code of transaction *T* of an SQL statement that retrieves tuples from a single relation *R* (otherwise, the missing data may become unintuitive). The MD statement *refers* to *R*. The label *sql-stmt1* is passed to the IFM by means of the following technique. The SQL statement is marked by a special symbol in the source code of transaction *T*, and a precompile step ensures that for each statement so marked, at run time, the label of the statement (*sql-stmtl*) passes to the IFM. In other words, at run time, whenever the query sql-stmt1 is issued by transaction *T*, the IFM is informed of the label when intercepting the resulting set of tuples.

The condition and action proposed have special syntax and semantics. Those needs of an application that cannot be expressed in the proposed language require some type of extended transaction.[1] Thus, for example, if the condition can be expressed in the terminology, but the action cannot, the specification of the action can be substituted in an MD statement by an (extended) transaction. The remainder of this section discusses the condition that makes it possible to detect missing data.

The condition of an MD statement is of the form:

*sql-stmt2, comparison-operator, core-set or numeric-constant.*

Generally, the semantics are as follows. Denote the set of tuples returned by the DBMS in response to the query *sql-stmtl* by *S* (*sql-stmt2*). The label of a query is input to the IFM (possibly as part of the condition). Before passing set *S* to transaction *T*, the IFM retrieves from set *S* the relation returned by *sql-stmt2*. Suppose that *sql-stmt2* retrieves set *U*. In that case, the answer to the query labeled *sql-stmtl* is considered incomplete (i.e., missing data occur if set *U* stands in relationship comparison-operator to core-set). If so, the condition is satisfied, and the action part of the MD statement is invoked. Core set is a relation that is input to the IFM. The *sql-stmt2* label can be omitted if the answer set to *sql-stmtl* is considered, as in the comparison.

If either *sql-stmtl* or *sql-stmt2* retrieves an aggregate function (e.g., count) of a set of tuples, the MD rule may specify a numeric constant rather than a core set. The comparison operator is then arithmetic-oriented rather than set-oriented. This option is used, for example, when the user needs to specify that data are incomplete when the number of retrieved tuples is less than 3.

Thus, to deal with a missing BOM, the MD statement

[1] An extended transaction is a set of basic transactions that is partially ordered in time; the basic transactions pass information from one to the other. Commit or abort dependencies may exist among basic transactions in the extended transaction. Many formalisms are available for specifying extended transactions, (e.g., ACTA, Extended Sagas, MIL, contracts, flexible transaction, etc., and any one will suffice. A WFM to run these transactions is assumed to be available [ELMA 91].

*T, read-BOM-x, condition, approximation, adjustment*

is defined, where the condition is (=∅). That is, when *T* reads BOM-*x*, if the answer set is equal to the empty set, the IFM requests that the WFM execute the approximate-and-adjust extended transaction. It returns to *T* a BOM that approximates BOM-*x* and later adjusts the error introduced by this approximation.

*Read-BOM-x* is the label of the SQL statement:

*Select * from BOM where item number = x.*

Suppose that the where clause of *sql-stmt1* specifies the key(s) of one or more tuples. Then, rather than a core set in the condition part of an MD statement, the keyword *where-clause* can be used. It indicates that the core set, rather than being predefined as input to the IFM, is defined by the *where clause* of *sql-stmt1*. Specifically, core set consists of all the tuples specified in the *where clause* of *sql-stmt1*. For example, consider the MD statement

*T, read-BOM-xz, condition, approximation, adjustment,*

where the condition is (=,*where*-clause).

*Read-BOM-xz* is the label of the SQL statement:

*Select * from BOM where item number = x or z.*

This SQL query reads the BOM of items *x* and *z*, and the MD condition indicates thatit can be satisfied if either the BOM of *x* or the BOM for *z* is not in the database.

The same mechanism can deal with missing data (i.e., data that should have been transmitted from clients at a particular time, but are missing when expected). For example, suppose that all store records are read into a relation with the following attributes:

*store-id, item-number, period, units sold, units returned.*

Suppose further that a transaction $Q$ reads all the item record of item $x$ by using the following SQL statement, labeled Read-$x$ for convenience:

*Select store-id, units sold, units returned*
*from DEMAND*
*where item-number = x, period = 12/10/93 - 12/17/93.*

This study also assumes that a relation in STORES-$x$ exists that contains the set of the store-ids that are supposed to post demand information for item $x$ at the end of each period. The MD statement that detects if a store filed to post demand information at the end of the period is *(Q, Read-x, condition, action)*, where the condition is

*project-store-id, not-contains, STORES-x.*

*Project-store-id* is the label of an SQL statement defined together with the MD statement; it projects out of the set of retrieved by *Read-x* all attributes, except the *store-id.* Thus, if the set of *store-ids* retrieved by *Read-x* does not contain the set of *store-ids* in *STORES-x*, *action1* is invoked. *Action1* consists of an approximation and an adjustment. For example, the demand for the week ending December 17, 1994, is approximated to be the demand of the previous week. Adjustment is invoked when the actual demand record is posted.

When the condition is satisfied, the missing objects are identified and made available to any transaction $T$ specified in the action part of an MD statement. The set of missing objects is identified by the set of keys in the core set of the condition, but not in the set of tuples retrieved by *sql-stmt2* (or *sql-stmt1* if *sql-stmt2* is missing). For example, consider the latest case discussed, where *sql-stmt1* retrieves the BOMs for items $x$ and $z$. If the statement retrieves only the BOM of $x$, $z$ will be passed as a parameter to the action part of the MD rule. In addition, the invocation time $t$ and the external parameters passed to $T$ are also made available to any transaction specified in the action part of an MD statement. Sections 3.4 and 3.5 discuss the case where the action is the pair (approximation, adjustment).

## 3.4 APPROXIMATING MISSING DATA

This section describes the approximation component in the *action = (approximation, adjustment)* part of the MD statement. First, the approach to approximation and language constructs is outlined to support the approach. The approximation returns to the application transaction additional data items that are not returned to the DBMS. The defined language constructs inform the IFM to compose these items from the database.

The approach to approximation works for numeric data. It is based on regarding each tuple as a vector, which, if missing, is approximated by a linear combination of other vectors (tuples). If there is insufficient information for the IFM to perform the approximation for a missing data item, that data item will not be approximated. The application transaction *T* will have to manage this situation.

The language constructs also enable one to specify, as part of the approximation step, a solicitation-transaction *U*. If specified, *U* will be executed after the response *sql-stmt1* is returned to the application transaction *T*. Transaction *U* solicits exact data. For example, in the case of a missing BOM, *U* would initiate the staging of the BOM from the archive, or it would request the BOM from the central authority referred to here as "DAMA Inc." In the case of missing demand data, *U* sends the message that informs the client that the demand data are delinquent. Syntactically, the approximation is specified by a triple (*LCR, P, U*), where *LCR* is a relation that indicates how to perform the approximation, *P* is a set of parameters, and *U* is the above-mentioned solicitation-transaction.

The approach to approximation may be inappropriate for some applications. For example, symbolic data may need to be approximated. In this case, the MD statement may specify a separate transaction to perform this approximation, or it may bypass the approximation stage and let application *T* handle the missing data. Alternatively, a surrogate to the data may be generated to be replaced by transaction *U*, which obtains the correct data from "DAMA Inc." at a later date.

The approach in Section 3.4.1 assumes the use of the relational model, but could easily be extended to the object-oriented model.

### 3.4.1 Two Types of Approximation

Two types of approximation are considered: time series and similarity. In *time-series approximation*, a missing tuple is approximated by a linear combination of previous versions of the same tuple. For example, *X* represents the sales of an item for a particular week and is approximated as the sum $0.8 * Y + 0.2 * Z$, where *Y* and *Z* are the sales of the *same item* one and two weeks ago, respectively.

In *similarity approximation*, a missing tuple is approximated by a linear combination of other tuples in the same relation. For example, *X*, the BOM for an item, is approximated as the sum $0.8 * Y + 0.2 * Z$, where *Y* and Z are the BOMs of *two other items.*

This paper assumes that for each MD statement, the approximation is either similarity based or time-series based, but not both. That is, a database administrator cannot specify that for a missing tuple with key *X*, the approximation is similarity based, and for another tuple with key *Y*, the approximation is time-series based. Elimination of this restriction is not difficult, but it clutters the presentation.

The approximation component of the action is a pair (relation, parameters). Each member of the pair is described in Sections 3.4.2 and 3.4.3.

### 3.4.2 Relation Used by the IFM for the Approximation

Each MD statement has a *linear combination* relation called *LCR*. The *LCR* is a set of linear combination tuples. Each linear combination *(lc)* tuple provides for a key that is potentially missing (i.e., the vectors' keys and the coefficients that should be used in the approximation linear combination).

Specifically, consider similarity approximation for an MD statement that refers to relation *R*. Assume that the key of *R* is the attribute *K*, and each missing tuple in *R* is approximated by a linear combination that involves two other tuples in *R*. The attributes of *LCR* are then *K, COEFF1, K1, COEFF2, and K2*. Suppose that if the tuple with key *X* is missing in *R*, it should be approximated as the tuple obtained by the linear combination $0.8 * Y + 0.2 * Z$. Inserting the *lc* tuple *t* into the *LCR* relation results in $K = X$, $COEFF1 = 0.8$, $K1 = Y$, $COEFF2 = 0.2$, *and* $K2 = Z$.

The semantics are as follows. Suppose that the tuple with key *X* is missing in *R* (the IFM detects this when evaluating the condition part of the MD statement). The IFM then approximates the tuple by performing the following functions. It first reads from the relation *LCR* the tuple with $K = X$. This tuple is *t*, indicating that *x* is approximated by a linear combination of *Y* and Z. The IFM then reads from *R* the tuples with keys *Y* and Z and computes the tuple $q = 0.8 * Y + 0.2 * Z$, which is returned to the application transaction.

Any combination of the parameters type, status, and insertion mode is legitimate. For example, status = dynamic and insertion mode = automatic means that when an *R* tuple with key is deleted, it triggers the initiation of a method that inserts a tuple with key in the LCR. When status = dynamic and insertion mode = manual, deletion of the *R* tuple with key *lc* triggers a message to the database administrator (which in turn inserts the tuple). When status = fixed and insertion mode = automatic, the fixed relation is computed by a program. The program has to be provided with the keys of the *R* tuples that have to be approximated (i.e., have an *lc* tuple in the LCR.

The fourth parameter, *recursive* = *Y* or N, pertains to the approximation procedure rather than the LCR relation. It indicates whether the IFM is to perform the approximation recursively. For example, suppose that the LCR indicates that the *R* tuple with key is to be approximated by the tuple with key *Y*. Furthermore, suppose that at the time of approximation, the tuple with key *Y* is not in the relation *R*. This problem has two possible solutions. The first solution is to abort the approximation and indicate to application transaction *T* that the *R* tuple with key is missing. The second solution is to approximate the *Y* tuple (assuming that there is an *lc* tuple with key *Y* in the LCR) and then to use this approximation to approximate the *R* tuple with key . If a *Z* tuple is used to approximate the *Y* tuple, the *Z* tuple may also be missing and may need to be approximated. Thus, this is approximation recursive. Rather than recursive = *Y*, the database administrator may specify, for example, recursive = 3, indicating that recursive approximation should be attempted up to a depth of three. If the approximation does not succeed at that level, application transaction *T* is notified that the *R* tuple with key is missing.

### 3.4.4 Automatically Computing a Linear Combination Tuple

This section discusses a particular method of automatically computing the linear combination that approximates a numeric tuple. The method is a heuristic that computes a simple approximation, consisting of one key and one coefficient. The heuristic is discussed in the context of BOMs, but it applies to any application in which the objective is to approximate a set of numeric values of a tuple.

In a simple approximation, item $x$ is approximated by item $y$; presumably the BOM of item $x$ is similar to that of item $y$. This study proposes a vector-based approximation that numbers all the raw materials by 1 through $K$. The BOM of item $I$ is represented by a vector $I = (i_1, i_2, \ldots, i_k)$, where $i_j$ is the number of units of material $j$ in the production of item I. If $i_j = 0$, then the $j$'th material is not used. A vector is normalized if its length is 1. A vector $A = (a_1, a_2,..., a_k)$ is normalized when A is multiplied by

$$|A| = \Sigma_{i = 1,...,k}\, a_i^2.$$

The cosine function is adopted as a measure of the similarity between two normalized vectors [SALT 89]; the higher the value of the cosine is, the higher the similarity is. The cosine between the two normalized vectors is given by the dot product of the vectors. Specifically, if $A = (a_1, a_2,..., a_k)$ and $B = (b_1, b_2,..., b_k)$ are two normalized vectors,

$$\cos(A, B) = \Sigma_{i = 1,...,k}\, a_i. * b_i.$$

The method proposed here approximates the BOM of item $x$ by item $y$, such that the cosine between the two normalized vectors is maximum. The approximation is automatically computed as follows. First, the vector representing the BOM of each item $z$ in the BOM relation is normalized to obtain the vector $zn$. Suppose that

$$zn = c_z\, z.$$

The method then finds the BOM of item $y$, such that the cosine between $xn$ and $yn$ is maximum among all the possible vectors $y$ that represent the tuples in the BOM relation. The BOM for item $x$ is then approximated by the BOM for item $y$. The coefficient of the approximation is the constant $c = c_y / C_c$. Thus, the tuple $K = x$, $coeff1 = c$, $k1 = y$ is inserted in the LCR relation of the MD statement.

## 3.5 ADJUSTMENT OF MISSING DATA

This section describes the adjustment component in the *action = (approximation, adjustment)* part of the MD statement. Consider an MD statement $m$ = *(T, sql-stmt1, condition, action)* that refers to the relation $R$. The adjustment step of $m$ is optional and, if specified, it is executed when the first of the following two events occurs. The first event occurs when a tuple $t$, identified as missing in the condition part of $m$, is inserted in the relation $R$. In other words, the insertion of $t$ triggers the adjustment step. The second event occurs when a time limit specified as part of the adjustment expires.

On the other hand, if $T$ is not idempotent, a separate transaction has to be written to perform the adjustment. Transaction $Q$ is invoked with the parameters of $T$ at the original invocation.[2]

## 3.6 ALGORITHM FOR PROCESSING MD STATEMENTS

When missing data are detected, save invocation parameters and the ID of the missing tuple(s) in a working relation. The working relation schema is as follows:

*md-label, transaction start time, invocation parameters, missing tuples keys,*

Then, set up an adjustment trigger that invokes the IFM when a missing tuple is inserted in the database and one for expiration of a deadline. Whichever trigger occurs first cancels the others. The key of the missing tuples is inserted in the IFM work relation. When the IFM is invoked as a result of a tuple insertion, it first identifies which entry in the working relation the invocation related to; it then invokes the transaction in the MD nonidempotent of idem. If the IFM is invoked as a result of expiration time, it invokes the approximation transaction.

## 3.7 APPROXIMATE DATA MANAGEMENT

Previous sections have discussed the functionality of the IFM. It can be implemented without substantial changes to the DBMS. However, the DAMA application can be better supported if the DBMS is enhanced to manage approximate data. Without such an enhancement, a node does not know whether it is demand data or approximate until a revision arrives.

This study proposes to enhance the DBMS functionality. Each field of each database object can be specified to be approximate (for numeric fields) or uncertain (for symbolic fields). A transaction may examine the "exactness" of each field it reads from the database. Suppose that transaction $T$ has read from the database an approximate field. By default, the DBMS marks as approximate every field of every item output by $T$. Therefore, a field is marked approximate if either it is approximated by the IFM or it is output by a transaction that read an approximate field.

2 Exact invocation may not produce the same results as if the exact input were in the database at the time of the approximation because other variables in the database may have changed. The output is the same as if the exact input has just arrived, and the approximation simply served as a placeholder.

This functionality can be extended to include the time at which the exact value is expected, the degree of certainty of a field, and a language by which to specify how approximate input is to be mapped to an approximate output (rather than if the shotgun approach in an approximate indication is mapped by the DBMS to every output field).

## 3.9 ANALYSIS OF DAMA PARTNERS COMMENTS

The IFM material was made available for review by some members of the DAMA project, and two issues were raised. The first addresses the usefulness of the IFM's approximation concept, and the second addresses the extent of IFM deployment.

### 3.9.1 Incorrect Approximations

The approximation concept introduced herein was challenged as a step that might produce incorrect results with financial ramifications. For example, a pair of jeans with buttons rather than zippers results in a very different set of orders to suppliers of those items. An approximate transaction, if interpreted as a nonexistent continuing demand for buttons, could result in an inventory of buttons with no use and thus adverse financial results. Such an effect of using the IFM would cause it to be abandoned.

The answer to this objection is that the IFM actually represents how business is performed today; the only flaw is that the IFM concept makes this problem visible. What is really being challenged is that a company at level $i + 1$ is being fed approximate data from a company at level $i$, and the accuracy of these data is uncertain. Of concern to DAMA and the IFM's algorithms, however, is the flow of information. Thus, an alternative formulation is proposed that could be more acceptable in the DAMA setting. A company at level $i + 1$ could be informed that no demand was being transmitted from a company at level $i$, and then the company could use its internal sales analysis and forecasting system to generate a set of approximations. That is what happens today. These approximations could then be used to approximate demand figures transmitted to companies at level $i + 2$. To the internal databases, the transaction steam would not look any different.