

Information flow in the DAMA project beyond database managers: information flow managers

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1996 Distrib. Syst. Engng. 3 263

(<http://iopscience.iop.org/0967-1846/3/4/006>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.211

The article was downloaded on 20/02/2012 at 20:59

Please note that [terms and conditions apply](#).

Information flow in the DAMA project beyond database managers: information flow managers

Lucian Russell†§, Ouri Wolfson‡|| and Clement Yu‡¶

† Argonne National Laboratory, Building 900, 9700 South Cass Avenue, Argonne, IL 60439, USA

‡ EECS Department (M/C IS4), University of Illinois at Chicago, 851 S Morgan Street, Chicago, IL 60607-7053, USA

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 (DAMA) of the American Textile Partnership (AMTEX) 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. Enabling this communication requires an approach that shares common features with both workflows and database management. 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 so as to keep estimates of demand as current as possible.

1. Overview

In March 1993 the United States Department of Energy entered into a Cooperative Research and Development Partnership (CRADA) with the American Textile (AMTEX) Partnership, participating companies ranging from the retailers to the producers of cotton and synthetic fibre. One project initiated during that year was the 'Demand Activated Manufacturing Architecture' (DAMA) project. The 1994 project plan stated that the DAMA project was aimed at 'helping the US 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 this was to be accomplished and introduced the term 'electronic marketplace' as part of vision and objective statements. The project has continued on and has a revised statement; however, the issue of the electronic marketplace initiated a technology exploration of exactly how the data of such a market could be managed. The specific question was whether technology for data management was up to the challenge. This 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, feet of 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 were stored and processed centrally, but such a system would require duplicating every company's disk storage for 500 gigabytes of data/year and transaction processing for up to 126 000 users.

Could the processing be accomplished through a distributed database management system (DBMS)? Such systems are built upon the technology of transaction processing (for a discussion of transactions see [4,5]). DBMSs, however, are engineered and finely tuned based on the assumption that transactions last from seconds to minutes, or at most hours. Such systems are inappropriate when time scales for processing are measured in days or weeks as is typical of electronic commerce today.

§ E-mail: lrussell@anl.gov

|| E-mail: wolfson@eeecs.uic.edu

¶ E-mail: yu@eeecs.uic.edu

Workflows were also considered, as electronic commerce's focus is upon the processing of business documents that make up the most important operational work flow within an enterprise. In this context, however, the flexibility and dynamism of workflows work to its disadvantage as they have no unifying paradigm except scheduling and autonomy of processing. An approach in between workflows and DBMSs was needed; it is the information flow manager (IFM).

2. Prior research

Information flow managers are halfway between distributed database managers and workflow managers. Both literatures are touched on as needed.

2.1. Relevant literature on distributed database models

Dozens of references on distributed database processing are available. Elmagarmid's book [8] describes a number of them. However, the most closely related research is that of the MARIPOSA project [14, 15] and the work of Dayal *et al* [6, 7] on long running transactions. More references are not provided because the DAMA project's 100 000 nodes is a new level of scalability in distributed processing: in the introduction to the MARIPOSA project Stonebraker *et al* state they are forging new ground at the level of a distributed database with 10 000 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 to *query processing* on a collection of data seen as a logical unified entity, but one that is widely physically distributed. In the DAMA project the data are also widely physically distributed, but the focus is on *update* to a network of distributed databases. Unlike the MARIPOSA project, the data are private at each node at each node (for levels > 1), and only a subset is shared among any two nodes at higher levels. Global queries are not permitted, so the issue of moving data and queries to different nodes, a technique in MARIPOSA, 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 (atomicity, consistency, isolation and durability). In [7], 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 master transaction managing, albeit loosely, a group of other transactions. The IFM contains no such master transaction. Information flows between level 1 (retail) and level 2 (fabrication) are not the start of a chain of nested transactions. Multiple flows may occur before a corresponding level 2 to level 3 (textiles) flow (e.g. if

retail stores update daily, and fabricators update weekly). Thus, many transaction 'roots' are possible. Therefore an 'aborted' information flow exists as it would if data were communicated by distributed transactions.

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 first level 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 embodied in the rules of their association.

2.2. Relevant literature on workflow models

The other related area of research is in workflows. Workflows are characterized as being long-duration multi-step activities with data and control flow between their constituent steps; these steps are activities within a processing station, and the station is taken to be considered as a black box. A survey of the recent literature on workflows appears in [10]. There is no restriction upon the duration, the number or type of steps and/or control, whether it is static or dynamic, or whether the number of steps is fixed or a dynamically determined number.

In a recent paper [3] the environment of workflows was characterized as one with activities exhibiting autonomy, dynamic control flow, partial automation and partial connectivity. The authors then proposed a model that addressed the issues raised in [12], namely that there are three key issues in workflows: task specification, task coordination and execution (correctness) requirements. Specifically, they developed a computation model for workflows based on the notion of an information carrier (INCA). These are packages of information describing a service to be performed at a processing station within the workflow, together with a context in which the service is executed.

The IFM model was developed because whereas workflows capture the notion of work being passed through a chain of processing stations, information flows are not defined by a single flow of activities but rather multiple flows among thousands of processing stations, which in turn execute a common type of activity. The information flow is coordinated, governed by standard agreements reached among all the enterprises that will be linked. Workflows therefore are a framework in which IFMs can be considered, much as the framing of a house, to borrow the analogy of architecture propounded by Zachman [16]. The IFM, however, has specific details that distinguish it, driven by the fact that when reaching an enterprise the information flows directly feed into its internal database. In terms of the architecture analogy, the next level of detail has been specified: the information processing flows all look like transactions between databases.

Existing literature on workflow models, as recapitulated in [3] shares a legacy with that of long transactions. Thus a more detailed look is taken at 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 long-term activity of which they are part has aborted. For example, a traveller could rent a car for a trip and then need to cancel the reservation. The work on Sagas [9] offers a possible solution to this problem. That work postulates that each transaction is part of a Saga whose impact on the database can be adjusted out (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 the rental car example, the compensating transaction would enter a reservation cancellation.

A Saga could be used in the DAMA project to define the transactions related to the sale of a single garment. That is, the Saga would consist of five to six transactions that propagate the demand information down the customer-supplier chain; its execution is assumed to be controlled by a workflow manager (WFM). The advantage is that when a garment is returned, the WFM can automatically compensate by subtracting the demand information from the databases of the affected 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 there is a simple solution to the garment-return problem, which 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) of a single 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. In practice, however, this information is restricted to the retail level, and is made known to other levels only qualitatively.

2.2.2. Concurrency control. Concurrency control *per se* is not an important DAMA project issue, but the way in which the IFM handles missing data touches on related issues, notably serializability. The principle of serializability is that transactions processed by a system

should be handled in such a manner as to appear to have been processed sequentially in some order. The issues of concurrency control are in essence how to relax the notion of serializability. In the current case the relaxation would be to permit long transactions, provide site autonomy, and allow a reasonable amount of concurrency. Concurrency controls guarantee this. Although the notion of serializability, discussed in detail in [11], is simple to state in principle, it is difficult to describe precisely—in fact there are *final state*, *strict*, *view*, *conflict* and *multi-version* serializability concepts.

Moreover, the choice of serializability concept has an impact upon other data management issues such as rollback, recovery, and checkpointing. The result is that the overhead of keeping track of multiple transactions produces a lack of predictability for the completion time of any particular transaction and a performance penalty for the set of transactions as a whole. Because of this alternatives have been sought. In [8] a number of different transaction models are discussed. Some of these are characterized by having a pre-specified amount of error (versus the serialized case) that they are willing to tolerate. In the DAMA project, serializability is not an issue. All transactions are generated and subsequently processed in a batch using the ANSI formats for electronic data interchange. Also, only one company's data need be processed at a time.

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 T_1 aborts, transaction T_2 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 customers for the last period (say a week) must be available (e.g. at the value added network that transmitted the data). This dependency can be handled via managing missing data.

3. The information flow manager

This section defines the functionality of the IFM. Such a software system monitors the transmission of information to and from each node in a network of interconnected nodes (enterprises). 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. The nodes, as in workflows, process autonomously at some level, but are supposed to receive and or transmit information to other nodes according to an agreement within the entire ITC. Such an agreement includes the necessity of retaining information relevant to the POS information backflow for a definite though limited amount of time. For practical purposes a window of about two weeks is adequate today.

This section specifically 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 point-of-sale (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 the 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 activities: detecting the missing data, approximating the missing data, and subsequently adjusting the error introduced by the approximation. Activity 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. Rather, 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.

Activity 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 so as to provide suppliers a more accurate estimate.

Finally, in activity 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 then 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 the 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 the enterprise's 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

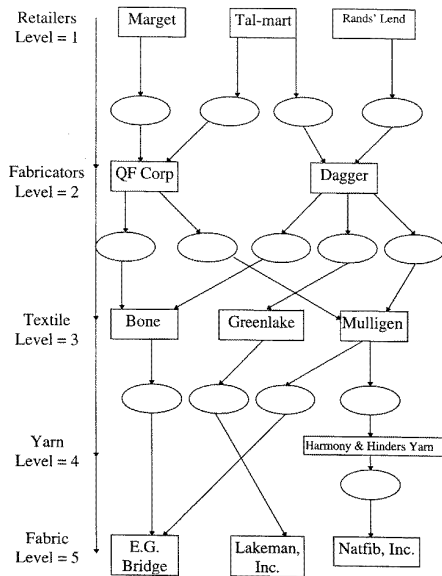


Figure 1. Demand information flow.

the textile industry to increase their competitiveness and responsiveness to changes in demand. The model discussed here is 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 modelled 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 customer) 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) fibre manufacturers. Each node N at level i represents a company that produces items numbered, for example, I_1, \dots, 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 customers and suppliers. This communication structure is depicted in figure 1.

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 fibre manufacturers.

Each arc in the layered communication graph is implemented by a mailbox. The mailbox resides at the value-added-network (VAN) 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 in that failures at the client or supplier should not lose the 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. Tal-mart) may have stores across the country, 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 (e.g. sales of waterproof caps). 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 concatenated 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. The total item record for 123 is computed by R 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 fabricator 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 have to 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 will be passed to the fabricator customers of M , who will first find the garments that have 456 in their BOM. They will 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 provide early warning of production malfunction. Note that 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 customer (retailer), the problem may be 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,

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

is in 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 an enterprise (node) in the DAMA project necessitates that the enterprise post its BOM records in an escrow account and that it update 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 that the relational

terms (e.g. structured query language (SQL), tuple) can be substituted by analogue 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:

$$(T, \text{sql-stmt1}, \text{condition}, \text{action}).$$

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-stmt1*) 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 will require some type of extended transaction†. 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:

$$\text{sql-stmt2}, \text{comparison-operator}, \text{core-set} \\ \text{or numeric-constant}.$$

Generally, the semantics are as follows. Denote the set of tuples returned by the DBMS in response to the query *sql-stmt1* by *S* (i.e. $S = \text{sql-stmt2}$). 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 labelled *sql-stmt1* is considered incomplete (i.e. missing data occur if set *U* stands in relationship *comparison-operator* to *core-set*). If so, the condition

† 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 [8].

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-stmt1* is considered, as in the comparison.

If either *sql-stmt1* 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 three.

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

$$T, \text{read-BOM-}x, \text{condition}, \text{approximation}, \text{adjustment}$$

is defined, where the condition is ($=, \emptyset$). 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:

$$\text{Select}^* \text{ from BOM where item number} = x.$$

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

$$T, \text{read-BOM-}xz, \text{condition}, \text{approximation}, \text{adjustment},$$

where the condition is ($=, \text{where-clause}$).

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

$$\text{Select} * \text{ from BOM where item number} = x \text{ or } z.$$

This SQL query reads the BOM of items *x* and *z*, and the MD condition indicates that the missing data condition is 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:

$$\text{store-id}, \text{item-number}, \text{period}, \text{units sold}, \text{units returned}.$$

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

$$\text{Select store-id}, \text{units sold}, \text{units returned} \\ \text{from DEMAND} \\ \text{where item-number} = x, \text{period} = 12/10/93 - 12/17/93.$$

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 tuples 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 activity 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.

3.4.1. Two types of approximation. The approach in section 3.4.1 assumes the use of the relational model, but could easily be extended to the object-oriented model. 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. The concepts build upon [13].

3.4.2. The relation used by the IFM for the approximation. Each MD statement has a linear combination relation called LCR . The relation 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.

A time-series approximation assumes that in addition to relations R (also called R_0) and *LCR*, there are time series relations, R_1, R_2, R_3 , etc. R_0 holds the latest version of each tuple; R_1 holds the version before latest for each tuple; R_2 holds the version before that, etc. Suppose the user wants to specify that if the tuple with key X is missing in R , it should be approximated as the tuple obtained as the linear combination $0.8*Y + 0.2*Z$; Y and Z are versions of X , 1 and 3 before latest, respectively. Next, the lc tuple $K = X$, $COEFF1=0.8$, $K1=1$, $COEFF2=0.2$, $K2=3$ is inserted into the relation *LCR*.

It is clear that this approach can be extended to accommodate a key, either $K1$ approximated by n tuples and key $K2$ approximated by m tuples with $m \neq n$ or $K1$ having a similarity approximation and $K2$ having a time-series approximation. For this purpose, it is easier if *LCR* is an object class rather than a relation.

It is also clear that if the coefficients are identical for all the lc tuples, they can be 'factored out' to save space. For example, the *LCR* relation can be represented by the coefficients tuple (0.8, 0.2) and the *KEYS* ($K, K1, K2$) relation.

3.4.3. Parameters. The following parameters pertain to the *LCR* relation. On the basis of the previous discussion, the first parameter of an *LCR* relation is the *TYPE* of approximation supported: time-series or similarity. The second parameter is the *STATUS* of the *LCR* relation. One possible status is *fixed*, which means that the lc tuple with key X is inserted in the *LCR* relation at the outset (i.e. when the MD statement is defined, even if the tuple with the key X is in R). If the status of the *LCR* relation is *fixed*, the *LCR* relation is static during run time (i.e. it does not change when the relation R changes). The second possible status is *dynamic*, which means that a tuple with key X is inserted in the *LCR* relation only when the tuple with key X is deleted from R . In this case, inserting the lc tuple is initiated by a trigger that fires whenever the tuple with key X is deleted from R . Conversely, whenever the tuple with key X is inserted in R , the tuple with key X is deleted from *LCR*.

The third parameter is the *INSERTION MODE*, which can be manual or automatic. *Manual insertion* means that the database administrator inserts the lc tuple manually. *Automatic insertion* means that the IFM computes the lc tuple. In this case, the MD statement should specify to the IFM which method to use to compute the lc tuple. One such method is outlined in section 3.4.4.

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 X is deleted, it triggers the initiation of a method that inserts a tuple with

key X in the *LCR* relation. 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 X 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* relation).

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* relation indicates that the R tuple with key X 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 X is missing. The second solution is to approximate the Y tuple (assuming that there is an lc tuple with key Y in the *LCR* relation) and then to use this approximation to approximate the R tuple with key X . 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. Instead of *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 X 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 $I = (i_1, i_2, \dots, i_k)$, 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 $A = (a_1, a_2, \dots, a_k)$ is normalized when A is multiplied by

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

The cosine function is adopted as a measure of the similarity between two normalized vectors; 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

$A = (a_1, a_2, \dots, a_k)$ and $B = (b_1, b_2, \dots, b_k)$ are two normalized vectors,

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

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. Before proceeding to the technical description it is important to consider the fact that business decisions may be made based upon approximations of data that are later found to diverge quite radically from what was estimated. Although this is true, without the information flow that DAMA should supply these same businesses would merely be in the same situation they are today: they have no information about the realized demand for their product. Business decisions are therefore currently based upon information that is guaranteed, at least some degree, to be erroneous. The POS information backflow improves the situation markedly.

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 , that was 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.

Syntactically, the adjustment is a triple $(Q, time\ interval, V)$, where Q and V are transactions. The semantics are as follows. When all the tuples identified as missing when *sql-stmt1* was issued are inserted in the database, transaction Q is executed. Intuitively, transaction Q performs the adjustment, and transaction V informs the 'central monitor' (e.g. DAMA, Inc.) of the regulation violation.

Transaction Q can be T when T is idempotent, which is defined as follows. A *database* is a set of data items x_1, x_2, \dots, x_k . A *transaction* is a program. Each invocation of the transaction receives some data items as *input* and produces some data items as *output*. The input set of data items is partitioned into *external* and *internal*

input data items. The external output is the set of data items output to the outside world, and the internal output is the set of database items written by the transaction. Transaction T is *idempotent* if for each invocation, the sets of internal input output data items are disjoint. For example, a transaction that *adds* x to the AMOUNT field of a tuple identified by key y (where x and y are parameters (i.e. external input)) is not idempotent because the tuple identified by key y (where x and y are parameters (i.e. external input)) is not idempotent. The reason is that the tuple identified by key y is both internal input and output. On the other hand, a transaction that *puts* x in the AMOUNT field of a tuple identified by key y (where x and y are parameters) is idempotent. The reason is that the tuple identified by key y is only in the internal output but not in the internal input. Observe that an idempotent transaction, if invoked multiple times with the same external input, produces the same output (both internal and external) each time. Observe also that the transaction that produces the demand information for the suppliers of a node can be written to be idempotent. The reason is that this transaction reads as input the total item records for a particular period and the BOMs, and it produces the item records for its suppliers. The input and output sets are disjoint because the input and output item numbers are disjoint.

Transaction T is idempotent. As a result, the adjustment can be performed by invoking T when the missing tuples are inserted in the database. The invocation uses the parameters of T at the original invocation (i.e. the invocation that gave rise to the detection of missing data). Presumably, missing data will not occur at this time; if they do, the adjustment is aborted, and the expiration time is reinstated. Observe that the approximate output of T written previously is replaced by the exact output. In addition, because the input and output of T are disjoint (idempotency), the exact output is unaffected by the fact that the approximate output was in the database at the time when T produced the exact output.

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[†].

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

[†] 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.

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 of the POS information backflow recommends that vendors enhance the existing level of DBMS functionality as follows. 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.

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).

3.8. Dama partners and other comments

The IFM material was made available for review by the database sub-task 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. The DAMA project also stipulated in 1994 that the concepts in this paper are still under investigation within the DAMA project and that drawing conclusions relative to any DAMA implementation would be misleading at this time.

3.8.1. 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, and today there is a business problem (wrong estimates of demand) because there is no information flow. Thus, the only

'flaw' in the IFM approach is that discussing it makes this business problem visible. Today 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 IFMs algorithms, however, is the flow of information. Thus, an alternative formulation of actions within the IFM 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 an automated version of business practices today. These approximations could then be used to approximate demand figures transmitted to companies at level $i + 2$. To the internal databases, the transaction stream would not look any different.

The issue, therefore, is not whether incorrect approximations exist, but who takes responsibility for the resulting uncertainty. The foregoing variant to the IFM algorithms makes each company responsible for approximating demand for its own products rather than accepting an approximation from another company.

3.8.2. Scope of deployment. The IFM, although well thought out, is a complex system, perhaps too complex for the small and even medium enterprises that numerically dominate the industry. The IFM was developed because the alternatives that used centralized or decentralized database management techniques were totally unsuitable because of the large transaction volumes and uncertainty of communications links. However, the IFM and a centralized transaction manager might be combined.

When DAMA is realized and a system exists that connects the industry, the small and medium enterprises will most likely use a VAN to implement their EDI transactions, including any transactions for demand. For each of these companies, the number of customers and suppliers will be relatively small. The VAN suppliers currently each have an internal database that manages EDI transactions; thus, in the case of met product demand, a large number of companies could delegate their data management responsibilities to the VAN provider. Thus, the VAN would become a broadcaster of met product demand information. The larger companies have existing methods of communicating. For these companies, the IFM provides a multicompany, multisector model of how a fully automated system could work.

Thus, the issue of IFM deployment is best managed with an 80/20 approach. Notionally, the 80% of the companies that create 20% of the data would use VAN services, and the 20% of the companies that produce 80% of the data would have the choice of using the VAN or some variant of the IFM process to transmit demand information directly.

3.8.3. Relationship to approximate transactions. The techniques which are discussed above include the use of approximations. Techniques such as those in [8] have been heretofore proposed to approximate data. The idea of the IFM differs, however, from these approaches because data

approximation is in the final analysis only a temporary issue. The net product demand data will within a fixed time frame, weeks, be 100% historically accurate or else irrelevant.

The information flow manager, however, is only superficially related to database level managers of approximate-value transactions. The initial data gathered at the retail store are traditional transaction level data. The information of interest, however, is not that of any individual transaction, but rather the count of transactions of the same type: how many of a certain type of product were sold in a certain date range. This information is then matched at the fabricator level with bill of material information to convert the count of products to an aggregate amount of product for each supplier. There is no attempt to convert each individual product sale transaction to an equivalent 'amount of raw material' sale. Only were such a level of detail retained would transaction level approximation be relevant. The transmission of data to the next levels of the manufacturing chain are also at a summary level.

The purpose of the information transmitted is to allow companies to make better estimates of demand for their product. The IFM guarantees that the approximation will always be accurate in value not because of the management of values *but because of the management of timing and completeness of data transmission*. The individual nodes use of approximations, or sales forecasts, in place of supplied demand information, is only provided to overcome the lack of some information from some sources at some times. Its generation is solely at the discretion of each individual company in the chain. It is quite possible, therefore, that a company will choose to provide its suppliers with only the information it has on hand, and include no approximations or forecasts for the demand data it has not received. The IFM does not set any standards or policies in this regard.

The reason that the IFM provides a reasonable estimate of demand is that it guarantees the performance of nodes that individually fail to perform within an agreed time frame. With the exception of the retail level, the failure of a node to perform triggers the re-transmission of data to a backup node, the one containing the BOMs and suppliers of the failed node. The backup node performs the necessary aggregation and transmission to the supplier nodes, who are expected to expedite processing of this backup information and ship it to their suppliers. Failure of a retail node to transmit data is handled in one of two ways. The industry association lawyers will telephone the retailer's lawyers if it is still in business. If it has closed its doors it will not be a source of new information anyway, so demand will shift to other retailers. The absence of demand data from a non-reordering retailer is of no practical concern to any of its former suppliers.

When backup data are transmitted, failure to expedite processing at a supplier node triggers expedited processing in turn at their backup nodes, a process that is repeated throughout the supply chain. Thus within a time frame expected to be two weeks, all of the transaction level data from functioning retail stores is counted and summarized

to all companies in the production process. Thus the correction of the estimate of demand is a matter of guaranteeing process completion and is independent of any data approximation techniques used by individual nodes. The only approximation issue for the IFM is temporal, whether the data generated at time t_1 are all available by time t_2 .

4. Conclusion

From its outset, the DAMA project envisioned a future where all the companies in all sectors of the American industry would be interconnected electronically so that they could collectively respond immediately to changing consumer demand. This immediate response requires that the estimated 10 000 retail companies, with their 100 000 stores, and their 26 000 suppliers have an up-to-date database of demand information. The companies are distributed across the nation, and they use a variety of heterogeneous computers and networks for data processing and communication. From a technical point of view, serious questions exist as to whether such a system could be built at all. On the one hand, the system was an order of magnitude large than current research and development prototypes for distributed databases; thus, its complexity was beyond the state of the art. On the other hand, the time-delays between updates were measured in days. All research and development for distributed databases during the last 15 years had been funded by government and commercial interests that wanted to reduce the lengths of transaction times a hundredfold, not increase them by that amount. Thus it was unclear whether DAMA's data could be managed.

The research effort that culminated in this description of the IFM was undertaken to find a new type of transaction, one that did not have the constraints associated with short-lived transactions of today's DBMSs but had more coherence than the 'anything goes' protocols associated with WFMs. The IFM successfully combines the logical coherence of transaction management with the flexibility of workflow. By using it, the DAMA project may ensure the technical viability of any agreements to automate the sharing of demand data within the entire ITC.

References

- [1] Alonso G, Agrawal D, El Abbadu A, Mohan C, Gunthor R and Kamath M 1995 FMQM: A persistent message-based architecture for distributed workflow management *Proc. IFIP WG8.1 (Trondheim, 1995) Information Systems Development for Decentralized Organizations* (London: Chapman & Hall) pp 1–18
- [2] Alonso G, Gunthor R, Kamath M, Agrawal D, El Abbadu A and Mohan C 1995 FMDC: handling disconnected clients in a workflow management system *Proc. 3rd Int. COOPIS (Vienna, 1995)* pp 99–110
- [3] Barbara D, Mehrota S and Rusinkiewicz M 1996 INCAs: managing dynamic workflows in distributed environments *J. Database Management* **7** (1) 5–15
- [4] Bernstein P A, Hadzilacos V and Goodman N 1987 *Concurrency Control and Recovery in Database Systems* (Reading MA: Addison-Wesley)

- [5] Biliris A, Dar S, Gehani N, Jagadish H V and Ramamrithan K 1994 ASSET: a system for supporting extended transactions *Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data (Minneapolis, MN)* (New York: ACM) pp 44–54
- [6] Dayal U, Hsu M and Ladin R 1990 Organizing long-running activities with triggers and transaction *Proc. 1990 ACM SIGMOD Int. Conf. on Management of Data (Atlantic City, NJ)* (New York: ACM) pp 204–14
- [7] Dayal U, Hsu M and Ladin R 1991 A transactional model for long running activities *Proc. 17th Int. Conf. on Very Large Data Bases* (San Mateo, CA: Morgan Kaufmann) pp 113–22
- [8] Elmagarmid A K (ed) 1991 *Database Transaction Models for Advanced Applications* (San Mateo, CA: Morgan Kaufmann)
- [9] Garcia-Molina H and Salem K 1987 Sagas *Proc. 1987 ACM SIGMOD Int. Conf. on Management of Data (San-Francisco, CA)* (New York: ACM) pp 249–59
- [10] Hsu M (ed) Special issue on workflows and extended transaction systems *Data Eng. Bull.*
- [11] Papadimitriou C 1986 *The Theory of Database Concurrency Control* (Rockville, MD: Computer Science Press)
- [12] Rusinkiewicz M and Sheth A 1993 Specification and execution on transactional workflows *Bellcore Technical Memorandum TM-ST5-023284*
- [13] Salton G and McGill M 1993 *Introduction to Modern Information Retrieval* (New York: McGraw Hill)
- [14] Stonebraker M, Aoki P, Devine R, Litwin W and Olson M 1994 Mariposa: a new architecture for distributed data *Proc. 1994 Data Engineering Conf.* (Los Alamitos, CA: IEEE Computer Society Press) pp 54–65
- [15] Stonebraker M, Devine R, Kornacker M, Litwin W, Pfeffer A, Sah A and Staelin C 1994 An economic paradigm for query processing and data migration in Mariposa *Proc. 3rd Int. Conf. on Parallel and Distributed Information Systems (Austin TX, 1994)* (Los Alamitos, CA: IEEE Computer Society Press) pp 58–67
- [16] Zachman J A 1987 A Framework for information-systems architecture *IBM Syst. J.* **26** 276–92