# CONSISTENCY AND CURRENCY IN FUNCTIONAL DATABASES[1]

by

Gary Lindstrom
Frances E. Hunt

UUCS-82-012
February 1982

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

## Abstract

We consider a hybrid model of databases, in which a functional component T is defined as an extension to an imperative component B. T is loosely coupled to B through a highly parallel function network N, which provides a simple failsafe test of whether an existing assignment of values to a given view (subset) T' of T is consistent. If this test fails, or when a "current" view of T' is desired, N can be requested to "refresh" T' so that its values become consistent with respect to the current assignment to B. These requests are serviced by N without danger of system deadlock.

[1]This work was supported in part by grant MCS-78-03832 from the National Science Foundation.

## Table of Contents

## List of Figures

## 1. A Hybrid Model of Databases

In this work, we seek to bring together "traditional" imperative (i.e. assignment-based) database mechanisms and newer functional (i.e. object-based) approaches. We postulate an imperative component augmented by a counterpart capturing information which may be functionally derived from values resident in various subsets of the imperative component's cells. Our objective in this approach is a smooth integration of the two components with low overhead. Moreover, we seek a formulation of this combination suitable for a highly distributed parallel implementation.

### 1.1. Functional Programming

Functional programming is a computational framework based on the use of functions, in the mathematical sense, to express the relationships on a set of values [1, 6]. This supports a high level programming methodology, in which systems are designed by refinement of function definitions, rather than by the direct specification of operator sequences.

Functional programming systems lend themselves naturally to implementation on distributed architectures, such as AMPS [8]. Such architectures are very attractive for database systems, since the multiple processors provide the potential for a high degree of concurrency and distribution in query processing. Moreover, with lazy evaluation [4, 5], as exemplified in the execution model of AMPS, the distinction between implicit and explicit data representation softens, permitting higher levels of abstraction in database programming.

### 1.2. Modeling Databases Functionally

Research in functional programming techniques applied to databases has been primarily concerned with language interfaces to conventional systems [2, 11]. Recently, however, increasing attention has paid to the problem of updating in functional database systems [3, 9]. In general, updating is accomplished by conceptually creating a new version of the entire database. Since physical re-creation of the entire database is economically undesirable, update functions must be written with considerable care and implementation awareness to achieve acceptably low copying.

### 1.3. Distributed Databases

Distributed database systems refer to multiple site and/or multiple copy systems. The latter approach uses redundancy to reduce communication overhead incurred when processing queries from physically separate locations. Reliability is also a factor for choosing a distributed system, since the malfunction of one site may be circumvented by rerouting requests through other working sites in the distributed network. Such systems may introduce extra levels of programming complexity, however, unless there exists an underlying support system which makes the distribution mechanisms transparent to the user [12].

Our approach concerns databases with a very special sense of distribution, namely that the imperative and functional components are logically separated and mediated by a mechanism well suited for either a teleprocessing protocol connecting the two, or a highly parallel local computer system. The principal novelty of the approach will be a derived sense of consistency and currency in the functional component, and a mechanism for efficiently updating cells in that component without resorting to special update functions.

## 2. Our Approach

We now outline our particular approach to the hybrid database model, emphasizing the external interfaces to the database presented by each of the two components. Section 3 will present an implementation of the desired connection between these two interfaces.

## 2.1. The Database Partitioning

Figure 2-1 depicts our hybrid model. The key components are:

- B: a "bottom end" comprising imperative cells $c_i$, $1<=i<=n$, and a lock manager $M_B$ controlling read/write access to these cells by transactions, and
- T: a "top end" comprising functional cells $f_j$, $1<=j<=m$, and a lock manager $M_T$ controlling read-only access to these cells by queries.

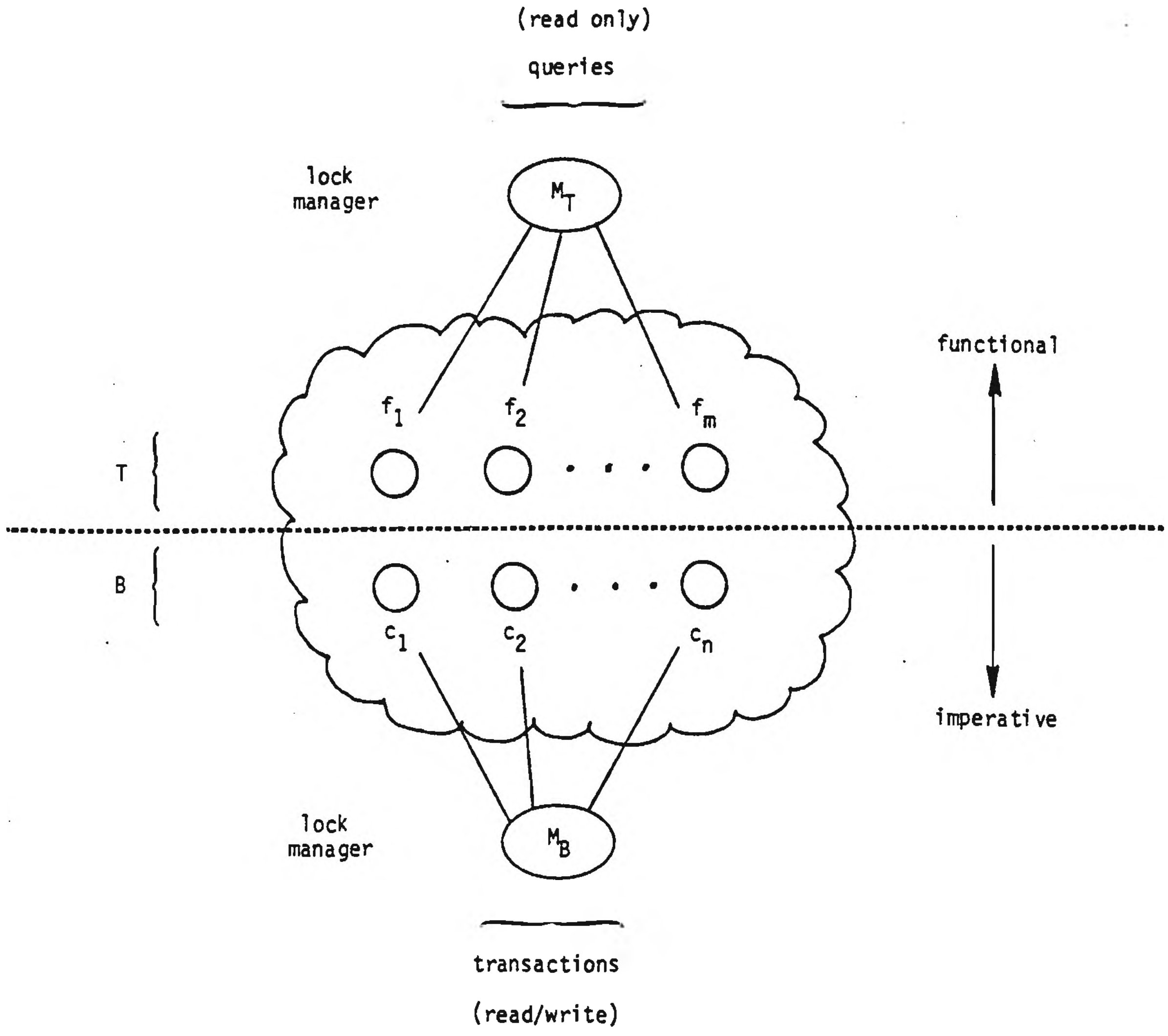

Figure 2-1: Hybrid database model.

## 2.2. Bottom End Behavior

The bottom end should be viewed as a conventional database supporting concurrent transaction processing through a lock manager guaranteeing serializability of all permitted transaction interleavings (e.g. as per [13] section 10.3).

### 2.2.1. Cells in B

At any moment each cell $c_i$ in B possesses a state and a value.

- States for each $c_i$ are selected from {unlocked, scheduled, queried, acknowledged}. Initially each $c_i$ is unlocked.
- A value binding for each $c_i$ in B is termed an assignment for B.

### 2.2.2. Transactions on B

Transactions on B are each sequences of read and write actions on the $c_i$ cells. The transactions are assumed to perform reads or writes on each $c_i$ cell only after that cell has been appropriately locked (WLOCK for writes, and RLOCK or WLOCK for reads). Moreover, we assume a "shadow" update technique, whereby all writes done by a transaction are buffered until the transaction completes, at which time a unitary UNLOCK is performed. This causes all cells WLOCKed by the transaction to become simultaneously unlocked and updated.

### 2.2.3. Lock Manager $M_B$

The exact strategy and implementation of the bottom end lock manager $M_B$ is unimportant to us here. We assume $M_B$ maintains its own internal table of which cells are WLOCKed and RLOCKed. However, in support of our functional extension, we require $M_B$ to coordinate its actions with our postulated $c_i$ states as follows:

1. $M_B$ will grant a WLOCK on a cell $c_i$ only when it is in state unlocked. This causes the cell's state to become scheduled.
2. $M_B$ will permit a transaction to terminate (i.e. perform its UNLOCK) only when all its WLOCKed $c_i$ are in state acknowledged. When this condition is fulfilled, the following actions transpire

indivisibly:

- all cells WLOCKed by the transaction are updated, and
- all locked cells $c_i$ are put into state unlocked.

Note that RLOCKs are handled directly by $M_B$, and require no cell state changes. (The role of state queried will become evident in section 3.2.2.)

### 2.2.4. Consistency of Assignments to B

While we have left the exact nature of the transactions on B unspecified, we do assume they preserve all essential consistency constraints on B, whatever these may be. If we assume the initial assignment to B is consistent, we have directly a sense of consistency on later assignments to B:

Definition 1: Let b be an assignment to B. If b can be obtained through a serial application of transactions to B, b is a consistent assignment.

### 2.2.5. Currency of Assignments to B

In the obvious sense, we say the assignment of values to B existing at any given moment is the current one.

Theorem 2: The current assignment to B is always consistent.

Proof: Follows immediately from the transaction serialization effect of $M_B$, and the shadow update effect.

## 2.3. Top End Behavior

### 2.3.1. Cells in T

Each $f_j$ has an associated function $F_j$ determining its desired value in terms of assignments to some minimal subset ("view") $B_j$ of B. We term $B_j$ the support of $f_j$. We assume each $f_j$ is initialized to the functional image under $F_j$ of the initial assignment to B.

Each cell $f_j$ has a state selected from {evaluated, retracted, locked, or demanded}. Initially each $f_j$ is in state evaluated.

### 2.3.2. Queries on T

In contrast to the bottom end where direct updating is supported, we assume the top end is dedicated to read-only querying and reporting. Hence a query on T involves simply reading an assignment of values to a subset (again, "view") T' of T. Following the model of shadow updating on B, we assume queries on T' do not actually read the $f_j$ until the query is completed, at which time a block containing the existing values assigned to T' is delivered. For simplicity, we assume further that queries on T are handled sequentially.

### 2.3.3. Lock Manager $M_T$

Under these assumptions, it would seem no lock manager is needed for T at all. However, in preparation for our implementation of the interface between T and B, we define a top end lock manager $M_T$ nevertheless.

Each top end query comprises a sequence of RLOCK $f_j$ requests, terminated by a unitary UNLOCK. $M_T$ services these requests as follows:

- on RLOCK $f_j$ requests:
    * if $f_j$ is evaluated, $M_T$ puts it into locked.
    * if $f_j$ is retracted, $M_T$ puts it into demanded.
- on UNLOCK requests: $M_T$ waits until all cells in the query view are locked. It then reads the values of RLOCKed cells and delivers their values in a block.

### 2.3.4. Consistency of Assignments to T

The functionality of T induces a sense of consistency on assignments to its cells in terms of consistent assignments to B.

Definition 3: Let T' be a view of T. We say an assignment of values to T' is consistent if it is the functional image (under the $F_j$'s associated with the cells in T') of a consistent assignment to B.

### 2.3.5. Currency of Assignments to T

Definition 4: A value assigned to an $f_j$ is said to be current if it equals $F_j$ applied to the current assignment to its support.

Definition 5: Let T' be a view of T. We say an assignment of values to T' is current if the value of each $f_j$ is current.

Note that a current assignment to any top end view is always consistent.

## 2.4. System Objectives

Having established the basic properties of the top and bottom ends of our hybrid model, we now pose the behavioral objectives for our composite system.

- Top end consistency test: We desire a simple test for consistency of existing top end view assignments. This test should be:

  * local, i.e. involve only the states of cells in T, and

  * fail-safe, in the sense that if the test is positive, consistency of the existing view assignment is guaranteed.

- Top end currency test: Similarly, there should be a local and fail-safe test for the currency of any existing assignment to a view T' of T.

- Finite time querying: If an existing assignment to a view T' is judged to be inconsistent, or a current (and consistent) new assignment to T' is desired, this should be obtainable after finite delay, no matter how busy the bottom end may be at the time.

- Overhead damping: If activity at either the top or bottom end subsides, the time overhead incurred at the opposite end due to their functional relationship should approach zero.

- Liveness: It should be impossible for either end to become deadlocked.

- Loose coupling: The functional linkage between the two ends should be message based, and permit a highly distributed, parallel implementation.

## 2.5. An Example Application

As an illustration, we consider a rudimentary parts jobber application. The bottom end of our model corresponds to the Shipping Dock, where parts are received and shipped. The top end corresponds to the Executive Suite, where various inventory and sales reports are produced.

For simplicity, we assume only two parts are kept in stock, and they are received and shipped in single units. Repricing is limited to a unit increase. Bottom end transactions comprise $\text{Receive\_part}_i$, $\text{Ship\_part}_i$, and $\text{Reprice\_part}_i$ (see fig. 2-2). Top end queries include cumulative revenue, current price list, biggest seller to date, and value of inventory on hand (see fig. 2-3).

<u>cells</u>

| | |
|---|---|
| $P_{bot,1}$: | price of part 1 |
| $P_{bot,2}$: | price of part 2 |
| $O_1$: | on-hand quantity of part 1 |
| $O_2$: | on-hand quantity of part 2 |
| $N_1$: | quantity of part 1 previously shipped |
| $N_2$: | quantity of part 2 previously shipped |
| $R_{bot}$: | revenue generated from parts shipped thus far |

<u>transactions</u>

```
Ship_part$_i$:
    WLOCK $N_i$;
    $N_i := N_i+1$;
    WLOCK $O_i$;
    $O_i := O_i-1$;
    RLOCK $P_{bot,i}$;
    WLOCK $R_{bot}$;
    $R_{bot} := R_{bot}+P_{bot,i}$;
    UNLOCK
```

```
Receive_part$_i$:
    WLOCK $O_i$;
    $O_i := O_i+1$;
    UNLOCK

Reprice_part$_i$:
    WLOCK $P_{bot,i}$;
    $P_{bot,i} := P_{bot,i}+1$;
    UNLOCK
```

Figure 2-2: Bottom end in parts jobber example.

## 3. Our Solution

An implementation of this hybrid model meeting our system objectives can be obtained through use of a special variety of function network interfacing the two ends.

cells

| | | |
|---|---|---|
| $R_{top}$ | = | $R_{bot}$ |
| $P_{top,1}$ | = | $P_{bot,1}$ |
| $P_{top,2}$ | = | $P_{bot,2}$ |
| $B$ | = | if $N_1 > N_2$ then 1 else 2 |
| $V$ | = | $O_1*P_{bot,1} + O_2*P_{bot,2}$ |

queries

| | |
|---|---|
| Revenue: | $\{R_{top}\}$ |
| Price List: | $\{P_{top,1}, P_{top,2}\}$ |
| Biggest Seller: | $\{B\}$ |
| Inventory Value: | $\{V\}$ |

Figure 2-3: Top end in parts jobber example.

## 3.1. Function Network

We place a function network N between the cells $c_i$ on the bottom end and the cells $f_j$ at the top end. Associated with each $c_i$ is an INPUT node in N, and with each $f_j$ an OUTPUT node in N (see fig. 3-1).

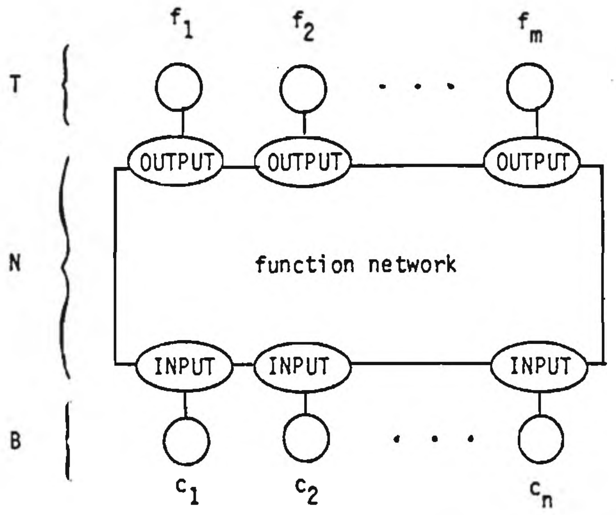

Figure 3-1: Function network placement.

We can think of the function network N as a conduit for messages which pass between INPUT nodes and OUTPUT nodes. When evaluated, the OUTPUT nodes reflect a functional image of the values the INPUT nodes held at the time of the computation of the OUTPUT values. The actual computation may be envisioned as a two-phase process. OUTPUT nodes transmit demand messages downward through N in order to obtain the values needed for their computation. In response, INPUT nodes pass value messages upward through N to the OUTPUT nodes which demanded them.

### 3.1.1. INPUT/OUTPUT Node States

For the purposes of this discussion, we will ignore the inner workings of N in order to concentrate on the dependency relationships between INPUT and OUTPUT nodes.

An INPUT node has a state selected from {_evaluated_, _retract-requested_, _retract-acknowledged_, _demanded_}. Initially all INPUT nodes are in _evaluated_. Each INPUT node cycles through these states as follows:

- _evaluated_ to _retract-requested_: an external request has been made to change an INPUT value. A "retract-request" message is sent through N to notify all OUTPUT nodes dependent on the given INPUT node.
- _retract-requested_ to _retract-acknowledged_: N has routed the "retract-request" message to all nodes potentially affected by this change, and has received "retract-acknowledged" messages from all such nodes.
- _retract-acknowledged_ to _demanded_: a demand for this INPUT node has been routed through N as a result of a demand on an OUTPUT node.
- _demanded_ to _evaluated_: a value is released from this INPUT node.

An OUTPUT node has a state selected from {_evaluated_, _retracted_, _demanded_}. Initially all OUTPUT nodes are _evaluated_. Each OUTPUT node cycles through these states as follows:

- _evaluated_ to _retracted_: a "retract-request" message was received from N as a result of a proposed change to one of the INPUT nodes upon which this OUTPUT node is functionally dependent. This transition also causes a "retract-acknowledged" message to be sent back through N.
- _retracted_ to _demanded_: an external request has been made for the value of this OUTPUT node. This causes a demand to be routed

through N.

- demanded to evaluated: a value has been computed and received from N thereby satisfying the demand.

### 3.1.2. Axiomatic Behavior

The essential properties of N can be characterized by the following propositions, which we take here to be axioms:

Proposition 6: When an OUTPUT node makes the transition from demanded to evaluated, the value received is a functional image of the values resident in the set of INPUT nodes upon which the OUTPUT node is functionally dependent. Furthermore, those INPUT nodes will be either evaluated, or retract-requested.

Proposition 7: When an INPUT node undergoes retract-requested to retract-acknowledged, none of the OUTPUT nodes depending on it may be evaluated.

Proposition 8: A demand for a value at an OUTPUT node will be satisfied after a finite delay.

Proposition 9: Demand has priority over retraction in N. That is, suppose an INPUT node undergoes a transition to retract-requested at approximately the same time a functionally related OUTPUT node undergoes a transition to demanded. Then the OUTPUT node will reach evaluated before the INPUT node reaches retract-acknowledged.

Proposition 10: A damping effect occurs at the OUTPUT (INPUT) end of N when the messages passing through N are primarily demands (retracts). This happens as a result of a demand (retract) being satisfied immediately upon encountering an already evaluated (retracted) node within N.

### 3.1.3. Details of Mechanism

The message passing behavior of the network is based on the concept of incremental recomputation on data-flow graphs as developed in [10]. Formal details and proofs about the retraction mechanism will be available in [7].

## 3.2. Network Interfaces

The INPUT and OUTPUT nodes described in the previous section have a direct relationship with the bottom end and top end database cells, respectively. These cells act as the external influences which create demands at OUTPUT nodes and cause retraction at INPUT nodes.

### 3.2.1. Top end

Let the OUTPUT node associated with a top end cell $f_j$ be denoted $O_j$. The state transitions which $f_j$ undergoes (see fig. 3-2) are related to those of $O_j$ as follows:

- evaluated to retracted:

  * caused by: $O_j$ undergoing evaluated to retracted.
  * causes: (No direct effect.)

  The OUTPUT node associated with this cell has received a "retract-request" message signifying that the value contained in this cell is based on one or more INPUT values about to become outdated.

- evaluated to locked:

  * caused by: RLOCK granted by $M_T$.
  * causes: (No direct effect.)

  A query has been made involving $f_j$. The cell is prohibited from honoring subsequent retraction requests until it once again becomes unlocked (i.e. evaluated).

- retracted to demanded:

  * caused by: RLOCK granted by $M_T$.
  * causes: $O_j$ to undergo retracted to demanded.

  A query has been made involving the value in this cell. Since the value in residence is outdated, a new value must be computed.

- demanded to locked:

  * caused by: $O_j$ undergoing demanded to evaluated.

* causes: (no direct effect, unless this is the final $f_j$ in the query view to become locked --- see next transition.)

A value has been computed by the network, and installed in the OUTPUT node associated with this top end cell.

- locked to evaluated:

  * caused by: All $f_j$ in the query view being locked, and UNLOCK granted by $M_T$.

  * causes: assignment of value of $O_j$ to $f_j$, and delivery of query value block.

All cells involved in the query have current values and the query has been satisfied. Locks granted for that query are thereby relinquished.

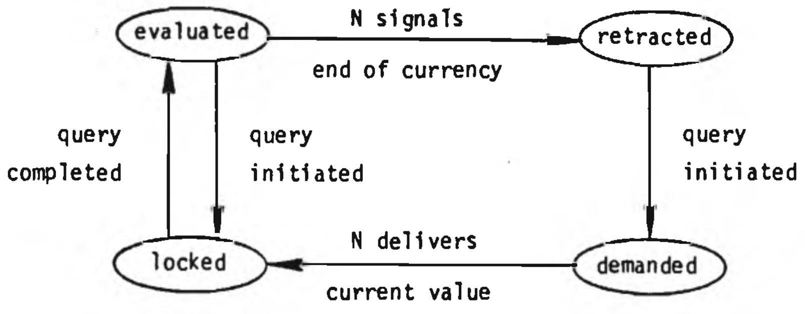

Figure 3-2: Top end state transitions.

### 3.2.2. Bottom end

Similarly, let the INPUT node associated with a bottom end cell $c_i$ be $I_i$. The state transitions which $c_i$ undergoes (see fig. 3-3) are related to those of $I_i$ as follows:

- unlocked to scheduled:

  * caused by: WLOCK granted by $M_B$.

  * causes: If $I_i$ is in evaluated, it becomes retract-requested.

The lock manager has requested a WLOCK on this cell. The associated INPUT node (if evaluated) becomes retract-requested and sends a "retract-request" message through N.

- scheduled to acknowledged:

  * caused by: $I_i$ being retract-acknowledged.

  * causes: (No effect, unless this is the final transition permitting $M_B$ to grant an UNLOCK --- see below.)

  The associated INPUT node is or has become retract-acknowledged, signifying that all dependent OUTPUT nodes have been notified of the impending change.

- acknowledged to unlocked:

  * caused by: $M_B$ performing an UNLOCK.

  * causes: (No direct effect on $I_i$.)

  The cell is given a new value.

- acknowledged to queried:

  * caused by: $I_i$ undergoing retract-acknowledged to demanded.

  * causes: Assignment of the value of $c_i$ to $I_i$; $I_i$ undergoing demanded to evaluated, and then to retract-requested. Cell $c_i$ then undergoes queried to scheduled.

  A demand has arrived from the network N as a result of a request for current information at the top end. The outdated value is returned, and the retraction process to confirm the WLOCK on $c_i$ must be re-initiated.

If $I_i$ undergoes retract-acknowledged to demanded when $c_i$ is unlocked, an assignment of $c_i$ to $I_i$ occurs without $c_i$ changing state. This is accompanied by $I_i$ undergoing demanded to evaluated.

Since priority is given to the requests for current information at the top end of the network, there may be some delay in allowing bottom end cells to be updated. Note that when a cell $c_i$ undergoes acknowledged to unlocked the associated INPUT node remains in the retract-acknowleged state until a demand arrives from N. This allows many updates to the bottom end cells, without interaction with N, until the next currency request arrives.

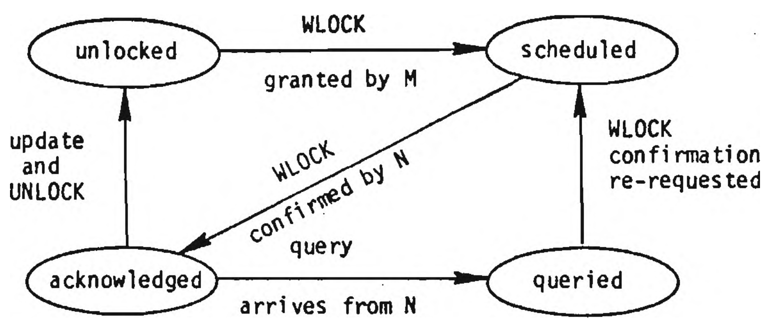

Figure 3-3: Bottom end state transitions.

## 3.3. An Illustration

We illustrate the interaction between the top end queries and bottom end transactions through a brief scenario taken from the parts jobber example (see fig. 3-4). In this illustration, we assume all top end cells are evaluated except V, which we assume to be retracted. Two special effects are worth noting:

- The immediate transition of $O_1$ from scheduled to acknowledged in transaction $t_1$. This results from V (the only top end cell dependent on $O_1$) being already retracted.
- The transitions of $P_{bot,1}$ from acknowledged to queried to scheduled to acknowledged again, reflecting the deferral of $t_2$ due to the higher priority access of $P_{bot,1}$ by N in the service of $q_1$.

## 4. System Performance

We now argue that our composite system design, with N mediating the interactions between T and B, achieves our objectives as enumerated in section 2.4.

### 4.1. Individual Cell Consistency and Currency

Lemma 11: Whenever an INPUT node $I_i$ is evaluated or retract-requested, its value equals that of its associated bottom end cell $c_i$.

| $t_1$: Ship_part$_1$ | $t_2$: Reprice_part$_1$ | $q_1$: $\{P_{top,1}, P_{top,2}\}$ |
|---|---|---|
| $t_1$: $N_1$: U to S | | |
| $t_1$: $O_1$: U to S | | |
| $t_1$: $O_1$: S to A | | |
| | | $t_1$: B: E to R |
| | $t_2$: $P_{bot,1}$: U to S | |
| $t_1$: $N_1$: S to A | | |
| | | $t_2$: $P_{top,1}$: E to R |
| | $t_2$: $P_{bot,1}$: S to A | |
| | | $q_1$: $P_{top,1}$: R to D |
| | | $q_1$: $P_{top,2}$: E to L |
| | $q_1$: $P_{bot,1}$: A to Q | |
| | | $q_1$: $P_{top,1}$: D to L |
| | $t_2$: $P_{bot,1}$: Q to S | |
| | | $q_1$: $\left.\begin{matrix}P_{top,1}\\P_{top,2}\end{matrix}\right\}$ L to E |
| | | $t_2$: $P_{top,1}$: E to R |
| | $t_2$: $P_{bot,1}$: S to A | |
| | $t_2$: $P_{bot,1}$: A to U | |
| $t_1$: $R_{bot}$: U to S | | |
| | | $t_1$: $R_{top}$: E to R |
| $t_1$: $R_{bot}$: S to A | | |
| $t_1$: $\left.\begin{matrix}N_1\\O_1\\R_{bot}\end{matrix}\right\}$ A to U | | |

| | |
|---|---|
| U = unlocked | E = evaluated |
| S = scheduled | R = retracted |
| Q = queried | D = demanded |
| A = acknowledged | L = locked |

Figure 3-4: Jobber example scenario.

Proof: By the provisions of section 3.2.2, whenever $I_i$ enters <u>evaluated</u> its value is "refreshed" by $c_i$. This is the only occasion for the value of $I_i$ to change. Hence in <u>retract-requested</u> $I_i$ has the same value it possessed in its preceding <u>evaluated</u> state (recall the states of $I_i$ are cyclic). If the value of $c_i$ is constant until $I_i$ becomes <u>retract-acknowledged</u>, we are done. But this is clear, since $c_i$ changes value only when it undergoes <u>acknowledged</u> to <u>unlocked</u>, which can only occur when $I_i$ is in <u>retract-acknowledged</u>.

Theorem 12: The existing value in any individual top end cell $f_j$ is always consistent.

Proof: Let the OUTPUT node associated with $f_j$ be $O_j$. The value at $f_j$ clearly arose from $O_j$. By Proposition 6, when that value was delivered at $O_j$ it was the functional image of values existing at the INPUT node associated with the support of $f_j$. By the same Proposition, each of those INPUT nodes were in states <u>evaluated</u> or <u>retract-requested</u> at that time. Hence by Lemma 11 each of those INPUT nodes possessed values equal to that of their associated bottom end cells. Therefore that value is consistent as an assignment to $f_j$.

Theorem 13: If a top end cell $f_j$ is <u>evaluated</u>, its value is current.

Proof: By the reasoning in the proof of Theorem 12, whenever $O_j$ receives a new value it is current. But $f_j$ receives a new value only when it becomes <u>evaluated</u>, which occurs only when $O_j$ receives a new value. Now consider when that value becomes no longer current. A bottom cell in its support must have undergone <u>acknowledged</u> to <u>unlocked</u>, which implies its associated INPUT node must have undergone <u>retract-requested</u> to <u>retract-acknowledged</u>. By Proposition 7, $O_j$ must be no longer <u>evaluated</u> at that moment, which implies $f_j$ is no longer <u>evaluated</u>.

## 4.2. Consistency and Currency of Larger Views

Theorem 14: Let T' be a view of T. If all $f_j$ in T' are in state <u>evaluated</u>, the existing assignment to T' is current (and consistent).

Proof: Since every cell in T' is in <u>evaluated</u>, they all are current. Hence the overall view T' is current.

Theorem 15: Let T' be a view of T. An existing consistent assignment to T' remains consistent until a $f_j$ in T' undergoes a transition from <u>demanded</u> to <u>evaluated</u>.

Proof: Trivial, since no change is made to an existing assignment to T' until one of its $f_j$ receives a new value, which only occurs during the transition specified.

## 4.3. Querying

Theorem 16: Let T' be a view of T. Every read query on T' results in a current assignment to T' in finite time.

Proof: Follows from Propositions 8 and 9, and the fact that INPUT cell transitions from <u>demanded</u> to <u>evaluated</u> take place without interference from $M_B$.

## 4.4. Liveness

Theorem 17: The overall hybrid database system is deadlock free.

Proof: By Theorem 16, the top end cannot become deadlocked. Similarly, the bottom end cannot become deadlocked since the only effect of its connection to T is the occasional "forced" transition of a $c_i$ from <u>acknowledged</u> to <u>scheduled</u>. But this simply has the effect of deferring the transaction involved, and does not introduce new locking interactions among the bottom end transactions.

## 4.5. Damping

Theorem 18: If top end queries cease, bottom end transactions eventually incur no delays due to the presence of the top end.

Proof: Eventually all INPUT nodes will stabilize at <u>retract-acknowledged</u>, by Proposition 10, and bottom end cell transitions from <u>scheduled</u> to <u>acknowledged</u> will be immediate, without reversion to <u>scheduled</u> due to queries arriving from above.

Theorem 19: If bottom end transactions cease, top end queries eventually incur no delays due to the presence of the bottom end.

Proof: Similar.

We note informally that the implementation of N described in [10] provides a form of continuity between these two extremes, whereby the more active end of the database tends to receive proportionally lower overhead.

## 5. Conclusion

We have presented a hybrid model of database systems, combining a traditional imperative component with a more modern functional component. A notion of consistency and currency in the functional component has been presented, as well as a querying mechanism ensuring currency. The updating of cells in the functional component is achieved through a distributed form of incremental recomputation on data-flow graphs, and requires no special update functions.

If an active function net N is not available, the method may be adapted to a conventional two-way communication channel. Under this variation, the network would represent identity functions between top and bottom cells, and a special form of multiple copy distributed database model would result. All functional computation would then be placed in the top end view consumption; nevertheless, all system objectives would still be met. Moreover, a highly efficient communications protocol would be obtained, whereby the traffic over the channel would be limited to messages known to be essential to currency requests at the top end.

### 5.1. Limitations

This approach favors querying on the functional component, at the expense of transaction delay on the imperative component. A special, but familiar, two-phase locking protocol is required on the imperative component. System reliability is not considered, nor are distributed implementations of the two database components themselves. Query processing on the functional component is handled serially.

## 5.2. Possible extensions

Future research is needed on weakening some of these limitations. For example, it seems clearly possible to adapt the functional component to permit overlapped query processing. Another possibility might be to introduce top end value time stamping, which would associate with each value a pair of times [$t_{low}$, $t_{high}$] bracketing the interval over which the value was known to be current. Non-current existing multiple cell views could then indicate consistency by possessing non-null time bracket intersections.

A more challenging goal would be to establish greater processing symmetry between the two components, e.g. by permiting the functional component to do direct updating of imperative cells. A related extension would be to support individual cell unlocking in the imperative component, perhaps while retaining the two-phase protocol. However, preliminary investigations indicate that either of these last two extensions seems to invite either undetectable inconsistency in the functional component, or system deadlock.

## REFERENCES

[1] J. Backus.
Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.
_Communications of the ACM_ 21(8):613-641, August, 1978.

[2] P. Buneman and R.E. Frankel.
FQL - A functional query language.
In _ACM Sigmod_, pages 52-58. May-June, 1979.

[3] R. Doany.
Implementation of a network database using a function graph language.
Master's thesis, University of Utah, Dept. of Computer Science, June, 1981.

[4] D.P. Friedman and D.S. Wise.
_CONS should not evaluate its arguments._
Edinburgh University Press, 1976, pages 257-284.

[5] P. Henderson and J.H. Morris, Jr.
A lazy evaluator.
In _Proc. Third ACM Conference on Principles of Programming Languages_, pages 95-103. 1976.

[6] P. Henderson.
_Functional programming._
Prentice-Hall, 1980.

[7] Frances E. Hunt.
_Applicative Updating and Provisional Computation in Functional Programming._
PhD thesis, Computer Science Dept., Univ. of Utah, 1982.
forthcoming.

[8] R.M. Keller, G. Lindstrom, and S. Patil.
A loosely-coupled applicative multi-processing system.
In _AFIPS_, pages 613-622. AFIPS, June, 1979.

[9] R. M. Keller and G. Lindstrom.
_Toward function-based distributed database systems._
Technical Report UUCS-82-100, University of Utah, Computer Science Department, Jan., 1982.

[10] G. Lindstrom and R. Wagner.
Incremental recomputation on data-flow graphs.
In Homstrom, et al. (editors), _Symposium on functional languages and computer architecture_, pages 472-489. Laboratory on Programming Methodology, Department of Computer Sciences, Chalmers University of Technology and Goteborg University, June, 1981.

[11] D.W. Shipman.
The functional data model and the data language DAPLEX.
ACM TODS 6(1):140-173, March, 1981.

[12] I.L. Traiger, J.N. Gray, C.A. Galtieri, B.G. Lindsay.
Transactions and consistency in distributed database systems.
Technical Report RJ2555, IBM Research Lab., San Jose, CA, 1979.

[13] J.D. Ullman.
Principles of database systems.
Computer Science Press, 1980.