# DATA CONSISTENCY IN HARD REAL-TIME SYSTEMS†

*N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings*

*Real-Time Systems Group*
*Department of Computer Science*
*University of York*
*June 1993*

*ABSTRACT*

The incorporation of database technology into hard real-time systems presents an important but difficult challenge to any system designer. Notions of consistency and failure atomicity are attractive for architects of dependable hard real-time applications. Unfortunately the hard real-time requirements of bounded access times are not easily imposed upon the concurrency control methods usually found in current database models. The model of data consistency presented here uses temporal consistency constraints in order to ensure that data values are *sufficiently* recent to be usable. Blocking is avoided by imposing the restriction that only one type of transaction can write to a particular data object, and by using a non-blocking access mechanism. Data itself is considered to be either perishable or non-perishable.

## 1. INTRODUCTION

Traditionally, databases are used in systems which either do not operate in real time, or where quick response is desirable but not strictly necessary. Incorporating databases into hard real-time systems necessarily imposes stricter timing constraints, such that the

† This work is supported in part by the UK Information Engineering Advanced Technology Programme Grant, GR/F 35920/4/1/1214.

failure to meet a constraint can have catastrophic consequences. Formally, the correct functioning of the system depends on timely operation as well as on functionally correct operation.

A number of algorithms have been proposed and used in order to improve the performance of conventional databases. Generally, these rely on one of two techniques. Firstly, the consistency constraints which apply to the database, and to the views of it which can be obtained, may be weakened; these are generally known as *pessemistic* schemes. Examples include *multiversion serialisability* [Pap84, Agr89], where older versions of data objects are maintained for read-only transactions; *semantic atomicity* [Cel88], where the semantics of the transactions are exploited; *multilevel atomicity* [Lyn83] which is a generalisation of semantic atomicity; and *weak correctness* [Sha88a, Sha88b], where some transactions are allowed to see the intermediate results of other transactions. Secondly, schemes exist which take advantage of typical database access patterns, so that transactions may proceed more speedily than otherwise, on the expectation that no conflicts with other transactions will arise; these are generally referred to as *optimistic* schemes. If such conflicts do occur, then transactions may have to be aborted and restarted. Such a scheme is described by Kung and Robinson [Kun81].

However, the above methods are characterised by the fact that the performance improvement is statistical in nature, and cannot be guaranteed. Hence, while they may be beneficial in (soft) real-time systems, they are not sufficient for use in hard real-time systems. In this paper, we present a database model which can be used in a class of hard-real time systems, and which takes advantage of the characteristics of those systems. The temporal requirements of the database are formalised, and lead to scheduling constraints on the transactions which access the database.

For the purposes of this paper, we define the term *database* as follows:

**Defn (1)***: A database is a structured collection of data items, where specified consistency constraints between data items must be maintained.*

Note that we do not include associated systems (such as data base management systems) within this definition.

Section 2 of this paper presents some characteristics of conventional and of hard real-time databases. Section 3 then proposes a model for hard real-time databases, and section 4 defines database consistency for this model. An outline of the way in which database consistency requirements can be transformed to transaction timing is given in section 5, followed by a simple example in section 6. Conclusions are presented in section 7.

## 2. CHARACTERISTICS OF DATABASES

The types of systems in which conventional databases are used are generally different from those in which hard real-time databases are used. In this section we outline some of these different characteristics.

### 2.1. Conventional Databases

• Conventional databases have very large numbers of data objects, and the number of data objects may not be known in advance. For example, a banking system may have hundreds of thousands of customer accounts, which may be created and deleted. In principal, the size of the database is unbounded.

• A particular type of transaction may access any (possibly unbounded) subset of the data objects. In the banking example, a transaction which transfers money between accounts may access any pair of accounts. Further, the particular subset may be dependent on parameters passed to the transaction.

• The arrival of transactions cannot, in general, be predicted. Although some banking transactions may occur at regular intervals, such as accrual of interest, the transfer transaction will be invoked as and when customers request it.

• The data held in the database is generally discrete and is changed in a stepwise manner. Further, there is usually at best limited scope for using anything other than the most up-to-date value for a particular data object. For instance, although a customer enquiry as to the state of an account might be satisfied by the value at the close of business the previous day, transfer and interest transactions would not.

• The databases are typically *event* driven, that is, changes are made in response to events in the environment (for example, the occurence of a transfer request).

• The *consistency* of the database, and the *correctness* of the transactions which operate on it, are defined purely over the values of the data objects. Hence, in a banking system, money must neither be lost nor created, but there is no requirement for a transaction to be performed within a particular limited time.

• The problem of scheduling multiple concurrent transactions is one of performing those transactions correctly. While it may be desirable that the transactions are performed as quickly as possible, this desire is secondary to the correctness. Hence, developments in scheduling aim to statistically improve throughput.

### 2.2. Hard Real-Time Databases

• Hard real-time systems have a relatively small number of data objects, limited perhaps to a few thousand. More importantly, the number is exactly known in advance, or at least is bounded. If this were not so then it would never be possible to guarantee any set of timing requirements.

• Each transaction in a hard real-time system will access a bounded subset of the data objects, with the particular subset being known in advance. For instance, a transaction might read satellite, inertial and sonar position values, to derive a more accurate calculated position.

• The design of the system may require that particular transactions are invoked with some regularity, and the implementation may allow the invocation of others to be made regular.

Hence, a transaction which fetches a satellite position may have to execute in synchronisation with the satellite broadcast, while the transaction which combines the positions can be made to run with some sufficient frequency.

• Hard real-time systems are typically embedded as the control element within some system which is essentially continuous in nature. Although the system's calculated ship position might be updated at regular intervals, and hence steps through a series of values, it is reflecting a continuous variable. Also, the system may be able to perform adequately provided that the values available for particular data objects are sufficiently fresh, where *sufficiently* is defined by the dynamics of the environment.

• The databases tend to be *state* driven, that is, the values stored in the data objects tend to reflect the state of the environment, for instance, the position of the ship.

• While the notions of consistency and correctness with respect to data values may still be applicable, there is an additional requirement concerning time. Hence, data objects may have to be updated at intervals, or in orders, that are temporally constrained. In the ship system, it might be desirable that the satellite, inertial and sonar position systems are sampled at about the same time, and that the calculated position has bounded freshness.

• The transactions must be scheduled such that both the data and the temporal constraints are met. Although this is a more complicated scheduling problem, especially as the time available in which to make scheduling decisions is also limited (unless the system is completely pre-scheduled), it is sufficient to find any schedule which meets the constraints. Statistical performance improvements beyond this do not necessarily confer any benefit.

### 2.3. Consequences for Hard Real-Time Databases

The above points have consequences, beneficial or otherwise, for hard real-time systems which embody databases. Note that the term *analysis* is used in the following to refer to the act of determining whether a system is guaranteed to meet all its timing requirements.

The necessity of providing timely response to events in the system's environment makes the scheduling problem much harder. Hence, although the general problem of deciding if a schedule for a conventional database is serialisable is NP-complete [Kun79], there are many algorithms which generate subsets of all serialisable schedules in polynomial time, such as the two-phase locking protocol [Esw76]. However, once timing constraints are added, even the problem of generating any feasible schedule becomes NP-complete [Mok83].

On the other hand, since (a) the number of data objects is bounded with transactions acting on fixed subsets of objects, and (b) the frequency with which transactions are invoked is bounded, it is possible to establish worst-case circumstances on which analysis can be based. The ability to use sufficiently recent values of data objects helps to eliminate interference between components of the system. From a scheduling point of view, this allows a larger number of possible schedules.

It should be noted that there is an important distinction between a database whose correctness includes temporal constraints, and one that does not. In the latter, conventional case, a consistent database remains consistent even if no further transactions are executed. However, a real-time database may become temporally inconsistent if certain transactions **are not** executed.

## 3. A SYSTEM MODEL

To analyse further the requirements of a hard real-time database it is necessary to postulate a general database model which provides applications with a suitable set of abstractions. We are not concerned with a database that has only a peripheral or archival importance. Rather we focus on the use of a hard real-time database as the main structuring component of the systems architecture.

### 3.1. The Object Structure of the Database

Consider an embedded (possibly distributed) application that has *N* distinct streams from sensors or HCI (human-computer interface) inputs; and *M* streams to actuators (effectors) or HCI outputs. Suppose also that there are *P* data objects which contain intermediate, or *derived*†, values used in the calculations involved in transforming the inputs to the outputs. The database holds *N+M+P primary* objects.

Clearly, the input values represent measurements of quantities in the environment. The derived values can also be considered to correspond to quantities in the environment, which are however inferred rather than measured directly. In this paper, we use the term *real-world value* to refer to the actual value of the quantity in the environment (some derived data objects may appear to have little direct relation to the real world, for instance *time to reach destination*; however, this is often just a matter of a suitable coordinate transformation). We say that a value *v corresponds* to its real-world value if it is equal to the real-world value within a static and predefined error.

Ultimately, system timing requirements are expressed between inputs and outputs; ie., a change in the value of an input must be result in an output being appropriately updated within some specified response time. The response time is a function of the system being controlled, and the precision with which that control must be exercised.

### 3.2. The Transaction Structure of the Database

All software components interact via the database; they read a subset of data objects and write to some other subset. A simple end-to-end timing requirement may thus involve the reading of a sensor to generate a value *S* in the database; the transmission of this value to a shadow *S'* on a different machine; the calculation of some derived value *V*; and the use of *V* to calculate an output value *A*. In this paper, we use the term *transaction* to identify these software components. This is a little different to the meaning used in conventional

†This is the term used by Song and Liu [Son90], and will also be used here.

databases.

In a conventional database, an event in the environment may require the execution of a transaction which updates a number of data objects. Typically, this involves *locking* the objects, making the necessary updates, and then *unlocking* the data objects, at which point the updates become visible to other transactions. The locking is necessary in order to prevent conflicting concurrent update. However, the completion of the transaction does not ordinarily imply any further activity; all changes that must follow on from the event will have been made by the transaction which it triggers.

In the model presented here, a transaction which is triggered by an event in the environment need not update all data objects which are dependant on that event. It may update a subset of those data objects, the update of the remaining data objects being the function of further related transactions. In effect, the event triggers the first of a chain of transactions, where the completion of one transaction triggers the execution of the next. This reduces the potential for contention between transactions over access to data objects, and hence the potential for transactions to become *blocked*.

However, as a result of the transaction model, the hard real-time database may, by comparison to a conventional database, be functionally inconsistent for some interval of time. This is analagous to the notions of *eventual* and *lagging* consistency; a summary of these is given by Sheth *et al* [She90]. The issue is addressed the section 4, where database consistency is defined for this model.

### 3.3. Distributed Databases

If the application is distributed then the database may not be centralised. In this case some primary objects will have corresponding *shadow* objects on remote processors. If the value of a primary object changes, then the change must be propogated to all of its shadow objects. We model the propogation as essentially *producer* driven, with an additional transaction on the same processor as the primary object being used to transmit the value. An alternative is a *consumer* driven model, in which an additional transaction is placed on the same processor as the shadow object; the choice is essentially arbitrary.

However, we choose to model the propogation by using transactions since the propogation requirements can be treated as database consistency constraints, and the timing requirements on these transactions can be derived in the same manner as for any other transaction. This is elaborated in the sections 4 and 5.

### 3.4. Database Input and Output

As implied above, inputs and outputs are modelled by database objects. An input object is not considered to be written by any normal transaction, but is updated directly from the sensor. Similarly, an output object is not read by any transaction, but is passed directly to an actuator. This corresponds directly to the model described by Song and Liu [Son90]. Alternatively, inputs could be modelled as transactions which do not read any data objects, and outputs as transactions which do not write any objects. However, for the

purpose of generating timing requirements, as described later, the database object input-output model is more convenient.

### 3.5. Perishable and Non-Perishable Data Objects

At this stage we introduce the notion of *perishable* and *non-perishable* data objects. All data objects in the database will fall into one of these two groups.

A data object is limited to taking values which are drawn from a finite set of possible values; in effect, the value is a state drawn from a finite set of possible states. Ideally, the data object should change states in a manner which follows changes in the corresponding real-world value. However, some real-world quantities will change value sufficiently rapidly that it is impossible or impracticable to keep the data object completely up to date; indeed, in some cases there may be no need for the data object to follow the real-world value exactly, even when it is possible. On the other hand, some will change sufficiently slowly that the data object can follow it, abliet with a slight delay due to the non-zero response time of the system.

We therefore define perishable data objects as follows:

**Defn (2)***: A real-world value can be mapped to a sequence of states which are its representation in the database. A data object is perishable if it cannot take all those states in time-ordered sequence; some states will inevitably be missed.*

The term *non-perishable* will refer to any data object which is not *perishable* by the above definition. These distinctions will be used later.

## 4. MAINTAINING DATABASE CONSISTENCY

We now consider the issue of specifying temporal consistency in a hard real-time database. We start by defining two properties, namely *absolute* and *relative* temporal consistency. These terms are derived from the *absolute* and *relative coherence* described by Song and Liu [Son90]. Here, however, the term *consistency* is used in preference to to *coherence*, as the properties under discussion are analogous to database *consistency*.

### 4.1. Absolute Temporal Consistency

In order that a transaction executes meaningfully, the transaction requires both that the values of the input data objects are accurate within some error bounds, and that the value stored in the data object is sufficiently recent. We quantify *sufficiently* by the *absolute* temporal consistency (abbreviated as *ATC*) of the data objects, as follows:

**Defn (3)***: If a data object X has an ATC* $a_x$ *then, at any given time* $t_{now}$*, the value* $v_x$ *stored in X must correspond to the real-world interpretation of X at some time in the interval* $[t_{now}\text{-}a_x, t_{now}]$*.*

We note at this point that, if a data object *V* is updated by a single transaction *T*, then the responsibility for maintaining the ATC of *V* falls solely on transaction *T*. Also, we will

use the term *ATC expiry* of a data object to mean that the ATC requirement for that data object no longer holds.

### 4.2. Relative Temporal Consistency

If a transaction reads a set of data objects, and uses the values to calculate some result, then it may be necessary that the data objects contain values which correspond to the real-world at similar times. For instance, a combination of satellite, sonar and inertial position information may only be meaningful if they have been sampled sufficiently close together. This type of timewise consistency will be referred to as *relative* temporal consistency, and abbreviated as *RTC*. Formally:

> **Defn (4)***: If a set of data objects* $\mathbf{S}=\{V_i\}$ *has an RTC* $r_s$*, when the values* $v_i$ *of the* $V_i$ *must correspond to real-world interpretations of the* $V_i$ *at times which fall within an interval of length* $r_s$*.*

It is likely that different input data objects to some transaction will themselves be updated by different transactions. In order to maintain the RTC of those objects, it is necessary to ensure the updating transactions are suitably invoked. We will use the term *RTC set* to refer to a set of data objects over which there is an RTC requirement, and *RTC value set* to a set of values which satisfy the RTC requirement.

It should be noted that an RTC requirement on its own is not particularly useful. If the data objects for which an RTC requirement exists do not have any ATC requirements, then consistency could be maintained by leaving the data objects unchanged once a consistent set of values had been written. Where an RTC are particularly useful is when the RTC requirement is significantly shorter then any ATC requirements which the group possesses (for example, sample a set of position sensors within 2ms (the RTC), but only every 2s (the ATC)).

### 4.3. Functional Consistency

If the update of some data object *X* requires a subsequent update to another data object *Y*, in the manner described in section 3, then there is a functional dependency of *Y* on *X*. This can be expressed either as $Y = \mathbf{f}(X)$, or as $\mathbf{P}(Y,X)$, where **P** is a predicate which must hold for the database to be consistent. The latter form is generally applied to conventional databases.

In our model, the function **f** may be executed some time after *X* is updated, and while the old value of *Y* is still visible. Hence, until **f** completes, the predicate **P** may not hold. We will therefore define functional consistency in the following manner:

> **Defn (5)***: Suppose than some predicate* **P** *applies to data objects X and Y where the data objects have ATC requirements* $a_y$ *and* $a_x$*, and Y has value* $v_y$*. Then X and Y are functionally consistent at time* $t_{now}$ *if* $\mathbf{P}(v_y,v_x)$ *holds, where* $v_x$ *is some value held by X in the time interval* $[t_{now}-(a_y-a_x),t_{now}]$*.*

This is justified as follows. We assume that, in order for $v_y$ to satisfy the ATC for *Y* at time $t_{now}$, it must have been calculated from a value $v_x$ which itself corresponded to the real-world within the time interval $[t_{now}-a_y,t_{now}]$†. In the worst case, the value $v_x$ could have been read from *X* at time $t_w=t_{now}-(a_y-a_x)$ and, at the time it was read, the value $v_x$ could have corresponded to the real-world value of $X_i$ at a time $a_x$ earlier. Hence, the value $v_y$ could in the worst case be based on a value corresponding to the time $t_w-a_x$. This is equal to $t_{now}-a_y$, which is the ATC requirement of Y.

This definition can trivially be generalised to the case where Y is dependant on a set of data objects $\mathbf{S}=\{X_i\}$. In this case, there may also be an RTC requirement on the values of the $X_i$.

### 4.4. Some Examples

In this section we present some simple examples. We write $a_x$ to indicate the ATC requirement of data object *X*, and $r_{p,q,...}$ for an RTC requirements over the data objects P,Q,... All examples are based on an aircraft control system.

**(a)** Suppose that object B is an input from a barometric altimeter, and H is a derived data object containing altitude, corrected for barometric pressure at sea level. B might, for practical purposes, contain the instantaneous barometric altitude, hence $a_b=0$; if the value in H must never be more than two seconds out of date, then $a_h=2$. It is likely that H is perishable for any reasonable resolution and update rate, and hence the transaction which implements H=**f**(B) must be executed periodically.

**(b)** Suppose that fuel is contained in two tanks, whose levels are available via derived data objects T1 and T2. Fuel is pumped between the tanks for trimming purposes. Also, the total fuel remaining must be periodically written to an output data object R for display to the pilot. It may be adequate to update the display every 10 seconds, hence $a_r=10$, and we might choose $a_{t1}=a_{t2}=5$. However, if a significant amount of fuel can be pumped from one tank to another within a 5 second period, we also require that the values in T1 and T2 correspond to similar times; hence, we might have $r_{t1,t2}=1$.

**(c)** Suppose that input data object D indicates whether the cargo door is open, and that output data object W is a warning display in the cockpit, indicating that the door is open. W is a non-perishable data object, since the cargo door will not be opened and closed with any great frequency. We might have $a_d=0$ and $a_w=1$, so that the cockpit display never lags the door state by more than one second. The transaction which implements W=**f**(D) might be executed periodically; however, sporadic execution with a suitable deadline would entail lower processor loading.

---

†If the transaction could perform perfect forward interpolation, then it could use earlier values. However, in this paper we assume that forward interpolation is used only to better the ATC requirement, and not to meet it.

### 4.5. Transaction Scheduling and the Single-Writer Restriction

In order to maintain the consistency requirements described above, two conditions must be met:

- Each individual transaction must itself satisfy its corresponding predicate **P**. This is exactly the functional consistency of a conventional database.
- The transactions must be scheduled in a timely manner, such that definitions (3), (4) and (5) are maintained.

Note that, in a conventional database, the ATC requirement is effectively zero. If this is applied to definition (5), then the predicate **P** must hold over data values in the interval $[t_{now},t_{now}]$, that is, the current values. This is the conventional definition of functional consistency, and commonly leads to *serialisable* [Kun79] schedules. Much database theory is directed to finding schedules which, in the interests of speed, overlap transactions, but which are equivalant to some serial schedule.

The primary obstacle to schedule generation is the need to provide mutual exclusion between transactions which access common data objects. This problem can however be alleviated, or even removed, if the single-writer restriction can be applied:

**Defn (6)***: The data base is single-writer if each data object, other than input data objects, is updated by a single transaction.*

If this restriction is imposed, then mutual exclusion is only needed for the period during which a transaction writes to a data object. This is the case even for read-recompute-write access, since the data object cannot be changed by any other transaction during the computation. However, mutual exclusion schemes, such as locking[Esw76] can be avoided using the algorithms described by Simpson [Sim90a, Sim90b]. Briefly, these allow shared access to data objects in constant time, and work on both uniprocessor and shared-memory multiprocessor systems. The most general of the algorithms has the additional useful property that data cannot be lost as a result of the failure of the writing transaction. Note that it is possible to implement multiple-writer access without locking, as described by Herlihy [Her91]. However, Herlihy's algorithm is much more complicated and computationally expensive, although still bounded.

The generation of transaction timing requirements from the ATC and RTC requirements is described in the next section, based on the single-writer restriction.

## 5. TRANSACTION TIMING REQUIREMENTS

In this section, we derive some transaction timing requirements from the database timing requirements.We will use the notation V(X) to denote the value of data object X. Also, $V_t(X)$ denotes the value at time t, and $\mathbf{V}\ t_0\ t_1\ (X)$ to denote the set of values taken by data object X in the time interval $[t_0,t_1]$.

### 5.1. Periodic Transactions

Suppose that a periodic transaction $\tau_p$, with period $T_p$ and deadline $D_p$, reads a data object X, performs some calculations, and writes data object Y, such that Y=**f**(X), with the corresponding predicate **P**. Also, assume that X and Y have ATC requirements $a_x$ and $a_y$. By the definition of functional consistency (5), we require that, at any time $t_{now}$:

$$\exists v \in \mathbf{V}^{t_{now}}_{t_{now}-(a_y-a_x)}(X) : \mathbf{P}(v, V_{t_{now}}(Y))$$

That is, at any given time $t_{now}$, the value of data object Y must be based on a value stored in data object X in the interval $[t_{now}-(a_y-a_x), t_{now}]$.

Now, the maximum time interval between a change in the value stored in data object X, and the subsequent update of data object Y based on the new value, occurs over two successive executions of $\tau_p$, such that $\tau_p$ reads X immediately on release of the first execution, and writes Y at the deadline of the second execution; call these times $t_0$ and $t_0+(T_p+D_p)$. In this case, a change in X immediately after $t_0$ will not be reflected in Y until $t_0+(T_p+D_p)$. Clearly, just before $t_0+(T_p+D_p)$, Y=**f**($V_{t_0}$(X)), so in order that functional consistency is maintained over this interval, we require:

$$T_p + D_p < a_y - a_x$$

This can be trivially generalised to cover the case of transactions which read and write multiple data objects. Suppose that the transaction reads data objects **X**={$X_1$,...,$X_n$} and writes to data objects **Y**={$Y_1$,...,$Y_m$}. Then, in order that functional consistency is always maintained:

$$\forall i=1,...,n, j=1,...,m : T_p + D_p < a_{y_j} - a_{x_i}$$

and hence:

$$T_p + D_p < \min_{i,j}(a_{y_j} - a_{x_i}) \qquad \text{(E1)}$$

### 5.2. Sporadic Transactions

A similar argument can be applied where Y=**f**(X) is maintained by a sporadic transaction $\tau_s$ with deadline $D_s$. If we assume that the transaction is released as soon as X is updated, then the longest interval before the subsequent update of Y is $D_s$, and hence:

$$D_p < a_y - a_x$$

This can be generalised to multiple data objects in exactly the same manner:

$$D_p < \min_{i,j}(a_{y_j} - a_{x_i}) \qquad \text{(E2)}$$

However, in order that there is any possibility of providing guarenteed schedulability, it is necessary to assign a mimimum inter-arrival time to the transaction $\tau_s$. This corresponds to a minumum interval between changes to data objects.

### 5.3. RTC Requirements

The presence of an RTC requirement introduces transaction timing requirements over and above those derived in the previous two sections. Specifically, whenever the data objects in an RTC set are read by some transaction, a valid RTC value set must be available.

Consider an RTC group **S**=($X_1$,...,$X_n$), where the data objects have ATC requirements $a_{x_i}$, the RTC requirement is $r_s$, and where the data objects are written by transactions $\tau_1$,...,$\tau_n$. For simplicity, we assume that the transactions are periodic (with periods and deadlines $T_i$ and $D_i$), and that they have a critical deadline at some time. Finally, assume that if $\tau_i$ writes to $X_i$ at time $t_i$, then the value corresponds to the real world at a time not earlier than $t_i$-$\delta_i$, where $\delta_i$ is a function only of $\tau_i$.

The latest time at which an execution of a transaction may write to data object X is at its deadline. Suppose that transaction $\tau_i$ has a deadline at t=$\varepsilon_i$. Then, the earliest time at which the write could occur is t=$\varepsilon_i$-$D_i$, and the earliest real-world time for the data value is t=$\varepsilon_i$-$D_i$-$\delta_i$. Now, the RTC requirement will be met if the total span of these intervals does not exceed $r_s$. That is:

$$\max_i(\varepsilon_i)-\min_i(\varepsilon_i-D_i-\delta_i)\le r_s$$

The window is smallest if the $\varepsilon_i$ are all equal, and we can, without loss of generality set $\varepsilon_i$=0 for all i. Hence:

$$\max_i(D_i-\delta_i)\le r_s \qquad \text{(E3)}$$

If the above relationship holds, then a set of data values for which the RTC requirement holds will be available immediately after time t=0. In general, a new set will be be produced at times t=n.$LCM_i(T_i)$, where n is an integer, and $LCM_i$ is the Least Common Multiple function†. This behaviour is analagous to a single periodic process with period $LCM_i(T_i)$, and a deadline equal to $\max_i(D_i)$. Hence, the last result above for periodic transactions can be used to ensure the ATC requirements of the data objects in **S**:

$$\underset{i}{LCM}(T_i)+\max_i(D_i) < \min_i(a_i-\max_j(a_{i,j})) \qquad \text{(E4)}$$

where the $a_{i,j}$ are the ATC requirements of the data objects read by the transactions $\tau_i$.

If the periods $T_i$ are not all the same (and hence equal to the LCM interval), then some transactions will generate values which are not members of any RTC value set. If this destroys the previous value, then the transactions which read the RTC group must be synchronised with the writing transactions, such that they read the RTC group after a valid RTC value set appears, but before any individual data object is changed. This is only guaranteed in the interval for t=n.$LCM_i(T_i)$ to t=n.$LCM_i(T_i)$+min($T_i$). This requirement

---

†Intermediate sets may be produced between LCM intervals, however this is relatively difficult to analyse and is ignored here.

places undue restrictions on the transactions which read the RTC set. Further, it means that a sporadic transaction cannot read the RTC group, since it may not execute in suitable synchronisation with the writing transactions.

In order to circumvent these problems, we arrange that data objects which are members of RTC sets hold multiple data values. Each data object $X_i$ holds the $LCM_j(T_j)/T_i+1$ most recent values written to the data object. Under these circumstances, it is always possible to select a set of value which comprise an RTC value set. Hence, transactions which read the RTC set need not be synchronised.

## 6. AN ILLUSTRATIVE EXAMPLE

A simple example is now presented in order to illustrate the ideas described above.

### 6.1. Example System

The system is a fuel and stability system for a boat. Fuel is stored in two tanks, port and starboard, and is used to provide ballast as well as feeding the engines. Fuel is pumped between the two tanks using a ballast pump. The tank levels are monitored via level gauges, and the total fuel remaining is displayed.

The software system is shown in the diagram below, where circles represent data objects and squares represent tasks. An arrow from a data object to a task indicates that the task uses that data object. Note that those parts of the system concerned with controlling fuel flow to the engines have not been included.

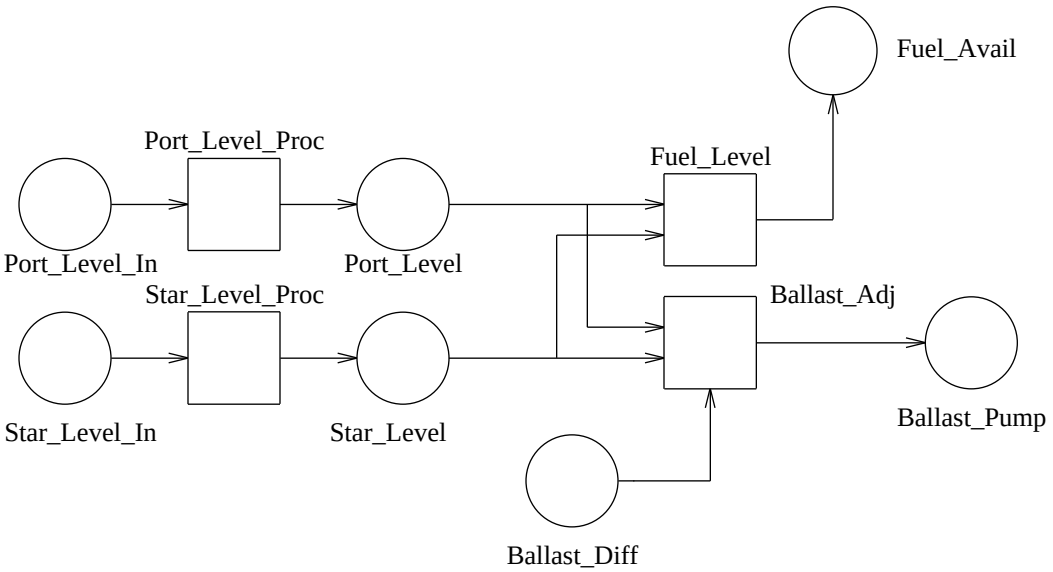


Diagram (1): An Example System

| **Database Object** | **Abbrev.** | **Usage** |
|---|---|---|
| Port_Level_In, Star_Level_In | pli,sli | Port (starboard) level input |
| Port_Level, Star_Level | pl,sl | Processed port (starboard) level |
| Fuel_Avail | fa | Calculated total fuel remaining |
| Ballast_Diff | bd | Required ballast difference |
| Ballast_Pump | bp | Ballast pump control output |
| **Process** | **Abbrev.** | **Usage** |
| Port_Level_Proc, Star_Level_Proc | plp,slp | Process port (starboard) level input |
| Fuel_Level | fl | Calculates total fuel remaining |
| Ballast_Adj | ba | Controls ballast pump and warning |

Table (1): An Example System

We assume (a) that the ballast pump can pump fuel in either direction at a maximum rate of 10gals/sec; (b) that the fuel available value must be accurate to within ±20gals (at the time to which the displayed value corresponds); (c) that the fuel available value must be not more than 30secs old; (d) that the ballast pump control must be set at least every 10sec; (e) that the ATCs of the input data objects are all zero; and (f) that the age associated with an object when written is the maximum age of the input objects to the writing task (ie., none of the tasks contain any predictive functionality)

(1) $a_{fa} = 30$
(2) $a_{bp} = 10$
(3) $a_{pli} = 0$
(4) $a_{sli} = 0$
(5) $a_{bd} = 0$

### 6.2. Task Timing Constraints

We now consider the requirements placed on each of the tasks. Also, for brevity, define $\mathbf{R}_{task}=T_{task}+D_{task}$.

**Fuel_Level**

The calculated fuel level value may be up to 30secs out of date; however, it must be correct within 20gals. Given that the ballast pump can move 10gals/sec, the tank levels must be sampled within not more than 2 seconds of each other (between such samples, the level in the tank sampled second could change by 20gals; this ignores fuel pumped to the engines). Hence, from the ATC requirements and (E1):

(6) $\mathbf{R}_{fl} \leq \min(a_{fa}-a_{pl}, a_{fa}-a_{sl})$
(7) $r_{pl,sl} \leq 2$

From the RTC requirement, and (E3) and (E4):

(8) $D_{plp}-\delta_{plp} \leq 2$
(9) $D_{slp}-\delta_{slp} \leq 2$

(10) $LCM(T_{plp},T_{slp})+\max(D_{plp},D_{slp}) < \min(a_{pl}-a_{pli},a_{sl}-a_{sli})$

**Ballast_Adj**

The ballast adjustment constraint is derived simply from the requirements on the Ballast_Pump output. Depending on the precision required of the ballasting operation, there might be a RTC requirement, however this is ignored here for simplicity. By (E1):

(11) $\mathbf{R}_{ba} \leq a_{bp}-a_{pl}$
(12) $\mathbf{R}_{ba} \leq a_{bp}-a_{sl}$
(13) $\mathbf{R}_{ba} \leq a_{bp}-a_{db}$

**Port_Level_Proc, Star_Level_proc**

In order to maintain the ATC of the Port_Level and Star_Level objects, the relationships below must hold.

(14) $\mathbf{R}_{plp} \leq a_{pl}-a_{pli}$
(15) $\mathbf{R}_{slp} \leq a_{sl}-a_{sli}$

Further timing requirements on these processes are implied by the earlier RTC requirement.

### 6.3. Task Timing Characteristics

Any set of timing characteristics for the tasks, which satisfy the requirements given in the previous section, and for which a feasible schedule exists, can be used to successfully implement the system. We will that all tasks will be periodic.

If we set $a_{pl}=a_{sl}=5$, then the ATC requirements imply the following:

(16), from (6) $\mathbf{R}_{fl} \leq \min(30 - 5, 30 - 5) = 25$
(17), from (11,12,13) $\mathbf{R}_{ba} \leq \min(10 - 5, 10 - 5, 10 - 0) = 5$
(18), from (14) $\mathbf{R}_{plp} \leq 5 - 0 = 5$
(19), from (15) $\mathbf{R}_{slp} \leq 5 - 0 = 5$

For the RTC requirements, assume that $\delta_{plp}=D_{plp}+a_{pli}$, and similarly for $\delta_{slp}$. This is justified by assumption (f). Hence:

(20), from (8) $D_{plp}-(D_{plp}+a_{pli}) \leq 2$
(21), from (9) $D_{slp}-(D_{slp}+a_{sli}) \leq 2$
(22), from (10) $LCM(T_{plp},T_{slp})+\max(D_{plp},D_{slp}) < \min(5 - 0, 5 - 0) = 5$

If we arbitrarily set $T_i=D_i$, then the following results satisfy (16-22).

(23), from (16) $T_{fl} = 12.5$
(24), from (17) $T_{ba} = 2.5$
(25), from (18) $T_{plp} = 2.5$
(26), from (19) $T_{slp} = 2.5$

### 6.4. Comments

For a given set of task timing constraints, there may be an arbitrary number of sets of task timing requirements; of these, not all may schedulable. Also, the choice of ATC requirements for derived data is not well defined; some choices may preclude the existance of a schedulable task set. To make the process usable, particularly for realistically sized systems, a practicable mechanism must be found. One possibility is *simulated annealing*; this is described by Tindell [Tin92] in connection with allocating hard real-time tasks to a processors in a distributed system.

## 7. CONCLUSIONS

In this paper a database model is presented that is applicable to at least some hard real-time systems. The essential properties of the model are:

- some objects may have ATC values defined
- some sets of objects may have RTC values defined
- objects are either perishable or non-perishable

If a conventional database is used in a real-time system then two inter-related (NP-complete) problems need to be addressed: concurrency control over access to the database, and the scheduling of systems of tasks to meet timing constraints. The motivation behind the model presented here is that value consistency (which requires concurrency control) is transformed into temporal consistency so that the complete problem becomes one of scheduling only. To this end we have also considered restricting the database so that all objects are written to by exactly one task. This does not necessarily restrict functionality but may require that more objects and tasks are introduced. Again the motivation is to constrain the problem to one of scheduling. The database does not require locking primitives or abort facilities.

Given these properties it is perhaps worthwhile to question whether the resulting formulation is indeed a database. One means of addressing this issue is to consider the four properties often associated with databases, namely: atomicity, consistency, isolation (or serialisability) and durability. The single writer model does provide atomicity; isolation is maintained by the scheduling of accesses to the database; and although durability has not been addressed in detail it is achievable (indeed, by not making use of locking the problem is simplified). It is in the issue of consistency that the proposed model is unconventional. The model supports two forms of temporal consistency; value consistency is only achieved when it can be derived from the temporal consistency relationships. Thus the real-time model uses temporal consistency, whereas the conventional model uses value consistency. The parallels are such that the term database is still appropriate.

One of the motivations for this formulation is as a system modelling technique. Within the correct application domain it is possible to capture all timing requirements as temporal consistencies on a system-wide database. From these requirements the necessary tasks (and additional database objects) can be derived systematically, together with all their temporal attributes (period, deadline, etc). The resulting task set is of a form that is

amenable to schedulability analysis. Hence the absolute and relative temporal consistencies can be guaranteed.

## REFERENCES


Agr89. D. Agrawal and S. Sengupta, ‘‘Modular Synchronisation in Multiversion Databases: Version Control and Concurrency Control’’, *1989 ACM-SIGMOD Intl. Conf.* (1989).

Cel88. W. Cellary, E. Gelenbe and T. Morzy, *Concurrency Control in Distributed Database Systems*, 1988.

Esw76. K. P. Eswaran, J. N. Gray and R. A. Lorie, ‘‘The Notions of Consistency and Predicate Locks in a Database System’’, *Communications of the ACM* **19**(11), pp. 624- (November 1976).

Her91. M. Herlihy, ‘‘Wait-Free Synchronisation’’, *ACM Trans. on Prog. Lang. and Systems* **Vol. 11, No. 1**, pp. 124-149 (January 1991).

Kun79. H. T. Kung and C. H. Papadimitriou, ‘‘An Optimal Theory of Concurrency Control for Databases’’, *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pp. 111-126 (1979).

Kun81. H. T. Kung and J. T. Robinson, ‘‘On Optimistic Methods of Concurrency Control’’, *ACM Trans. on Database Systens*, pp. 213-226 (June 1981).

Lyn83. N. A. Lynch, ‘‘Multi-Level Atomicity - A New Correctness Criterion for Database Concurrency Control’’, *ACM Transactions on Data Base Systems Vol 8 No 4*, pp. 484-502 (Dec 1983).

Mok83. A. K. Mok, ‘‘Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment’’, *PhD. Theses, MIT/LCS/TR-297* (1983).

Pap84. C. H. Papadimitriou and P. C. Kanellakis, ‘‘On Concurrency Control by Multiple Versions’’, *ACM Trans. on Database Systems* (March 1984).

Sha88a. L. Sha, J. P. Lehoczky and E. D. Jensen, ‘‘Modular Concurrency Control and Failure Recovery’’, *IEEE Transactions on Computers*, pp. 146-159 (Feb 1988).

Sha88b. L. Sha, R. Rajkumar and J. P. Lehoczky, ‘‘Concurrency Control for Distributed Real-Time Databases’’, *SIGMOD Record*, pp. 82-96 (Vol, 17, No. 1, March 1988).

She90. A. Sheth and M. Rusinkiewicz, ‘‘Management of Interdependant Data: Specifying Dependency and Consistency Requirements’’, *Proceedings of the Workshop on the Management of Replicated Data*, pp. 133-136 (Nov 1990).

Sim90a. H. Simpson, ‘‘Four-Slot Fully Asynchronous Communication Mechanism’’, *IEE Proceedings*, pp. 17-30 (Vol. 137, Pt. E., No. 1, Jan 1990).

Sim90b. H. Simpson, ‘‘Correctness Analysis for a Class of Asynchronous Communication Mechanisms’’, *BAe Report* (1990).

Son90. X. Song and J. W. S. Liu, ‘‘Performance of Multiversion Concurrency Control Algorithms in Maintaining Temporal Consistency’’, *Dept. of Computer Science,*

*Uni. of Illinois at Urbana-Champaign* (Feb. 1990).

Tin92. K. Tindell, A. Burns and A. Wellings, ‘‘Allocating Real-Time Tasks (An NP-Hard Problem made Easy)’’, to appear in *Real Time Systems* (1992).