Where do Time Constraints Come From and Where do They Go? *

Krithi Ramamritham
Department of Computer Science
University of Massachusetts
Amherst Ma. 01003
email: krithi@cs.umass.edu

Invited Paper – To appear in
International Journal of Database Management

Abstract

While a lot of work has been done in real-time systems, in real-time database systems, and more recently, in real-time active databases on the topic of time constrained processing of tasks and transactions, very little work exists that deals with the origin of the time constraints associated with the data, the events, and the actions. In this paper we identify the sources and semantics of time constraints and show that it is important to minimize the number of "mandated" timing requirements and also weaken the implications of timing constraint violations. The Event-Condition-Action rules of active real-time databases provide a useful framework to specify the timing properties of interest as well as the actions to be taken when the properties are violated. That is, an active real-time database can be made to store the data pertaining to the controlled system as well as the meta-data about the controlling system.

*This work was supported, in part, by NSF under grant IRI-9208920.

Contents

1 Introduction

What separates real-time systems from non real-time systems is the presence of data that becomes invalid with the passage of time, the presence of events that must occur in a timely fashion, and the presence of actions whose timely completion is as important as the results produced. Many of the timeliness requirements, however, are artifacts of the way a system is designed and the (usually ad-hoc) manner in which the time constraints are assigned. Of course, some time constraints are imposed by the external (physical) environment and these must be ensured by the system. However, quite often, since the origins of the time constraints are not known to a real-time system, it attempts to satisfy them all, leading to an overconstrained or overdesigned system.

Given specific time constraints, a lot of work has been done on the topic of time constrained processing of tasks and transactions [5]. But very little work exists that deals with the origins of the time constraints associated with the data, the events, and the actions. Knowing their origins it is possible to determine the semantics of the time constraints and also the best ways to satisfy timing requirements. This paper examines these issues and shows that the timing properties of interest and the actions to be taken when the properties are violated can be specified using the Event-Condition-Action (ECA) rules of active databases. Because of this, a real-time active database system (RTADB) can serve as a repository of not only the data about the environment of interest but also of the meta-data that can help in the adaptive handling of time constraint violations – of the data, events and the actions.

Throughout this paper we will use the term *action* to refer to computations with time constraints. Actions could be complex, containing subactions. Actions encompass real-time tasks and real-time transactions.

The rest of the paper is structured as follows. In Section 2 we discuss two real-time application examples that can benefit from RTADB technology. These will be used to substantiate the points made throughout the paper. Factors that determine the time constraints are discussed in Section 3. Section 4 provides a detailed discussion of the origins of the timing properties of actions, data, and events. Implications of timing constraint violations is the topic of Section 5. Section 6 briefly discusses how time constraints ought to be derived methodically. Section 7 summarizes the paper.

2 Motivating Examples

In this section we introduce two examples of real-time applications that can benefit from real-time active database technology. Aspects of these examples will be used throughout the paper to illustrate our ideas. (Further detailed examples can be found in [6]).

Consider recognizing and directing objects moving along a set of conveyer belts on a factory floor. An object's features are captured by a camera to determine its type and to recognize whether it has any abnormalities. Depending on the observed features, the object is directed to the appropriate workcell. In addition, the system updates its database with information about the object.

The following aspects of this example are noteworthy. First of all, features of an object must be collected while the object is still in front of the camera. Then the object must be recognized by matching the features against models for different objects stored in a database. This matching has to be completed in time so that the command to direct the object to the appropriate destination can be given before the object reaches the point where it must be directed onto a different conveyer belt that will carry it to its next workcell. The database update must also be completed in time so that the system's attention can move to the next object to be recognized. If, for any reason, a time-constrained action is not completed within the time limits, alternatives may be possible. In this example, if feature extraction is not completed in time, the object could be discarded for now to be brought back in front of the camera at a later point in time.

As another example, consider the following air traffic control scenario at an airport. The air traffic control system makes decisions concerning incoming aircrafts' flight path, the order in which they should land, and separation between landings based on a multitude of parameters including current wind velocity, position of the aircrafts, their speed, fuel position, and altitude and the type of aircraft. If an air traffic controller cannot accommodate all the incoming aircrafts, he or she can ask aircraft(s) to assume a holding pattern until the situation improves.

Applications such as these introduce the need for *real-time active database systems.* These applications involve gathering data from the environment, processing gathered information in the context of information acquired in the past, and providing *timely* response. Another aspect of these examples is that they involve processing of both temporal data, which loses its validity after a certain interval, as well as archival data. Finally, the applications involve triggering of specific actions in specific situations.

Real–time systems consist of a *a controlling system* and a *controlled system.* For example, in an automated factory, the controlled system is the factory floor with its robots, assembling stations, and the assembled parts; the controlling system is the computer and human interfaces that manage and coordinate the activities on the factory floor. The controlled system can be viewed as the *environment* with which the computer interacts.

3 Factors that determine the Time Constraints

In this section, we examine the factors that determine these time constraints. The *externally-imposed* temporal properties depend on many factors including:

- Characteristics of the physical systems being controlled:
  - e.g., the speed of the aircraft (to calculate time at which it will touch ground); the speed of the conveyer belt (to calculate available time to decide the path of the object).
- Stability characteristics of a system as governed by its control laws:

  - e.g., servo control loops of robot hands, fly-by-wire avionics.

- Quality of service requirements (service delays tolerable by the controlled system or humans):
  - e.g., sampling rates for audio and video; responsiveness of air traffic controller to an aircraft pilot's request for current weather conditions at destination.

- Human (re)action times, human sensory perception:
  - e.g., time between warning (e.g. low fuel level) and action based on the warning (e.g. immediate start of landing following emergency procedures).

These factors determine the time constraints that are inherited by the controlling system from the external environment. Understanding these factors is important because some of these are more important to satisfy (e.g., physical environment related factors) than others (e.g., quality-of-service factors related to human interactions). Using this information, importance levels can be set for actions.

In some sense, all time constraints, be they externally imposed or resulting from design decisions (see below), are artifacts. For instance, in relation to the items listed above, length of a runway or speed of an aircraft are determined by cost and technology considerations; quality of service requirements, for instance, in telephone networks, are quite often decided by regulatory authorities; response times guaranteed by service providers are determined by cost and competitiveness factors. Unfortunately, many of these decisions are not under the control of a computer system designer. Hence, the system designer's decisions must be made within the boundaries laid out by others. This is what we referred to above as being externally-imposed. What is important to note is that the number of factors "given" to the designer must be kept to a minimum leaving more options to satisfy the requirements.

Subsequent decisions of the designer introduce additional constraints. The type of computing platform used (e.g., centralized vs. distributed), the type of software design methodology used (e.g., data-centric vs. action-centric), the (pre-existing) subsystems used in composing the system, the nature of the actions (e.g., monolithic action vs. graph-structured action), etc., further curtail the decisions.

At some level in the design of a real-time application, decisions concerning whether an action is periodic, sporadic, or aperiodic have to be made, the values for the periods, deadlines, and offsets within periods must be chosen, and importance or criticality values must be assigned.

Thus, the decisions made at one level affect those at the other level(s). What has to be recognized is that while no decision at any level is likely to be sacrosanct, i.e., beyond modification, cost and time considerations will prevail in any such overhaul of prior decisions. Decisions at one level may percolate to many subsystems below and a previous decision may not be changeable just to accommodate the needs of a single subsystem.

While it will be beneficial to determine all related time constraints in an optimal fashion, for non-trivial systems the problem is likely to be intractable. This is the reason a divide-and-conquer approach is adopted wherein a system is designed from subsystems each of which must satisfy a set of given time constraints. Such time constraints reflect the specific design strategy and the subsystems chosen as much as the externally imposed timing requirements.

4 Origins of Timing Properties of Events, Data, and Actions

In this section we examine the origins of the time constraints from the perspective of the events, the data, and the actions.

4.1 Time Constraints of Events

In real-time applications, actions are triggered by the occurrence of events, that is, associated with each action is an event occurrence. Because of this, certain types of time constraints on the actions are inherited from the types of events that trigger these actions. Hence it is important to examine events with respect to their timing-related properties. In essence, there are three types of such time constraints [2].

1. *Maximum*: delay between two events.

   Example: Once an object enters the view of the camera, object recognition must be completed within t_1 seconds.

2. *Minimum*: delay between two events.

   Example: No two flight landings must occur within t_2 seconds.

3. *Durational*: length of an event.

   Example: The aircraft must experience no turbulence for at least t_3 seconds before the "seat-belts sign" can be switched off once again.

With respect to the last example, note that an event with a duration can be viewed as a complex event where the begin and end of the complex event are themselves marked by the occurrence of a (simple) event.

Events could be inputs or stimulus (S) events, output or response (R) events, and internal (I) or invisible (outside the system) events.

At some level within the system, even internal events can be categorized as S or R events with respect to subsystems of the system. For example, if the planning subsystem of an automated factory takes the current and desired position of a robot and creates a plan for the robot to reach the desired position, the arrival of the request to the planner is an S event as far as the planner is concerned; when the planner produces the plan and sends it to the rest of the factory control system, an R event occurs. Both of these, however, could be events that occur as a result of a larger request from the environment (e.g. the operator) to the factory control system, for example to get the floor to produce a new product.

Minimum, maximum, and durational constraints can occur between stimulus and response events (that is, all four combinations are possible) even though, the S-R combination is the only one usually considered in the scheduling literature.

The maximum and minimum type of time constraints associated with recurring (stimulus) events are also referred to as *rate-based* constraints since they relate to the maximum and minimum rate of arrival of recurring stimuli.

The occurrence of an event is handled by an action associated with that event. Time constraints on these events determine the constraints on these actions: rate-based constraints translate into periodicity requirements for the corresponding actions. Time constraints relating a stimulus and its response corresponds to deadline constraints. Specifications of minimal separation between response to a stimulus and the next stimulus dictate the behavior of the sporadic activity that deals with that stimulus. Thus, the constraints on events translate to constraints on the actions that deal with these events.

The ECA rules are very convenient to enforce these constraints and also to trigger the actions. It is very easy, for example, to see how one can translate the example given for the durational constraint into an ECA rule.

4.2 Time Constraints of Data

The controlling system interacts with its environment based on the data available about the environment, say from various sensors, e.g. position, velocity, and altitude sensors, and cameras. It is imperative that the state of the environment, as perceived by the controlling system, be consistent with the actual state of the environment. Otherwise, the decisions of the controlling system may be wrong and their effects disastrous. Hence, timely monitoring of the environment as well as timely processing of the sensed information is necessary.

The sensed data is processed further to derive new data. For example, data such as aircraft position, heading, and velocity, are used to derive the time at which it would touch ground if allowed to land. This derivation depends on the nature of the aircraft and its payload and so some of the needed information may have to be fetched from archival storage. The controlling system's responses are based on the derived data, where the derivation may involve multiple steps. In our air traffic control example, the derived information is used to send commands to the aircraft to land or to assume a holding pattern. In general, the history of (interactions with) the environment are also logged in archival storage.

In order for the control system to take the right decisions and provide the proper response to the environment, its view of the environment must be a close reflection of the actual state of the environment. This need to maintain consistency between the actual state of the environment and the state as reflected by the contents of the database leads to the notion of *temporal consistency.* Temporal consistency has two components [8, 1]:

- *Absolute consistency* – between the state of the environment and its reflection in the database; this arises from the need to keep the controlling system's view of the state of the environment consistent with the actual state of the environment. Let us denote a data item in the real-time database by

  $$d : (value, avi, timestamp)$$

  where d_{value} denotes the current state of d, and $d_{timestamp}$ denotes the time when the observation relating to d was made. d_{avi} denotes d's *absolute validity interval*, i.e., the length of the time interval following $d_{timestamp}$ during which d is considered to have absolute validity.

- *Relative consistency* – among the data used to derive other data; this arises from the need to produce the data used to derive other data close to each other. A set of data items used to derive a new data item form a *relative consistency set.* Each such set R is associated with a *relative validity interval* denoted by R_{rvi}.

Assume that $d \in R$. d has a correct state iff

1. d_{value} is logically consistent – satisfies all integrity constraints specified on the data.
2. d is temporally consistent:
   - Absolute consistency: $(current\_time - d_{timestamp}) \leq d_{avi}$.

- Relative consistency: $\forall d' \in R, \mid d_{timestamp} - d'_{timestamp} \mid \leq R_{rvi}$.

Ideally, we would like the data in the database to be temporally correct at all times. In practice, however, it is crucial that data be temporally correct when it is used or consumed by some action. Viewed from this need, absolute consistency can be seen as a *freshness* constraint and relative consistency can be seen as a *correlation* constraint [3]. That is, when a data item is *used*, it must be fresh (as specified by the absolute consistency requirements) and must be temporally correlated (as specified by the relative consistency requirements) with other data that it is used with.

Keeping the data in the database temporally correct requires timely execution of the actions that record the arrival of stimulus or input and the timely derivation of data. If the data is temporally invalid corrective actions must be taken.

> **Example:** Position, velocity, and heading values pertaining to aircrafts, as reflected in the air traffic controller's database must be fresh as well as temporally correlated. If temporal validity is lost, new values must be obtained to restore validity.

There are obvious interrelationships between the way one type of data is used to derive another, the composition of the relative consistency sets, and the manner in which timestamps are set for derived data. Methodical approaches must be developed to address this problem such that the system is not overconstrained, i.e., temporal consistency requirements must not be stricter than necessary. This is important since temporal consistency requirements translate into time constraints on actions (see Section 6), and the more restrictive the temporal consistency requirements, the tighter the time constraints on actions, and the harder it is to satisfy them.

Before we conclude our discussion of data temporal properties, several points must be noted.

The *avi* and *rvi* of data may change with system dynamics, e.g., mode changes. For instance, during the takeoff and landing stages of a flight it is necessary to monitor speed and altitude closely, i.e., they must have a small *avi*. But, it might be appropriate to increase the *avi* once the aircraft is cruising.

A data item that does not reflect the state of the environment any more can still be useful to a controlling system. First of all, it can be used as a predictor. For instance, past wind speeds over a particular flying area can help in determining future trends. Secondly, a particular algorithm might be able to work with approximate state values. Generally speaking, different subsystems may require data with different grades of temporal validity.

4.3 Time Constraints of Actions

Having covered the timing aspects of events and data we are now in a position to summarize the reasons for actions being associated with time constraints.

- Time constraints dictate the behavior of the environment – they constrain the rates and times at which inputs arrive at the system.

  Example: In an air traffic control system, a flight commander must not ask for permission to land until the aircraft is 10 minutes from the airport.

- Time constraints prescribe performance of the system – they dictate the responsiveness of the system to these inputs.

  Example: Once landing permission is requested, a response must be provided within 30 seconds.

- Time constraints are imposed on actions that sense the environment and update the controlling system's database so as to maintain data temporal consistency.

  Example: Actions that update an aircraft's dynamic parameters, such as position and altitude, must execute with specified periodicity. These actions may have deadlines depending on how soon the current data will become invalid.

These three types of time constraints are end-to-end in nature, governing the relationship between the controlling system (the real-time system) and the controlled system (the environment).

The first and second types of time constraints are related to the events that occur and the system's response to stimulus events. This response can itself be considered as an event. Time constraints from the perspective of events were examined in Section 4.1. In some cases, a specified time constraint is specified directly on an action, e.g., action A must complete within t time units. Since action completion is an event, we can view this as a time constraint on the event occurrence also. The third type of time constraint arises primarily due to the need to keep the controlling system in synchrony with the controlled system. In Section 4.2 we examined the details of such data temporal consistency requirements.

Time constraints belonging to a fourth category facilitate the development of feasible solutions – they constrain the internal behavior so that solutions may be derived. For example, the timely synchronization of nodes' views of each others' states is often used to detect faults in a distributed system. If a node fails to receive an "I am alive" message from a healthy node within a specified time-out interval, expensive fault diagnosis procedures may be invoked.

Here as well as in many other situations, time is used as a synchronization mechanism and to achieve coordination. It is important to impose the minimal and least stringent set of time constraints necessary to obtain correctness.

This section can be summarized as follows: Time constraints can be associated with events, data, and actions. Events trigger actions and data temporal consistency is maintained through timely actions. . Thus, from the viewpoint of the controlling system, the various types of timing constraints result in the need for the time-constrained execution of actions. Of course, the methodical translation of the data and event timing requirements into timing constraints on actions is, for the most part, still an open issue. We discuss some of the considerations in Section 6 after examining the implications of not meeting the timing requirements.

5 Dealing with Time Constraints and their Violations

An issue of interest to the designer of a real-time system concerns the scheduling of time constrained actions so as to minimize the penalty resulting from the actions that are either delayed or not executed at all, and to maximize the value accruing to the system from the actions.

If a large negative penalty will result from a delay or non-execution of an action, we have a *safety-critical* or *hard* time constraint.

> **Example:** The deadline set for an aircraft to leave the runway after landing is *safety-critical.*

If there is no value to executing an action after the deadline has passed and no penalty accrues, we have a *firm* deadline. An alternative action (including a null action) is possibly available to deal with the violation of the time constraint.

> **Example:** A transaction that is attempting to recognize a moving object must complete acquiring the necessary information before the object goes outside its view and hence has a firm deadline. If the firm deadline is not met the object must be brought in front of the camera once again and recognition attempted with increased importance.

ON (*deadline of* "object recognition")
IF (action not completed)
DO (*"try again with increased importance*).

> **Example:** If at time t_1 an aircraft has been cleared to land, necessary steps, for example, to lower the landing gear, to begin deceleration, and to reduce altitude, must be completed within 10 seconds. Otherwise, we would like to abort the landing within a given deadline, say 5 seconds; the abort must be *completed* within

the deadline, presumably because that is the "cushion" available to the system to abort the landing without affecting other aircrafts. This requirement can be expressed as follows:

ON (*10 seconds after* "initiating landing preparations")
IF (steps not completed)
DO (*within 5 seconds* "Abort landing").

In the second example above, we have encoded the behavior in terms of the Event-Condition-Action paradigm of active databases to show how the rules can be made to embody the control knowledge.

Many actions can be executed even after their deadlines because the consumer of the output of the actions can live with data that arrives late. Such deadlines are termed *soft*. But if indeed there is a follow-up action to this delayed action, the system must be capable of adapting its decisions to handle this delay so that the overall end-to-end time constraints are satisfied.

Example: An object's features must be captured by a camera, the features must be matched with those of objects in the database and the decision concerning which way to direct the object must all be completed before the object reaches a certain point in its traversal. This sequence has an overall deadline from which the time constraints of the component actions can be derived. Each of these time constraints is a soft deadline since in spite of their violation the overall action might still be able to complete.

It should be clear that firm and soft time constraints offer the system a certain amount of flexibility that is not present with hard or safety-critical time constraints. As we stated earlier, most of the latter arise from external considerations. It is therefore important to minimize the number of such mandated requirements.

The presence of firm and soft time constraints calls for meta-control algorithms which are capable of reacting to time constraint violations by adjusting deadlines and other parameters, such as importance levels, of future actions. These meta-control rules can be encoded as ECA rules.

Given this, an active real-time database system can serve not only as the repository of the data about the environment being controlled, but also as a repository of the control data, triggering the necessary adaptive responses to time constraint violations to effect recovery. These responses can relate to the environment being controlled, e.g., "place the aircraft in a holding pattern in case the runway is unavailable", or to the internal parameters set by the system, e.g., adjust the soft deadline of the "object matching" action in case the "object recognition" action is delayed.

Also ECA rules can be used to appropriately react to overloads, for example, by defining rules to shed load upon the recognition of an overload or to increase the importance of a periodic action whose m out of previous n instances missed their execution.

ON (n^{th} *violation within 10 secs*)
IF (crisis-mode)
DO (*drop all non-essential actions*).

ECA rules can also help in dealing with impending the temporal invalidity of data. Specifically, if a transaction needs a data item and it is (about to become) invalid, it can trigger another action to fetch that data from the environment.

Finally, ECA rules can also be used to govern the way transaction processing is done, especially the way data migrates through the levels of the memory hierarchy to ensure temporal validity, and how logging is done to ease timely recovery[7].

6 Derivation of Time Constraints

Clearly, algorithms are needed to derive the optimal values for the time constraints so as to maximize system performance, be it measured in terms of utilization or productivity of resources or the value. Pre-runtime as well as runtime support tools are necessary for the derivation of time constraints. The pre-runtime algorithms must derive the time constraints such that they have the weakest possible attributes (e.g., soft is preferable to firm) and runtime schemes must be able to maximize the leeway afforded by these weak attributes. This area has received very little attention even though recent results are encouraging.

Let us consider a simple example, one considering just freshness, or absolute consistency constraints of data. These are satisfied by periodic sampling of the environment, that is, executing the action that obtains data from a sensor and updates the database periodically. The issue here, how should the period of this action be set? Consider one of the many possible semantics of actions with period P: One instance of the action must execute every period, as long as the start time and completion time lie within a period, the execution is considered to be correct with respect to the periodicity semantics. Suppose a simple action takes at most e units of time to complete, $(0 \leq e \leq P)$. If an instance starts at time t and ends at $(t + e)$ and the next instance starts at $(t + 2 * P - e)$ and ends at $(t + 2 * P)$, then we have two instances, which are separated by $(2 * P)$ units of time in the worst case. This, for example, will be the case if the rate monotonic static priority approach is adopted.

Suppose the *avi* of *altitude* is 10, i.e., *altitude* must be no more than 10 seconds old. It follows from the above periodicity semantics that to maintain the *avi* of *altitude*, the period of the action that reads the *altitude* must be no more than half the *avi*, that is 5.

Let us assume instead that periodic actions are scheduled so that each instance of an action is guaranteed to start at the same time, relative to the beginning of a period. Then, the worst case separation between the start time of one instance and the finish time of the subsequent instance will be $(P + e)$. Since an action could write the relevant data item any time during its execution, the interval $(P + e)$ must be less than the given avi, giving $P = (avi - e)$.

The above discussion illustrates the dependence of action timing constraints not only on the temporal consistency requirements of the data but also on the execution times of the actions and the scheduling approach adopted. We should strive for a larger period since the larger the period, the lower the resource utilization per unit time, and hence higher the schedulability.

This simple example considered a single periodic monolithic action whose period equals its deadline. Most real-world actions are quite complex and as we have seen earlier the types of time constraints can also be very complex.

Gerber et al. [3] have proposed a solution to the problem of deriving the periods, offsets and deadlines of the subactions in a graph representation of an action so as to meet the temporal requirements, such as freshness and correlation constraints, imposed on the data. The solution assumes a uniprocessor execution environment (even though it could be extended to parallel and distributed environments) wherein all task execution times are known.

The problem is bad enough if we know the structure of the computations *a priori* and the data is in memory. When dealing with computations in disk-resident active database systems, one has to further contend with one action dynamically triggering another action where the computation times of the actions are unpredictable. Since these systems use a priority-driven scheduler, some researchers have begun to explore the problem of assignment of priority to these actions, skirting the problem of assigning time constraints first[4].

7 Discussion

In this paper, we have made a beginning in trying to understand the issues underlying the origin and semantics of time constraints. Not all deadlines are defined by users or by the application, as is said in much of the real-time literature. It is important that the flexibility afforded by derived deadlines be exploited by the control system. By the same token, in those cases where the deadlines are derived, the values must be chosen in an adaptive fashion and the deadline violation must also be handled adaptively. This calls for a sound methodology for assigning/deriving time constraints and for choosing their properties in the least stringent manner.

Given time constrained events, data, and actions, the system must be able to recover from violations of the time constraints and gracefully degrade under overloads. The strategies for recovering from timing violations can be encoded as ECA rules. Thus, an active real-time database can be used to store the data about the controlled system but also to store the meta-data about the controlling system.

Our focus has been on temporal properties. As has been mentioned very often in the literature, in order to achieve timeliness it is possible to relax logical correctness requirements, by relaxing the need for currency, coherency, consistency and completeness. This is an area worthy of investigation.

Acknowledgements

My sincere thanks to Gerhard Fohler, Jayant Haritsa, and Raju Sivasankaran, for their comments and suggestions on previous versions of this paper.

References

[1] N. Audsley, A. Burns, M. Richardson, and A. Wellings. A Database Model for Hard Real-Time Systems. Technical Report, Real-Time Systems Group, Univ. of York, U.K., July 1991.

[2] B. Dasarathy. Timing constraints of Real-time systems: Constructs for Expressing Them, Methods for Validating Them. *IEEE Transactions on Software Engineering*, pages 80-86, Jan 1985.

[3] R. Gerber, S. Hong, and M. Saksena. Guaranteeing Real-Time Requirements with Resource-based Calibration of Periodic Processes. *Transactions on Software Engineering*, July 1995.

[4] B. Purimetla, R. M. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Priority Assignment in Real-Time Active Databases. *Conference on Parallel and Distributed Information Systems*, Oct 1994.

[5] K. Ramamritham. Real-Time Databases. *International Journal of Distributed and Parallel Databases*, Vol. 1, No. 2, 1993.

[6] B. Purimetla, R. M. Sivasankaran, K. Ramamritham, and J. A. Stankovic. Real-Time Databases: Issues and Applications. In *Advances in Real-Time Systems*, Sang Son, Ed. Prentice-Hall, 1995.

[7] R. M. Sivasankaran, K. Ramamritham, J. A. Stankovic, and D. Towsley. Data Placement, Logging and Recovery in Real-Time Active Databases. *Workshop on Active Real-Time Databases*, Sweden, June, 1995.

[8] X. Song and J.W.S. Liu. How Well Can Data Temporal Consistency be Maintained?. In the *Proceedings of the IEEE Symposium on Computer-Aided Control Systems Design*, 1992.