A STUDY OF DISTRIBUTED REAL-TIME ACTIVE DATABASE APPLICATIONS

B. Purimetla, R.M. Sivasankaran,
J.A. Stankovic and K. Ramamritham

CMPSCI Technical Report 93-10
January 1993

A Study of Distributed Real-Time Active Database Applications [1] [2]

Bhaskar Purimetla, Rajendran M. Sivasankaran, John A. Stankovic &
Krithi Ramamritham
Department of Computer Science
University of Massachusetts, Amherst, MA 01003

1 Introduction

Real-time database systems has been an active research area in recent times. In spite of this, their applicability to specific applications has not been well studied. We attempt to rectify this shortcoming by studying two applications: 1) cooperative distributed navigation systems and 2) network services database systems. The former is a hard real-time system and the latter a soft/firm real-time system. Cooperative distributed navigation systems appear in several complex applications such as road following and robot navigation. These systems usually consist of multiple sensor-based agents which work in a distributed and cooperative fashion towards a common goal. These agents are guided by a high level controller which facilitates coordination and cooperation among the agents. Network services databases provide support for the service-providing intelligent network (IN). IN provides services like dialed number services (800 service), personal mobility service, virtual business group service and televoting service. The data and the service logic in this system is distributed.

These applications are of particular interest, because, a lot of concepts have been developed by assuming certain characteristics of an application for real-time database systems. For example, it is assumed that transactions have timing constraints, but how those constraints are derived from the application needs is not properly studied. Similarly various other data and transaction characteristics are assumed. In this paper we will study two real-time active database applications in detail. We will discuss their data and transaction characteristics in detail and see if the protocols and features developed in the research have applicability in these applications.

In section 2, we will consider the motivations for using a real-time active database for these

[1]This work was supported in part by NSF under grants IRI-9114197 and IRI-9208920.

[2]A report on the recent NATO Advanced Study Institute on Real-Time Computing identified an important missing ingredient in much of today's real-time research, i.e., we must take into account real needs of applications in developing appropriate solutions. This paper addresses this issue with respect to real-time databases.

complex applications. In section 3, we will discuss in detail the cooperative distributed navigation application and in section 4, we will study the network services database application. Then finally in section 5, we will give some concluding remarks.

2 Why a Real-time Active Database (RTADB)

Real-time databases combine the principles developed in traditional databases and real-time systems. They present several new issues that need to be addressed such as transaction predictability, temporal data consistency notions and real-time recovery. Most of the problems arise because traditionally databases have been designed to maximize throughput while maintaining the desired data consistency [3], whereas in real-time systems the ability to meet the time constraints is of paramount importance. Real-time databases try to meet timing constraints while maintaining desired data consistency and permanence. Hence, all the protocols including concurrency control and CPU scheduling protocols must be designed to be time cognizant.

Real-time databases have close connection with active databases. In active databases, not only is the data stored, but so is the control knowledge. This control knowledge specifies the requisite actions to be taken when specified conditions hold and specified events occur [5]. This paradigm is highly suitable to implement real-time database systems as usually these systems control real-world processes. Event driven control is easily specified using the event-condition-action (ECA) construct provided by active databases. The semantics of an ECA rule is that if the specified event (E) occurs and if the condition (C) is true then execute the specified action (A). An active database that explicitly takes time constraints into consideration provides a convenient abstraction to implement complex real-time systems.

There is a basic question as to whether there is a need for a real-time database system in the cooperative distributed navigation systems and network services database systems. For some small applications a simple file system may suffice. However when a lot of data sharing occurs and the data need to be maintained consistently (in spite of concurrent updates) and efficient and convenient access to that data is required, then a simple file system fails to support the needs. In this paper we will motivate the need for using a real-time database system as the main structuring component for these applications.

Distributed RTADB combines the features of traditional distributed databases, real-time systems and active databases. The need for a distributed RTADB arises when the application needs a distributed database and the data manipulated has temporal properties. Usually in such applications the environment is dynamic and needs constant monitoring. There is a need for a

[3]The most commonly used data consistency criterion in database systems is that of *serializability*. This notion is very strong and can be relaxed for real-time databases.

distributed database when the application requires efficient distribution, storage and retrieval of data on a large scale and transaction support, where the transactions have all or a subset of ACID (atomicity, consistency, isolation, durability) properties that ensures desired correctness and data integrity under concurrent accesses. The cooperative distributed navigation systems need access to large scale maps for calculating their position. Various indexing structures like quad trees are needed for this purpose. In addition database systems allow interactive queries by manual operators and facilitate debugging. Network services have been using databases in some form or the other. It is obvious that IN will use databases in future because of the volume and distribution of data (service logic, customer records, traffic management data, network configuration data) and concurrent accesses to it.

The need for real-time support comes when timely response of the system is a must. This requirement translates into transactions having to meet real-time constraints and be predictable. In cooperative navigation systems, there are a lot of actions which have strict time constraints. For example, if an obstacle is detected in an agent's path, a remedial action has to be taken within a certain time depending on the situation and the deadline is hard. Similarly in the IN the queries that go from the switches to the central database have stringent timing requirements. In the IN there is real-time data such as the traffic management data and the network configuration data that requires to be logically and temporally consistent. This also facilitates the need for real-time database support because the data needs to be consistent in the face of concurrent accesses. In real-time systems since timeliness is very important it might be possible to relax the strong notion of serializability (ACID properties). We can relax atomicity in certain contexts to get a timely response (monotonic queries) and isolation if the desired response can be inconsistent within certain limits (epsilon serializability [8]). In the next section we will see in detail about the consistency and permanence requirements of the data in IN databases.

In a complex real-time system there are a number of significant events like the timer events, real-time events in the environment, apart from the transaction generated events. ECA construct provided by active databases is a powerful mechanism to model the transactions that are to be triggered on such events. The cooperative navigation systems require many event driven actions. For example, if two agents come within collision distance, then the controller has to take a preventive action. If an obstacle is detected in the planned path, a new path may have to be computed. Similarly, in the network services database there are numerous such events. Incoming call, the traffic crossing a threshold, timeout after a query, and the answer for a service query are examples of events found in our application. One of the features that is lacking in current active databases is the explicit specification of time. Once we have that the ECA paradigm is a convenient one to model the events in a real-time system.

3 Cooperative Distributed Navigation Systems

Cooperative navigation systems usually consist of multiple semi-autonomous sensor based agents, which are coordinated by a high level controller to cooperatively and distributedly achieve a particular goal. The agents sense the environment (capturing images through a camera) and relay the data back to the high level controller if necessary. Each of the agents has an onboard frontend system which filters data before sending it to the high level controller. Also a limited amount of control knowledge is incorporated in the frontend system in order to perform some reflexive actions. The frontends may prefetch maps of their current position from the controller. Each of the agents perform some local matching in parallel to verify their position and pose (angle of placement). The high level controller is primarily to handle events which the frontend can't handle with its limited capabilities and to preserve them correctly in stable storage for future perusal. The events of different agents can be handled in parallel by the controller. The high level controller also performs as a coordinator of the multiple agents. The high level controller may be an RTADB, with control knowledge incorporated into it. The high level controller also maintains a map database and associated data such as the positional information of the agents, the path-plans of the various agents, destination information, archival data for future processing, various index structures which support efficient access, etc. The control knowledge consists of the various actions that need to be taken upon the occurrence of some pre-specified events.

The system executes the following control loop. As the agents travel in the real-world they sense and produce image data. The frontend system compares this data with cached image data and takes immediate actions when necessary. If the matching is unsuccessful then the frontend relays the information to the high level controller. The controller then performs a more sophisticated matching and identifies possible threats/obstacles which may not have been expected (like a car parked haphazardly in the path of an agent). The controller can do the matching of the images from different agents at the same time in parallel. The controller then relays the appropriate commands back to the agent. The entire control loop has timing constraints which depend on the position, direction and velocity of motion of the agent. Also the operator can query and intervene in the proceedings by issuing queries/commands to the high level controller. This is needed for emergency situations and manual overruling by an operator.

3.1 Application characteristics

In this section we will study the characteristics of data and transactions present in the navigation application. We will also discuss how the transactions derive their deadlines and also how the ECA paradigm of active databases helps in encoding the control knowledge in the system.

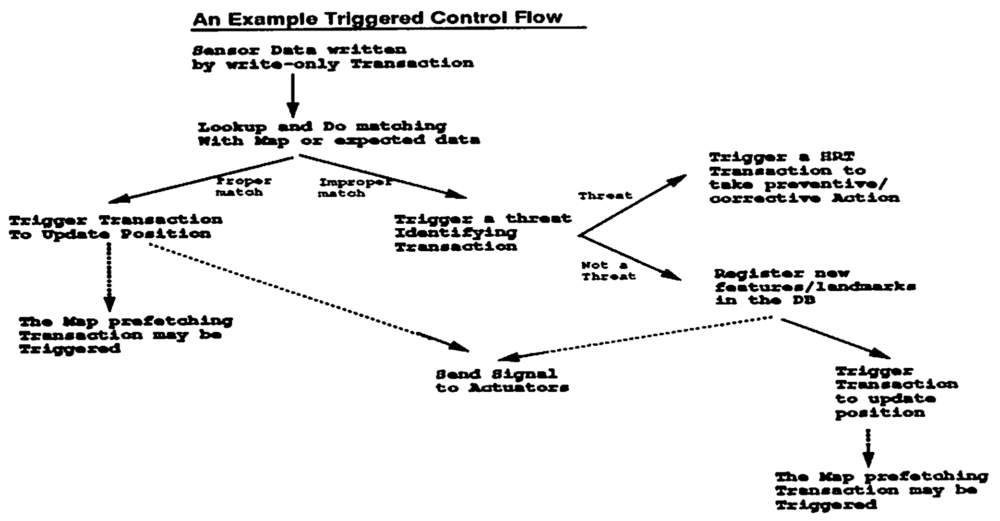
An Example Triggered Control Flow
Sensor Data written by write-only Transaction
Lookup and Do matching With Map or expected data
Proper match
Improper match
Trigger Transaction To Update Position
Trigger a threat Identifying Transaction
Threat
Trigger a HRT Transaction to take preventive/ corrective Action
Not a Threat
Register new features/landmarks in the DB
The Map prefetching Transaction may be Triggered
Send Signal to Actuators
Trigger Transaction to update position
The Map prefetching Transaction may be Triggered

Figure 2

3.1.1 An example control flow

In this section we discuss how the common case control loop looks in the navigation application and the triggered flow of transactions which might occur if we implement it using the ECA paradigm of active databases. The flow diagram is shown in Figure 2. An instance of the loop is initiated when the sensors write the image data into the system. This usually occurs at the frontends and can be implemented with periodic write-only transactions. Once the image data is available, it is processed and compared with the expected scene constructed using the map data and considering the present position and direction. This matching can also be done at the frontend system. If the matching is completely successful then there is no need for the agent to contact the high level controller. The position is updated and, if necessary, the map data is prefetched from the high level controller. The transaction control flow can be coded using the ECA rules very conveniently.

If the frontend system doesn't detect a successful match then the data can be relayed to the high level controller for more sophisticated analysis. Here the data is analyzed to identify any new landmarks/features or any perceivable threats/obstacles. A landmark/feature is any environmental feature of sufficient distinction to allow robust recognition from a large range of viewpoints. The set of possible threats/obstacles an agent can come across in the environment can be stored in the database. Similarly the map data is annotated with the set of landmarks/features known to be present in the scene. If present, the threats/obstacles trigger a hard real-time task which sends appropriate corrective/preventive commands to the actuators. The deadline of the task depends upon the position, velocity and direction of movement of the agent and the type of the threat. For example, a car coming in the opposite direction on the same lane demands a tighter deadline than a stationary vehicle, since the relative velocity is larger in the first case, and the value of the data itself affects the choice of transaction deadlines. One important point here is that the entire central control loop has to be executed with a very high priority until it has identified as to whether any threat is present. This is because it doesn't know whether or not there is a crisis until the matching is completed. Once a threat is identified the mode change can be done to execute in a crisis mode, if the threat demands high priority. If there are multiple threats, then either they are processed in order of priority or an action to counter all of them is taken. If there is no threat, then any new permanent landmarks/features are consistently entered into the database using update transactions. These are necessary for future use as well as to aid cooperation among the agents. In the case of multiple threats/landmarks then there may be a requirement to identify as many of them as possible in a short time. In this case there is no need to identify all of the threats/landmarks before proceeding further. But the time available for doing the job may be constrained by the present velocity of the vehicle and other physical parameters. This leads to soft real time transactions with firm deadlines. It may be useful to borrow ideas from the theory of imprecise computation if a threat cannot be completely identified. The system can roughly identify the characteristics of the threat and assume a worst case threat possibility. An incomplete result may be useful here.

In addition there will be *time-triggered* as well as *periodic* actions, which can also be conveniently specified using the ECA model. For example, periodically, the system verifies whether proper progress is being made towards a goal and, if not, a new plan can be calculated and actuated. Similarly if we detect a moving object for the first time while it is a safe distance away we keep track of its movements relative to the agent until it materializes as a threat or it exits the sensor range. This can be done by triggering a periodic transaction as soon as we recognize a potential threat which is quite far away. Archival data may be very useful here to monitor the situation progressively. The characteristics of a moving threat including its velocity, can be determined from the previous snapshots along with their temporal relationship. The high level controller may need to maintain such archival data. Another example is the periodic reception of information about the status of a traffic accident. The system analyzes it periodically and may do alternate planning depending upon the extrapolated trend.

3.1.2 Data Characteristics

The navigation application has data of different characteristics with varying consistency criteria, recovery criteria and access patterns and durability needs. Here we will discuss the different data present in the application and their various characteristics.

The data in a real-time database not only has to be consistent but also be temporally correct, since the contents of the database have to reflect the current status of the outside world. The notions of absolute and relative temporal consistencies [9, 2] guarantee that the system is consistent with the outside world. Absolute consistency requirements on data specify that the sampled data stored in the database must not lag behind the actual real world process being sampled by more than a certain time. If a data item X has an absolute validity interval a_X, then the value of X at any given time T has to correspond to the real-world value of X at some time in the interval $[T - a_X, T]$. This ensures that the state of the real world environment and its projection in the database are approximately consistent at all times. Relative consistency is required when we want to derive new data using data which have to be contemporary to each other in time. It specifies that the difference in timestamps of the data items being used to derive new data has to be less than or equal to a certain bound. If a set of data items S has a relative validity interval r_S, then $\forall d, d' \in S, \mid d_{timestamp} - d'_{timestamp} \mid \leq r_S$, where timestamp of a data item denotes the real-time when the observation relating to that data item was made..

The input to the system consists of sensor data obtained from cameras and other sensors. This data has absolute temporal consistency requirements as they have to reflect the real world closely. In addition, absolute validity intervals may vary depending upon the speed of the agent as well as the mode it is in. A Crisis mode may demand a smaller validity interval as we need to sample the world faster. Similarly when the agents are traveling at a high speed, there will be tighter absolute consistency requirements. If absolute consistency is not satisfied and the mode is normal, then extrapolation of archival data can be used to maintain consistency. The concept of invoking contingency transactions for compensation has applicability here. If two or more sensors from one or more agents are used to track a common target, then relative consistency among the data from these sensors needs to be maintained in order to make correct decisions. Again the relative consistency requirements may vary depending upon the mode of the system. As to the conventional consistency requirements, the sensor data is written by only one transaction type and no concurrency control is needed. Also, sensor data is strictly temporal in nature and may not require conventional recovery. If a transaction which updates this data is aborted then instead of doing a conventional rollback, the data can be declared invalid since the data will be updated during the next sampling period. This simplifies the recovery protocols.

The control loop attempts to identify the changes in the observed scene from the expected scene on a best effort basis. The system tries to identify as many of the threats/landmarks as possible before the deadline. This data is derived from the sensor data. The absolute validity deadlines may also have to be derived from the sensor data it uses. Also multiple firm real-time transactions may be used to compare and analyze different sections of the image. The different landmarks/threats may be identified by different transactions. This leads to relative consistency requirements on the data, that all these transactions finish within a certain interval. Since there is only one transaction type which updates this data, there is no need for any concurrency control. The system archives the threats/landmarks identified in the present scene for future perusal. This data may need the property of permanence.

The map data and the data of possible threats doesn't have any temporal consistency requirements. They require the conventional consistency criteria as both reads and writes are possible by multiple transaction types. Recovery is required to maintain the durability of the data. Also this data is the target of interactive queries and offline updates.

The path plans of various agents also do not have any temporal consistency requirements. Since they can be updated by both the operator and the system, they require the conventional consistency to be maintained. Conventional recovery may be needed to ensure atomicity and durability.

The output data of the system consists of the actuator commands for steering and stopping the agents. They have absolute consistency requirements since the data has to be updated once every command interval. The requirements may vary according to the speed of the agent as well as the mode of the system. The same recovery criteria applicable to the sensor data are valid here too.

3.1.3 Transaction Characteristics

A wide variety of transactions are likely in this application. Transactions can be classified as periodic/aperiodic, hard/soft/firm real-time [9, 6] and on the basis of their data access type like write/read/update. Hard real time transactions are those which have to be completed before their deadline or a huge negative value is imparted to the system. Usually all hard real-time transactions are preanalyzed and guaranteed to complete. Firm real time transactions are those which have to be completed by their deadline or their value becomes zero just after the deadline expiry. Soft real-time transactions are those whose value decreases proportionate to the amount of delay in completing them after the deadline expiration until it becomes zero at a point called zero-value point. They have to be completed by their zero-value point or else aborted. Here we will study the various transaction types present in the application along with their characteristics.

Transactions updating sensor data are periodic hard real-time transactions with their periods and deadlines dependent on the absolute consistency requirements of the sensor data. They are just write-only transactions with known data requirements. They don't face any data contention.

Similarly the image matching transactions and position update transactions are aperiodic hard real-time transactions. They also don't face any data contention, as their data requirements are fixed and only they update the data. These transactions are triggered by the completion of the sensor update transactions. In essence they are triggered once every period of sensor value update transactions.

The threat/landmark identification transactions are aperiodic transactions which are triggered only when a successful match doesn't occur. These are firm real-time transactions which are capable of giving incomplete results which can be used to construct a worst case scenario. If there are multiple threats, then a subset of them can be identified and returned by the deadline. The measure of success is the percentage of landmarks in the scene that are identified by the deadline. Deadlines depend on the type of the threat and the speed of the agents. Since the type of the threat is unknown, a very high priority execution is appropriate. They don't have any data contention as they are append-only transactions.

Threat handling transactions are hard real-time transactions whose deadlines vary according to the type of the threat and speed of the agent. They have to be completed by the deadline in order to avoid a catastrophe. For example if the agent is moving at 55mph and a 0.5G deceleration is possible then it takes 22.5m to come to a stop. The deadline must depend upon the distance between the threat and the agent, and whether the threat is static or moving.

The map data prefetching transactions are aperiodic firm real-time transactions with deadlines based on the speed of the vehicle. The frequency of invocation depends on the speed as well as the terrain type. If we are moving on a road full of curves then we may need to prefetch more often than when traveling on a straight section of highway. They are read-only transactions.

The actuator output update transactions are periodic with period and deadlines derived from the absolute consistency requirements of the output data. If the actuators have to be written to every 10 sec, then the period will be appropriately defined. Since they are just update transactions and don't share data, there will be no data contention. These transactions depending upon the result of the matching cycle and the map data give the appropriate commands for moving the agent forward.

Apart from these there are operator interactive transactions such as plan overruling, set of threats updating etc, which are aperiodic. The overruling transaction will be run at highest priority for responsiveness and safety reasons. These face some potential data contention and can be handled by a simple concurrency control mechanism.

4 Network Services Database

The telecommunication industry currently provides services such as the 800 number service and personal mobility service. In the future they want to provide numerous additional such services. Hence, they require a robust, evolvable Intelligent Network (IN) architecture which will be the backbone of such services. The features they are looking for in the IN are service independent architecture, flexible distribution of service logic and service supporting functions, user friendly, flexible service creation functions, open architecture with a set of standard interfaces, compatibility with existing networks, self-awareness, and self-adapting and self-provisioning capability [1].

In this section we will discuss network services databases as an application for dealing with real-time database issues. First we will briefly discuss the current implementation of the 800-service and then explain why network services databases can be considered as real-time database applications. We will look at the different aspects of the application such as the data characteristics, appropriate consistency notions, and transaction characteristics. Though the application may appear restrictive in a certain sense due to the strict standardization of telecommunication networks, it has all the needed ingredients to make it a challenging distributed real-time database problem.

The 800 number service network consists of five major components. They are Service Switching Points (SSP), Signal Transfer Points (STP), Service Control Points (SCP), the Signal Engineering and Administration System (SEAS), and the Service Management System (SMS) [11]. The SSPs are simple electronic switches and the STPs are highly reliable packet switches. The SCPs are on-line, fault-tolerant, real-time databases containing 800 records. They handle inquiries from SSP and return data to the SSP for call processing. Supporting the STP in a geographic area is the SEAS which the STP uses to route queries to the appropriate SCP. SEAS also collects traffic data from STP that is used for network management and engineering. The last architectural component is the SMS, an interactive operations support system that is used to maintain the network services customer records. The underlying network connecting these systems is the Common Channeling System (CCS) network that uses the Signaling System #7 (SS7) protocol. When an 800 number is dialed, the call is routed to a SSP. The switch launches a query via the CCS network which routes it to the SCP. The query contains both the 800 number and the originating station number. The SCP which is connected to the SMS and acts as the database that provides the answer for the query.

4.1 Application Characteristics

In this section we will look at the data and transaction characteristics of the network services database application. We will first discuss a typical execution of the application and discuss the kinds of data being accessed and the types of transactions that are executing. Most future services in IN will to be very similar to the 800 service. The following steps show an abstract view of a typical processing of a service.

```
1. Sense the incoming call
```

```
2. Activate the appropriate trigger in the trigger table
3. Query the appropriate database for service logic
4. Transfer the service processing to a different
   node, if necessary
5. Query the appropriate database for data
6. Do the accounting
```

We are not concerned with steps 1 and 2. These will be usually performed in the switching element. Trigger checkpoints may be set in the call states where call processing can be interrupted and required actions taken. Triggers can be set on a per-line, per-group or per-office basis. Off-hook trigger, dialing-plan trigger and automatic-route selection trigger are some of the trigger types. However it is interesting to note that trigger activation in step 2 can be modeled using the ECA mechanism used in Active databases to trigger transactions. The query in step 3 will be a read query that gets the service logic. We need a flexible distribution of service logic and the network has to be self-adapting and fault-tolerant. With these requirements in mind, step 3 can be translated into the three steps as follows

```
3.1. Query the appropriate database for traffic data
3.2. Query the appropriate database for configuration data
3.3. Get service logic from the NEAREST node
```

Since there is a necessity for *intelligence* in the network, and to meet stringent timing requirements, we have to constantly monitor the *environment* to improve the performance. Since transferring should be done in an *intelligent* manner, Step 4 will be similar to step 3. This step can spawn off a whole new service processing procedure. Step 5 can be similarly broken down except that the query can be read/write. Step 6 is again similar to step 3 and will mostly consist of updates.

We have described the application from the view of a call processing node. There are nodes that do the operations support and administration (SMS) and nodes responsible for network traffic management and engineering (SEAS). We are not going to explain The details of interaction between these nodes are not explained as the discussion on 800 service system gives a sufficient picture of the interactions.

4.1.1 Data Characteristics

The different logical data entities that are in the IN are service logic, trigger tables for the service logic, customer service records, account records, operations support data and Network Traffic Management (NTM) data. All these data are distributed and are subject to concurrent accesses. There is data that is mostly read-only and subject to infrequent updates like the service logic and trigger tables. There is read/write data like the customer record base and account information base. The most dynamic of all this data is the NTM data which gives the ability for real-time monitoring of the network making it self-adapting and fault-tolerant. In the future, most services require automatic call distribution capabilities. The NTM data can be gathered over a period of time that will provide us load statistics which can be used for optimal configuration of distribution of data and service. This data is not as dynamic as that used for real-time monitoring.

NTM data has to be *temporally consistent*, i.e., there is a need to maintain consistency between the actual state of the network and the NTM data. The notions of ***absolute consistency*** and ***relative consistency*** are applicable to NTM data. The sampled data of the real world (NTM data) should not lag behind the actual data of the real world (network traffic) by more than a specified time. This specified time is the ***absolute validity interval (avi)***. The NTM data used to derive other data should not differ from each other by more than the ***relative validity interval (rvi)***. The NTM data for any node i can be the status of the node ST_i (idle or busy), length of the queue of calls to be serviced SQ_i and length of the queue of calls to be transferred TQ_i. We can assume a hierarchical network structure where the network can be divided into regions and regions divided into subregions and so on. A data item d can be represented by the triple $(value, avi, timestamp)$, where d_{value} is the current value of the data item, d_{avi} is the time interval following $d_{timestamp}$ during which d is absolutely valid. ST_i, SQ_i, and TQ_i can be used to derive some information about the load on that node ($load_i$) and they can form a ***relative consistency set***. The loads of nodes in a subregion can be used to calculate the load in that subregion ($load_s ub_i$) and they can form a relative consistency set. The load of the nodes in a subregion and the configuration of the subregion can be used to compute the routing table for the subregion. These NTM data can form a relative consistency set. An rvi can be associated with any such set R denoted by R_{rvi}. Any data item d is in a correct state if and only if it is logically consistent and temporally consistent (absolutely and relatively).

The dynamic data that is used for real-time monitoring is temporal. Failure of a transaction that updates such data may not require conventional rollback and recovery protocol because the next transaction that updates this data will restore it to a consistent value. The best way to recover would be to just trigger a transaction to sense the value again. This can be done only if the relative consistency holds (rvi of the of relative consistency set R to which the data item belongs is greater than difference between the current time and the timestamp of each data item in R).

The non-dynamic data like the service logic, customer records, and accounts information do not have temporal consistency requirements. A conventional database will suffice for this purpose. There will be a necessity for conventional concurrency control mechanisms. It is possible in the case of account information to relax the concurrency control (isolation property of the accessing transactions - epsilon serializability) to give an approximate account information. The NTM data that is gathered over a period for engineering purposes can be treated in a similar fashion.

4.1.2 Transaction Characteristics

Transactions can be broadly classified as aperiodic or periodic, hard, soft or firm real-time depending on how critical they are, and on the basis of data access type i.e., read-only, read-write and write-only. This application includes all types of transactions except hard real-time transactions. Hard real-time transactions are usually found in hard real-time systems where the failure to meet a certain deadline by some transactions will result in a catastrophe. Realistically, the network database application can be viewed as a soft real-time system. But it is possible that the telephone network may be used as a part of a hard real-time system in the future.

Examples of periodic transactions include those responsible for updating SCP database using the data from SMS. Other periodic transactions include those responsible for obtaining the traffic data, and those responsible for billing. Let us briefly look at the transaction that obtains the NTM data for real-time surveillance. The *avis* of the NTM data require these transactions to be executed with a periodicity equal to the *avis*. These transactions have soft deadlines, because failure to do so will not be catastrophic and obtaining the traffic data after expiration of deadline might be useful for later routing. There are other transactions such as the monthly billing and accounting that are periodic and have soft deadlines.

The aperiodic transactions are the ones that are executed to process a service call. All the transactions that are part of this larger call processing transaction will be aperiodic. The call processing transactions have firm deadlines because they impart no value to the system once their deadlines expire. The transaction that queries the database about the loads that is part of a call processing transaction is monotonic in nature. It is good to know the load about as many subregions as possible for optimal routing and with more time we have more (better) results. In these kind of transactions we can relax the atomicity property. Partial completion of such transaction is not totally undesirable. The transactions that are part of a call processing transaction have soft deadlines. It might still be possible to complete a call after a miss in deadline in one of the many queries that is processed as part of the call. It does not make sense to give up and respond with a failure message to a caller just because a query to obtain the service logic is delayed. There are other aperiodic transactions such as the ones executed to process a call for information on accounts. In these transactions it is possible to relax the isolation property (more concurrency), if the customers are satisfied with an approximate information.

Notions of epsilon serializability is applicable to such transactions. Even in transactions that query the traffic data (part of call processing) an approximate information might be sufficient thus allowing to relax their isolation property.

There are all kinds of transactions with varying access patterns (read-only, read-write and write-only). The transaction that reads the configuration (probably a subtransaction of the call processing transaction) will be a read-only transaction. The transaction that looks for the address translation is a read-only transaction. The report generating transactions are read-only. There are other transactions like the SCP update transaction that is write-only. The transaction that does accounting is an example of a read-write transaction.

It should be noted that the application is conducive to the usage of the ECA paradigm from active databases. We can set triggers to update routing data when the traffic load reaches a minimum or a maximum value. The physical address depends on the time of the day. We can set triggers to update the address database when the time of the day is a certain value. We can trigger transactions to do the billing when certain events occur. For instance at the end of the month we can trigger transactions to calculate the bill.

5 Conclusion

Distributed cooperative navigation and network services database are rich and challenging applications for real-time active databases. The first one is a hard real-time system, whereas the latter is a soft real-time system. They contain rich data and transaction semantics which can be exploited to design better protocols for concurrency control, CPU scheduling and recovery, to improve the performance of the system. Different data have different recovery and consistency characteristics, which aids in the design of less expensive permanence maintenance protocols and allows more concurrency respectively. There are many interesting issues which have to be considered. These include the computational complexity of transactions, preanalysis of the rule set to guarantee predictable response times, etc. For example, if the system's integrity constraints are implemented using triggers, then how does one assign deadlines to the transactions? Does the restoration of consistency, which takes several transactions, as a whole has a deadline or each individual transaction has a deadline? Many similar questions arise whose answers may require the preanalysis of rule sets. The control structure provided by active databases seems to be highly suitable for the navigation application. As of now few studies have addressed the problem of making the active databases actively consider time constraints in their design and implementation. Many issues arise in this area as well. In conclusion we can say that RTADBs simplify the design and maintenance of such distributed applications which need access to large amounts of data and also desire data consistency.

References

[1] AT&T Technical Journal, "Intelligent Networking: Network Systems", Volume 70, Summer 1991.

[2] 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.

[3] R. L. Bennett, J. C. Chen, and T. B. Morawski, "Intelligent Network OAM&P Capabilities and Evolutions for Network Elements", AT&T Technical Journal, Summer 1991.

[4] J. O. Boese and A. A. Hood, "Service Control Point - Database for 800 Service", Globecom '86, 1986.

[5] U. Dayal et. al., "The HIPAC Project: Combining Active Databases and Timing Constraints", *SIGMOD Record*, 17, 1, March 1988.

[6] J. Huang, J. A. Stankovic, D. Towsley, K. Ramamritham, "Experimental evaluation of real-time transaction processing", *Proceedings of 10th RTSS*, Dec 1989.

[7] H. F. Korth, N. Soparkar and A. Silberschatz, "Triggered Real-Time Databases with Consistency Constraints", *Proceedings of the 16th VLDB Conference*, 1990.

[8] K. Ramamritham and C. Pu, "A Formal Characterization of Epsilon Serializability", COINS Technical Report 91-91, 1991.

[9] K. Ramamritham, "Real-Time Databases", *International Journal of Distributed and Parallel Databases*, 1993.

[10] T. E. Newman, D. P. Worrall, J. W. Murphy and W. F. Woodbury, "Intelligent Network Control of BOC 800 Service: The Service Management System", Globecom '86, 1986.

[11] N. Redding, "Network Services Databases", Globecom '86, 1986.