DISTRIBUTED, OBJECT-ORIENTED, ACTIVE, REAL-TIME DBMSS: WE WANT IT ALL - DO WE NEED THEM (AT) ALL?

Alejandro P. Buchmann, Christoph Liebig

Department of Computer Science, Darmstadt University of Technology, Wilhelminenstr. 7, 64283 Darmstadt, Germany

Abstract: Whenever technologies converge there exists the potential for huge benefits but also the risk of failure. The main pitfall when combining technologies that evolved independently consists in attempting to provide the union of features without properly considering the often incompatible assumptions and the crosseffects. In this paper real-time databases, active databases, and distributed object systems are analyzed together with some of the basic assumptions underlying previous work in these core technologies. Crosseffects and potential incompatibilities are discussed in an attempt to provide a better foundation for a configurable middleware platform that realistically combines selected features of active, real-time and distributed object systems.

1. INTRODUCTION

The rapid evolution of today's event-based computing environments and the increased use of distributed systems in time-critical applications combined with the desire of reaping the benefits of objet-orientation, has lead researchers to investigate ever more complex systems. At the same time, the combination of these features requires a thorough understanding of each of the underlying technologies, and more important yet, of the possible cross-effects. The simple concatenation of buzz-words will lead to ill-understood systems whose behavior will be unpredictable and may pose a danger to life and/or property.

If the semantics of complex system software are not well understood the users will prefer to implement themselves a minimalistic version of the required functionality. The resulting systems are ad-hoc solutions that are expensive and can neither be extended nor exploited in a different context. Therefore, a middleware platform with clear semantics is needed that can satisfy the requirements of a variety of applications. The research community has tried to take up this challenge by combining (at least on paper) active databases, real-time databases, and distributed object systems in a single platform. Unfortunately, the full set of features of each technology is incompatible with the other technologies that are being combined, and cross effects between technologies have not been sufficiently studied and considered.

In this paper we try to highlight some of the problems that may arise when combining technologies with divergent goals and requirements. For example, real-time systems require predictability of resource consumption and execution time in order to give performance guarantees; active databases react to events and trigger rule executions that dynamically alter the work load and thereby make any form of predictability rather difficult. The same is true for object-orientation with its tendency to encapsulate the behavior of objects and to hide their internal implementation while real-time systems need the implementation details to predict worst case execution times. Finally, the lack of a global clock in distributed systems and the uncertainty caused by varying communication delays introduce many additional factors of unpredictability to both the active and real-time behavior of a system.

When faced with the seemingly insurmountable difficulties and apparent contradictions outlined

in the previous paragraph one might be tempted to conclude that a combination of active, real-time, object-oriented and distributed functionality is impossible to achieve. Yet there are applications that do require several of these features. The question, therefore, is not whether we do need these features but rather: how can we combine meaningful subsets of them into generic software platforms with clear semantics that are modular and capable of satisfying the requirements of a variety of applications.

2. APPLICATIONS AND THEIR REQUIREMENTS

Applications that are often mentioned as motivation for research in the area of active, real-time, distributed databases fall typically into the domains of control, navigation, mobile systems, telecommunications, simulation, e-commerce auctions and profiling, and certain aspects of complex workflows.

Air traffic control systems cover a variety of aspects. While active real-time systems are typically mentioned to support the air traffic controller by filtering the information glut and alerting the controller of dangerous situations, there are several other interesting aspects [Liebig, Boesling and Buchmann 1999]. For once, the portion dealing with take-off preparation and the gate-to-runway movement are less time dependent but represent more complex workflows that are triggered by events. The new ATC systems will also depend more on electronically transmitted data delivered to the cockpit instead of voice traffic. To avoid swamping the crew with irrelevant information, event-driven publish-subscribe mechanisms that must fulfill reliability and timeliness constraints have been proposed. The new generation of ATC systems has been specified to use object-oriented middleware wherever feasible.

In [Locke 1997] aircraft mission control and spacecraft control are described. They have similar properties and are characterized by small main memory databases with extremely short mean latency requirements (0.05 ms mean latency and 1 ms maximum latency). Databases for these environments are typically used to keep track of the air- or spacecraft infrastructure in addition to sensor data. These systems are typically built around cyclic executives. The resource limitations that are typical for aircraft and space vehicles make complex, multifunction software impractical in many cases.

On-board navigation systems cover a wide range of support systems with quite different demands. However, they all must obey strict timing constraints since the time available for a reaction is dependent on the movement of the vehicle. While the constraints may vary based on the type of vehicle, they depend on the determination of the current position (the event) to issue the pertinent instructions, either to a human operator or to an automatic controller. Many on-board navigation systems are built as cooperative distributed systems [Purimetla et al. 1995] in which the front end agents perform sensing and filtering operations while the back-end controller handles the events that the front-end controllers cannot handle because of their limited capabilities. Typical of on-board navigation systems is a fairly small portion of dynamic data that must be combined with a large amount of static data, e.g. maps and landmark information. The most demanding on-board navigation and control systems are those used for control of rather unstable fighter planes that are essentially flown by the on-board computers. While the signals detected by the sensors can be regarded as events, one must be careful not to imply that this means the kind of event handling associated with active databases. Instead, sensor signals are used to invoke specific methods and persistence of the sensor data is only required for auditability purposes. It is interesting to observe that much of the on-board navigation and control software for the new generation of fighter aircraft is being built in an object-oriented manner and that the TAO Object Request Broker and its Event Service have been modified to support hard timing constraints for use in this context [Harrison et al. 1998]. Industrial interest in real-time distributed object systems is demonstrated by the recently approved real-time CORBA specification [Object Management Group (OMG) 1999].

In the automotive industry manufacturers are exploring emerging technologies for spontaneous networking, such as Jini [Arnold et al. 1999], as a mechanism for integration of navigation systems, embedded systems and the infotainment that is expected to be part of the next generation of automobiles. In the latter case one has, in addition to the distribution, the typical timing problems associated with mobility and spontaneous networking.

In telecommunication applications [Raatikainen 1997] the notion of timing constraints is interpreted mostly statistically, i.e., a predetermined percentage of the transactions must meet its deadline. This approach is more a high throughput rather than a typical real-time approach. The functionality that is frequently mentioned encompasses Intelligent Network functionality, such as verification of PINs, call forwarding, user management actions to update, for example, the call forwarding option, televoting, mass calling, etc. The highest requirements are expected from mobile networks with extremely high transaction rates and requirements that 96% of the queries

have a response time of 150 ms or less and the updates be performed in about 1 sec. with reliability of just a few seconds/year downtime. Many proposed telecommunication architectures assume an object-oriented paradigm and try to include standards, such as CORBA.

Simulations provide another class of demanding applications. Training simulation tries to present the trainee with a realistic environment and typical situations to which he must react. These systems often consist of two databases, a static read-only database for the environment and a dynamic database for the simulated situation.

Virtual testbenches are combinations of simulated systems with hardware in the loop. Such a virtual testbench may invoke very complex simulation routines that have timing constraints imposed by the hardware in the loop. For example, portions of a car might be simulated while others are actually running on the testbench and are being monitored and measured. The timing constraints imposed by the hardware in the loop are hard timing constraints and the volume of data may be extremely large since data gathering and analysis is the prime purpose of these systems. Test sequences and the simulation itself may be governed through plans or workflows that are conveniently expressed as event-condition-action rules.

In the area of e-commerce many interesting applications are just emerging, and the requirements of these applications with respect to activity, real-time and distribution have not been well analyzed. There are situations in e-commerce that appear to call for many of the features commonly associated with active and real-time systems. For example, profiling mechanisms must respond in real-time to the actions of a visitor to the site in order to retain the visitor and enhance the probability of a sale [Datta 1999]. This is an application in which no transaction processing is required but fast search and retrieval of profile data is required. The timing requirements are soft since missing a deadline does not invalidate completely the result of the profiling but the value of it may be diminished. In electronic auctions the proper timestamping and ordering of events may be critical.

As can be seen from the applications briefly described, there is no set of requirements common to all of them. However, some interesting generalizations may be possible.

Some form of monitoring and event-driven processing is common to all the analyzed applications.

The applications with tight deadlines often handle only a small data volume that fits into main memory, especially with current trends in memory availability. In many of these applications a small volatile portion may have to be combined with larger volumes of static data. Long term permanence of the data is often restricted to whatever is needed for auditability. Predictability is very important for these applications.

Distribution is clearly needed in several applications, such as Air Traffic Control and mobile telephony and this will increase as we move into the realm of ubiquitous computing and spontaneous networking. The trend towards object oriented middleware platforms with real-time capabilities can be observed in some of the most demanding applications. Pertinent examples are Air Traffic Control and on-board navigation and control systems. The emerging platforms for spontaneous networking, such as Jini, are also built on the object paradigm. These systems will handle considerable amounts of data, although not all the data handled is structured. Multimedia data is usually streamed and the quality of service requirements are quite different from the transaction processing typically assumed by the active and real-time database community.

3. INTEGRATING THE KEY TECHNOLOGIES

From the previous section we must conclude that some combination of real-time, active, distributed and object-oriented functionality is required by a variety of applications. The question, therefore, is not whether we should combine them, but how to do it right. We should be concerned with the possible interference of the various features and ask ourselves:

- What are the key features in each base technology: real-time databases, active databases, and distributed OO-middleware?
- What subset of active features could be compatible with real-time requirements?
- How can predictability be ensured?
- How can active capabilities be provided in a distributed environment?
- What is the effect of the temporal fuzziness of distributed systems on active and real-time capabilities?
- How can active and real-time features be integrated into distributed object-oriented middleware?
- What are suitable correctness criteria for event composition and rule execution in distributed environments?

3.1 *Real-time database issues*

According to the definition of [Locke 1997] a real-time database is a data store whose operations execute with predictable response, and with

application-acceptable levels of external consistency, temporal consistency, logical consistency, permanence and atomicity. In this definition, two key requirements that set an RTDB apart from a non-RT database system are the temporal consistency and the predictable response. Temporal consistency as defined in [Ramamritham 1993, Purimetla et al. 1995] has two aspects. Global temporal consistency refers to the absolute age of the data and characterizes its staleness. In real-time environments the data quality decreases as time progresses between the time data was acquired, e.g. from a sensor, and the time it is consumed. Global temporal consistency is referred to as external consistency in [Locke 1997]. Mutual temporal consistency refers to the relative age of a set of data values, for example, data used in a calculation should come from the same sensor cycle or, alternatively, the user may specify an acceptable maximum age difference that can be tolerated by the application.

Predictability in real-time database systems can be analyzed by looking at the various contributions to the total execution time of a transaction [Buchmann et al. 1989]. These are the time required for performing the database operations once data is in the buffer, the I/O time, the time lost due to transaction interference, the time used by the non-database portion of application processing, and the communication time. To determine worst case execution times in a deterministic way, each of these contributing elements must have an upper bound.

An upper bound for database operations can be achieved by limiting the size of data, for example, by setting an upper bound for the number of tuples in a relation. This approach is commonly used in real-time environments since it is the basis also for the calculation of the application dependent (non-database) contribution. Determining an upper bound for I/O operations is difficult, if not impossible, for disk resident databases. If one assumes a worst case of one page fault per access, nothing will be scheduled due to the extremely pessimistic worst case execution time that will be dominated by the high disk access time. Preexecution, an approach proposed in [O´Neil et al. 1996], in which a transaction is executed once without acquiring locks, just to determine what tuples are needed and to set the buffer, followed by the real execution in a second step, just shifts the problem but doesn't guarantee end to end predictability. Any disk-based approach can at best provide statistical values for the expected execution time. In addition, many real-time applications require mean read and write latencies in the range of 0.05 ms, something that is not achievable with disk-resident data. Therefore, the only feasible solution for real-time systems that must give guarantees is avoiding I/O altogether by using main memory database systems. The drastic drop in memory costs and the availability of 64-bit address spaces make it possible to accommodate without problem the structured data required by typical real-time systems. For large-volume static and multimedia data that is streamed without concurrency control and transactional semantics a hybrid approach with large buffers for data staging is usually enough.

The time of interference is the time a transaction is either blocked waiting for resources held by another transaction or the time required for roll-back and restart due to a transaction abort, for example, because of deadlock. Conventional database systems use aggressive schedulers that acquire data resources dynamically and may hold these until committing. Conflicts between transactions are detected and resolved according to some resolution criterion. The reasons for doing so are simple and derived from the large disk access times: To avoid idling resources, such as the CPU, control passes to another transaction whenever an I/O operation is required. The overhead of passing control to another transaction is justified by the slow response of the disk. Therefore, a central objective of conventional DBMSs is to provide high intertransaction parallelism. To achieve high intertransaction parallelism it is necessary to lock as few data as possible, i.e., to provide small locking granules, for example, at the tuple level. Since the tuples that will be accessed by a transaction may depend on previous operations of the transaction, dynamic lock acquisition has become the method of choice to guarantee small locking units and thus high intertransaction parallelism. Dynamic lock acquisition results necessarily in scheduling policies based on conflict detection and conflict resolution. The bulk of the real-time database research has followed this approach because the assumption of the disk as the basic medium was never challenged. However, in main memory databases inter-transaction parallelism is less important since no long I/O waiting periods exist. Since the data will now be available in memory, no need for passing control to another transaction due to I/O is required and transactions will execute with fewer interruptions. Therefore, they can also lock larger units, for example, whole relations, which in turn makes it possible to determine the resources needed by a transaction ahead of time through simple syntactic analysis. This syntactic analysis can be done off-line for canned queries, a situation that is typical of real-time environments. Once the data resources are known, a whole new class of conflict avoiding schedulers becomes feasible [Ulusoy and Buchmann 1998]. The main advantage of these schedulers is, that the time of interference is eliminated since a trans-

action starts execution only when its resources are available. While this approach guarantees the execution time once a transaction is ready and scheduled for execution, the end-to-end execution time still depends on the waiting time in the queue.

The application-dependent portion of the worst case execution time can be determined with the help of tools by analyzing all the possible paths the application program may take.

The communication portion of the execution time depends on the degree of distribution of the system and the kind of network [Verissimo 1993]. There is a clear difference between a distributed system running in a LAN with a token ring network, in which an upper bound can be given, or a distributed system running over a wide area network or the Internet. It is also important to consider in this context the fuzziness derived from the lack of one central clock. This fuzziness will have an effect on all time stamps and everything dependent on them, from deadlines and temporal consistency to event composition and event consumption modes. Therefore, we will return to this problem later.

In addition to the predictability and concurrency control issues addressed above, real-time applications have special recovery needs. These are often less stringent than those typical of operational databases used in commercial applications. For example, since sensor data will arrive periodically, if a sensor reading is lost, the value can be captured in the next cycle. In general, no undo/redo is needed. Logging in real-time systems often has an archival function for long-term durability and auditability and less so for transaction roll-back or redo. In main memory databases alternate logging approaches based on messaging and asynchronous logging take the place of write-ahead logging assumed in conventional database systems. A summary of RTDBMS characteristics can be found in [Stankovic et al. 1999] and a good compendium of research results in [Bestavros et al. 1997, Bestavros and Fay-Wolfe 1997].

3.2 *Active database issues*

Active databases include Event-Condition-Action (ECA) rules as first class objects in the database [Dayal et al. 1988]. ECA rules have an explicit event part that determines, when a rule is to be executed, a condition that acts as a filter, and an action that may be any database-internal or external action. The events determine to a large extent the expressive power of an active database's rule mechanism. In their most general form they include database events, such as insert, delete, update; control events such as begin of transaction, commit, abort; temporal events that maybe absolute or relative, periodic or aperiodic; and user defined events. Primitive events may be combined through an event algebra that may include operators for sequence, disjunction, conjunction, negation, history, closure, etc. [Gehani et al. 1992, Gatziu and Dittrich 1993, Chakravarthy et al. 1994]. An event typically may trigger more than one rule, and an event may also participate in many different event compositions. This raises the question of event consumption, i.e., if a stream of events contains more than one event instance of a certain type that participates in a composite event definition, which event instance will be selected for the composition? This problem was solved for the centralized case in Snoop [Chakravarthy et al. 1994] through the definition of contexts. Contexts specify the consumption mode of events, i.e. they are policy definitions that define whether events should be consumed chronologically (chronicle), or whether the most recent instances should be used (recent). Besides these two obvious policies, two more have been defined. The cumulative policy specifies that all instances of an event that is part of a composite event will be accumulated until the composite event is fully composed. At that point all instances of the participating events will be removed. The continuous policy defines a sliding window, which is opened for each new primitive event arrival that initiates a new composition. The details of event composition and consumption are beyond the scope of this discussion, but we must observe the basic assumptions. All the event compositions and event consumption modes implicitly assume a central clock and a total order on events. These assumptions are quite reasonable for the centralized systems for which they were defined but cannot be sustained in a distributed environment. Again, we raise the issue here and defer the discussion.

ECA rules are triggered by events that may originate in a user transaction or they may be triggered by temporal or external events. Once a rule is triggered its condition should be evaluated and in case the condition is true, the action should be executed. The coupling mode determines how an ECA rule should be processed relative to the triggering transaction. This may be as a subtransaction of the user transaction either immediately following the event detection (immediate coupling mode) or it may be done at the end of the user transaction (deferred coupling mode). However, rules that are triggered by temporal events or by composite events that are based on primitive events that were generated in more than one user transaction cannot identify a single transaction to which they should be attached and must be executed in a separate transaction (detached cou-

pling mode). Since events that are generated in a transaction may require bindings that reflect the state of the database and/or the transaction at the instant the event occurred, the participation in an event composition and the triggering of a separate transaction may imply a relaxation of the isolation property. This fact has not received the necessary attention so far, even for centralized systems.

For a comprehensive compendium of relevant work on active databases the reader is referred to [Paton 1998].

3.3 *Distribution issues*

Distribution affects all aspects of an active, distributed, real-time system. The discussion in this section focuses on event processing. Event-based computing is emerging as the paradigm of choice for composing applications in open distributed environments.

One of the core problems of distributed systems is the synchronization of processes running on physically and logically distributed nodes with communications that may exhibit unpredictable delays and without a central clock. The lack of a central clock implies that temporal order cannot unequivocally be established. Time, as provided by a distributed time service is imprecise with respect to clock readings at different nodes and inaccurate with respect to physical time.

Inaccuracies with respect to time have a major impact on timestamping which in turn is the basis for the determination of temporal consistency, both global and relative, the composition of events, and the consumption of events. The fundamental problem, then, is the characterization of the quality of the available time service and the proper consideration of its limitations.

A variety of approximations for modeling the time imprecision in distributed systems have been proposed. A thorough discussion of this issue is beyond the scope of this paper. Therefore, we will only address three approaches that have been adopted in the literature for use in distributed and active systems. For an excellent review of these issues the reader is referred to [Kopetz 1997].

Logical (Lamport) clocks [Lamport 1978] and vector time [Schwarz and Mattern 1994] make it possible to establish a partial order based on causality. Unfortunately, they are not appropriate for the open systems with external inputs that are the subject of this discussion.

When a sparse time base is assumed, the points at which events can be generated are discretized and predetermined. Only if events are at least two time granules apart, the sequence of these events can be determined unequivocally. This is known as the 2g precedence model [Kopetz 1992]. In the 2g precedence model an upper bound to the precision is assumed and a virtual clock granularity with granularity g is defined. The 2g precedence model is very useful for dealing with embedded systems and other small and tightly controlled environments. However, since the granularity depends on the assumed precision, it is not a feasible approach for wide area networks and open distributed systems.

The Network Time Protocol (NTP) offers a standardized time service with a reliable error bound. NTP is based on the notion of strata that can guarantee the accuracy within an accuracy interval. By using NTP and injecting an external reference time, e.g. GPS time, it becomes possible to provide timestamping with accuracy intervals and partial ordering of events in large scale distributed systems [Liebig, Cilia and Buchmann 1999].

3.4 *Cross-effects*

Even a condensed discussion of the base technologies illustrates the magnitude of the problems. If two or more base technologies are combined, a whole new set of problems must be considered.

The time notion in active databases is quite different from the time notion in real-time systems. Temporal events in active databases determine when a rule is fired but no information on execution time or deadlines is provided. Real-time systems are primarily concerned with execution time and meeting deadlines, and the temporal consistency of the data.

Real-time requirements impose serious limitations on the active capabilities that can be provided. The first and most obvious is the triggering of new rules. Active databases in general do not limit the triggering of rules by the action part of another rule. Especially the object-oriented aDBMSs may, by the very nature of object-oriented systems, execute any method of arbitrary complexity in the action part. Emphasis in the active database community has been placed on ensuring termination, i.e., the avoidance of cycles. However, if timing constraints must be obeyed, the size of a task may not dynamically expand. The strongest limitation on the execution model of a real-time active database system consists in disallowing immediate and deferred coupling modes, thus allowing only detached execution of triggered transactions. Given the possible violation of the isolation property because of parameter transfer, this approach may not be acceptable. Furthermore, if a rule is triggered by a transaction that is aborted, the atomicity property is violated since a detached transaction is an independent transaction

that cannot be rolled back. Therefore, sequential causally dependent detached transactions must be used. An alternative consists in limiting the depth of triggering, for example, to one rule. From the point of view of determining the execution time of such a transaction this is equivalent to the evaluation of a conditional statement. Limiting the depth of triggering raises, however, a serious problem, since the action part of the rules now must be strictly controlled not to produce any legal event, or we are faced with the problem of distinguishing between triggering and non-triggering events of the same type. Therefore, in a real-time active DBMS, the action portion of rules and the acceptable event set must be carefully matched and possibly curtailed. Before discussing further restrictions on the event set, we must analyze the effect of distribution on event composition.

Event composition in its general form depends on the ability to determine the sequence of occurrence of events. This is important not only for operators, such as, sequence, but also for all other operators since the consumption of events directly depends on it. For example, in a distributed system it becomes difficult to determine, whether an event generated at node N1 and detected at node N2, really should be consumed in a chronological consumption policy, or if there is another (older) instance of that event type generated at node N3 that was delayed in the network. The assumption of a 2g precedence clock synchronization model alone does not solve these problems. Particular care must be exercised when specifying a 2g precedence model to keep in mind the underlying assumption of sparse time and a guaranteed upper bound to the precission. If this is not the case, either because dense time is suddenly assumed or because Internet-based distribution is expected, the results will be wrong. Another weakness of applying the 2g precedence model to event composition that is frequently swept under the rug is the implicit treatment of the ambiguities. If two events are not distinguishable because they are not at least 2g apart, they are considered as concurrent, and the event consumption is ambiguous. This ambiguity must either be resolved explicitly through application semantics or it must be made clear to the user that an exception exists. The latter approach must necessarily be taken when developing a generic middleware platform. All the operators that have been defined for the event algebras of centralized active databases must be carefully reexamined and restated with their limitations being made explicit.

Whenever an event is detected, there is an inherent detection delay between the time the event occurred and the time it is detected. The detection delay may depend on system load and may impact the temporal consistency of the data. This problem was identified in [Branding and Buchmann 1995], where as a first pragmatic approach the complexity of events was drastically curtailed and special high-priority events where proposed for overload situations in real-time active databases. The detection delay is magnified in distributed environments. In [Liebig, Cilia and Buchmann 1999] the detection delay is explicitly taken into account for the case of event detection and composition in distributed active systems. Event composition in a distributed environment under real-time constraints becomes very difficult. Sparse time must be assumed and transmission delays must have an upper bound. This limits the feasible environments to logically distributed systems on single nodes or very tightly controlled specialized LANs. Unless this is the case, only a statistical approach to real-time is possible. Given the temporal fuzziness introduced through the distribution, one would be well advised to avoid event composition in a distributed real-time environment or at least to keep it to a minimum, since the resolution of ambiguities to guarantee correctness is a time-consuming process that real-time applications with tight deadlines may not be able to afford.

Event composition in an open distributed system presents additional problems. In a centralized system every producer of a certain event is known. In an open distributed environment this is not necessarily the case. This means that group communication concepts and mechanisms must be employed to ensure that all potential producers or consumers of an event are encounted for. In the case that event composition is itself distributed over several nodes, the result of event composition is influenced by transmission delays and communication failures. Therefore a correctness criterion for distributed composition of events is needed and a real-time cognizant approach to enforce such criteria must be developed. This problem is even more complicated, when implicit or explicit replication of composite event detection is introduced.

Finally, the crosseffect resulting from rule execution in distributed environments is addressed. In [Ceri and Widom 1992] various paradigms for distributed rule processing are identified and characterized on the basis of whether multisite rules are permitted or not, whether rules may execute before the transaction is done at that site, and whether intersite priorities exist or not. Even in the tightly integrated relational DBMS with a rather restrictive execution model that was analyzed in [Ceri and Widom 1992], distributed rule processing is quite complex and requires locking and coordination among sites to guarantee the same semantics as centralized rule execution. At present, research efforts in distributed active

databases are focused on the problems of event handling and composition. A serious discussion of the semantics of rule processing in open distributed systems under timing constraints has not even begun.

4. CONCLUSIONS

An analysis of a set of applications and the base technologies together with the possible cross-effects has shown the futility of attempting to combine the full spectrum of functionality of active databases, real-time systems, and distributed object systems in a single platform. However, some useful subsets, each challenging in its requirements but feasible, can be identified:

- Main-memory databases where distribution can be limited to mirroring for reliability and recoverability with limited active capabilities to provide timing guarantees in real-time environments with tight timing constraints.
- Centralized active databases with high query to update ratios, very high data volumes, and tight but soft deadlines.
- Distributed object platforms with enforcement of timing constraints in tightly controlled distributed environments with little database functionality and limited event sets.
- Time-constrained distributed systems with rather lax but not necessarily soft timing requirements, wide area distribution, and possibly large data volumes.

The biggest challenge will be to move away from application-specific solutions and to provide application-independent platforms for which the underlying assumptions and the resulting semantics are clearly stated and easily understood by the developers of the application systems. To accomplish this we must move beyond a superficial combination of base technologies, and analyze their interactions carefully. As we understand these interactions better we may be able to move the boundaries between the subsets identified above towards the goal of a more generic distributed object platform that offers active and real-time functionality.

5. ACKNOWLEDGEMENTS

The authors wish to thank Mariano Cilia for many interesting discussions.

6. REFERENCES

Arnold, O´Sullivan, Scheifler, Waldo and Wollrath: 1999, *The Jini Specification*, Addison Wesley.

Bestavros, A. and Fay-Wolfe, V. (eds): 1997, *Real-Time Database and Information Systems - Research Advances*, Kluwer Academic.

Bestavros, A., Lin, K.-J. and Son, S. (eds): 1997, *Real-Time Database Systems - Issues and Applications*, Kluwer Academic.

Branding, H. and Buchmann, A.: 1995, On providing soft and hard real-time capabilities in an active dbms, *in* M. Berndtsson and J. Hansson (eds), *Active and Real-Time Database Systems (ARTDB-95)*, Springer, pp. 158–169.

Buchmann, A., Dayal, U., McCarthy, D. and Hsu, M.: 1989, Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency control, *Proceedings Fifth International Conference on Data Engineering*, IEEE, Los Angeles.

Ceri, S. and Widom, J.: 1992, Production rules in parallel and distributed database environments, *Proceedings of the 18th VLDB Conference*, Vancouver, Canada, pp. 339–351.

Chakravarthy, S., Krishnaprasad, V., Anwar, E. and Kim, S.: 1994, Composite Events for Active Databases: Semantics, Contexts and Detection, *Proc. VLDB´94*, Santiago, Chile, pp. 606–617.

Datta, A.: 1999, Position statement on artdbs, ARTDB-99. Panel Discussion.

Dayal, U., Buchmann, A. and McCarthy, D.: 1988, Rules are objects too: a knowledge model for active, object-oriented database systems, *Proceedings of the 2nd International Workshop on Object-Oriented Database Systems*, LNCS 334, Springer.

Gatziu, S. and Dittrich, K.: 1993, Events in an active object-oriented database system, *Proceedings of Rules in Database Systems*, Edinburgh, pp. 23–39.

Gehani, N., Jagadish, H. and Shmueli, O.: 1992, Event specification in an active object-oriented database, *Proceedings of the International Conference on Management of Data (SIGMOD '92)*.

Harrison, T., O´Ryan, C., Levine, D. and Schmidt, D.: 1998, The design and performance of a real-time corba event service. Submitted to IEEE Journal on Selected Areas in Communications.

Kopetz, H.: 1992, Sparse time versus dense time in distributed real-time systems, *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yakohama, Japan, pp. 460–467.

Kopetz, H.: 1997, *Real-Time Systems - Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers.

Lamport, L.: 1978, Time, clocks and the ordering of events in a distributed system, *CACM* **21**(7), 558–565.

Liebig, C., Boesling, B. and Buchmann, A.: 1999, A notification service for next generation IT systems in air traffic control, *GI Workshop Multicast - Protokolle und Anwendungen*, Braunschweig.

Liebig, C., Cilia, M. and Buchmann, A.: 1999, Event composition in time-dependent distributed systems, *International Conference on Cooperative Information Systems COOPIS99*, Edinburgh.

Locke, D.: 1997, Real-time databases: Real-world requirements, *in* A. Bestavros, K. Lin and S. Song (eds), *Real-Time Database Systems - Issues and Applications*, Kluwer Academic Publishers.

Object Management Group (OMG): 1999, Real-time corba architecture, *Technical Report ptc-99-06-02*, OMG, Famingham, MA .

O´Neil, P., Ramamritham, K. and Pu, C.: 1996, A two-phase approach to predictably scheduling real-time transactions, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, Prentice-Hall, pp. 494–522.

Paton, N. (ed.): 1998, *Active Rules in Database Systems*, Springer-Verlag (New York).

Purimetla, B., Sivasankaran, R., Ramamritham, K. and Stankovic, J.: 1995, Real-time databases: Issues and applications, *in* S. Son (ed.), *Advances in Real-Time Systems*, Prentice Hall.

Raatikainen, K.: 1997, Real-time databases in telecommunications, *in* A. Bestavros, K. Lin and S. Son (eds), *Real-Time Database Systems - Issues and Applications*, Kluwer Academic Publishers.

Ramamritham, K.: 1993, Real-time databases, *International Journal of Distributed and Parallel Databases* **1**(2).

Schwarz, R. and Mattern, F.: 1994, Detecting causal relationship in distributed computations: In search of the holy grail, *Distributed Computing* **7**(3), 149–174.

Stankovic, J., Son, S. and Hansson, J.: 1999, Misconceptions about real-time databases, *IEEE Computer* **32**(6), 29–36.

Ulusoy, O. and Buchmann, A.: 1998, A real-time concurrency control protocol for Main Memory database systems, *Information Systems* **23**(2), 109–125.

Verissimo, P.: 1993, Real-time communication, *in* S. Mullender (ed.), *Distributed Systems*, 2 edn, Addison-Wesley.