The Design of Real-Time Extensions To The Open Object-Oriented Database System*

V. F. Wolfe, L. C. DiPippo,
J.J. Prichard, and J. Peckham
Computer Science Department
University of Rhode Island
LastName@cs.uri.edu

P. J. Fortier
Dept. of Computer Engineering
U. Of Massachusetts at Dartmouth
pfortier@umassd.edu

Abstract

This paper describes real-time extensions to the Open Object-Oriented Database system using the RT-SORAC data model. This model combines an object-oriented data model, real-time requirements, flexible transactions, semantic relationships among objects, and active database features. Several extensions to the Open Object-Oriented Database system, including development of interfaces for real-time objects and real-time transactions, use of a real-time operating system, incorporation of real-time object management, and incorporation of real-time transaction management, are also described.

1 Introduction

Many applications such as military command and control, multi-media, and medical patient monitoring, need support for large volumes of complex data that must be manipulated under timing constraints. For instance, a submarine's command and control system relies on complex data representing the position of external contacts that are being tracked. Some capabilities required by such applications can be found in *database management* systems that provide structured support for large volumes of persistent and shared data. Further support is provided by *real-time database management* systems [1], which extend traditional database management systems with *time-constrained* transactions, such as transactions that must retrieve the position of each enemy submarine in a combat scenario. Real-time databases also add support for managing *time-constrained data* that is only valid for a certain interval of time, such as data representing the "current" position of each submarine being tracked. However, the vast majority of real-time database management system research and design has been based on the relational data model, which only supports simple types and is therefore not suitable for complex data types found in applications such as command and control and multi-media.

Fortunately, there has also been a significant amount of research done to develop *object-oriented databases*. These databases possess rich typing capabilities that can potentially support management of complex data types. The *Open Object-Oriented Database* (Open OODB) system is a recent project initiated by the U.S. Advanced Research Projects Agency (ARPA) that is under development by Texas Instruments and several other research institutions. The project's goal is to establish a common, modular, modifiable, object-oriented database system suitable to be used by a wide range of researchers and developers [2]. Open OODB is designed so that features such as transaction management, query interface, persistence, etc. are modules that can be individually "unplugged" and replaced by other modules. We describe the architecture and features of Open OODB in Section 2. Unfortunately, Open OODB has failed to provide support for real-time database management.

The University of Rhode Island is an alpha site for the current release of Open OODB. In this capacity, we have designed real-time extensions to Open OODB. We have taken advantage of its open, modular structure to modify and add modules to incorporate support required by real-time applications. We have implemented a *real-time object policy manager* that performs a new form of semantic object-based real-time concurrency control [3]. We describe the design

* This material is based upon work supported by the U.S. Naval Undersea Warfare Center and the U.S. National Science Foundation under grant IRI-9308517.

of this part of the system, and other modifications, in Section 3. Section 4 summarizes our experiences with synthesizing real-time and object-oriented technology in database systems.

2 The Open Object-Oriented Database System

The Open OODB project seeks to provide an open, modular framework for common object-oriented database development. It has a well developed set of requirements including the use of the object-oriented data model, a full range of typical database requirements, distribution, change management (allowing replication), openness, seamlessness, performance, and industrial strength [2]. An alpha version that meets some of these requirements has been released. Many universities and companies are participating by developing modules to satisfy unmet requirements.

Open OODB's computational model strives to transparently extend the behavior of objects that are found in application programming languages [2]. To accomplish this goal, Open OODB wishes to avoid making programmers use embedded system calls. Instead, the computational model in the current alpha release is a transparent extension to C++. In the model, objects can exist in one of many address spaces. Currently there are two address spaces supported: transient, which resides in main memory, and persistent, which resides remotely in the Exodus [4] storage manager. Open OODB provides communication and translation facilities to allow transfers between different address spaces. There are extensions to C++ that specify this and other database functionality.

The basic conceptual system architecture of Open OODB is shown in Figure 1 (along with the proposed real-time extensions that we have added as indicated). The *support managers* are modules that are currently implemented as library routines which get linked into the user's C++ program to (transparently) provide the extended database capabilities. The *Address Space Manager* supports mappings between global identifiers and object identifiers used in the local address space. The *Communication Manager* provides support for interfacing to one or more underlying communications mechanisms. The *Translation Manager* translates an object stored in one format to a target format. For instance, it translates objects stored in Exodus into objects suitable for a C++ application program. The *Data Dictionary* is a globally known repository of the data model and type information, instance information, name mappings (of application names to instances) and possibly system configuration and resource utilization information.

Policy managers (PMs) provide extenders to the behavior of programs by coordinating the support managers just described. The *Persistence Policy Manager* provides applications with an interface through which they can create, access, and manipulate persistent objects in various address spaces. The *Transaction Policy Manager* enables concurrent access to persistent and transient data; its implementation in the current alpha release is a trivial mapping to Exodus write locks on all objects. Other policy managers include those for distribution, change management, indexing, and query processing.

The query interface is in two forms: an extended version of C++ and an SQL-like language called OQL, which must be embedded in C++ code [2]. The C++ interface is C++ code extended with methods that invoke capabilities of the managers. OQL has a very basic set of SQL-like commands that work on sets of objects. Although the current version is skeletal, actual examples can be executed.

The current alpha release contains partial implementations of many of the managers. It relies heavily on Exodus as it persistent storage and for concurrency control and recovery. Eventually, as Open OODB development progresses, many of the manager capabilities will be incorporated into the Open OODB architecture.

3 Real-Time Extensions to Open OODB

The current version of Open OODB does not support real-time data management requirements. However, its open, modular design and use of the object-oriented paradigm facilitate real-time extensions. As a basis for real-time extensions, we have designed a model of a real-time object-oriented database, called the *RTSORAC* model [1] [5], which synthesizes aspects of real-time databases, object-oriented databases, semantic databases, and active databases. We have incorporated the RTSORAC model into the architecture of Open OODB.

In this section we first summarize the RTSORAC model and then describe our extensions to Open OODB. These extensions include porting Open OODB to an operating system that is consistent with

[1] RTSORAC = **R**eal-**T**ime **S**emantic **O**bjects **R**elationships **A**nd **C**onstraints.

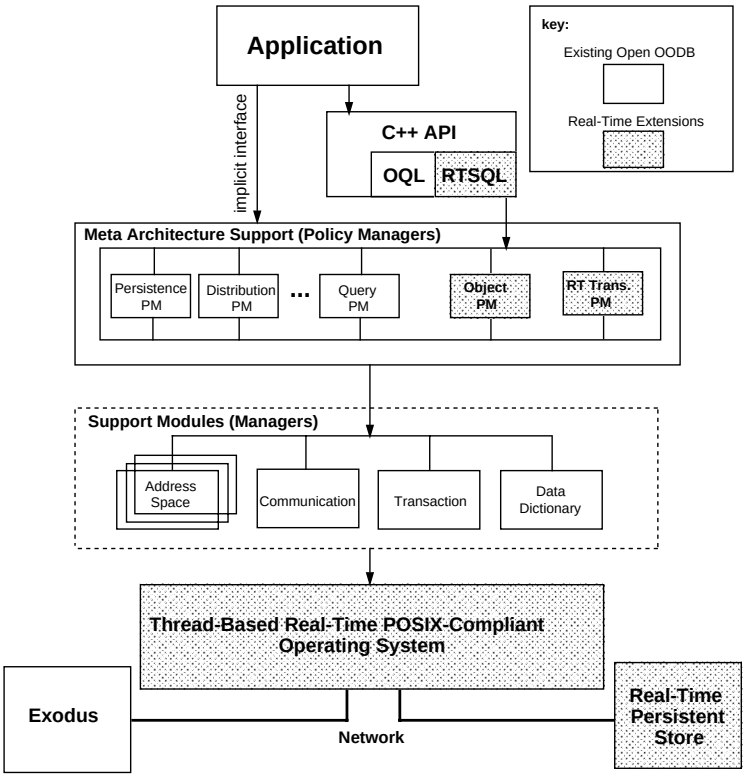

Figure 1: Open OODB Architecture With Real-Time Extensions

the real-time POSIX standards [10], incorporating a *real-time persistent address space* (as an alternative to Exodus as the only persistent address space), adding several policy managers, and developing a real-time query language interface. We summarize each of these extensions and then describe the design of a real-time *Object Policy Manager* (OPM) in detail.

3.1 The RTSORAC Object Model

RTSORAC has three components which model the properties of a real-time object-oriented database: *objects*, *relationships* and *transactions*. *Objects* represent database entities. *Relationships* represent associations among the database objects. *Transactions* are time constrained executable entities that access the objects and relationships in the database. We summarize the RTSORAC object model here; a complete description of all model components can be found in [5].

An *object* consists of five components, $\langle N, A, M, C, CF \rangle$, where N is a unique name or identifier, A is a set of attributes, M is a set of methods, C is a set of constraints, and CF is a compatibility function. Attributes are similar to those found in most object models in that they have a name and a value field. In the RTSORAC model, they have additional fields for storing a timestamp value, and an imprecision value. The timestamp value is necessary for determining the temporal consistency[2] of the attribute. The imprecision field is used to accumulate how far "off" the actual value might be, due to non-serializable concurrent access of the object, as described in Section 3.2.3.

Methods are also similar to those found in most object models in that they have a name, a set of arguments, and a set of operations to be performed. The set of arguments are of the same form as the attributes, and hence have additional fields for a timestamp value and an imprecision value. The RTSORAC model defines two additional components for methods: a set of exceptions that may be raised by the method to signal that the method has terminated abnormally, and a set of operation constraints. Operation constraints can be used to express precedence constraints, execution constraints, and timing constraints [6]. An operation constraint also includes an action to be taken if the constraint is violated.

[2]Data temporal consistency constraints specify the time interval of data validity

The next component of an object is a is a set of data constraints which permit the specification of correct object state. Each data constraint consists of a name, a set of attributes from the object, a predicate, and an enforcement rule. The predicate is a boolean expression that is specified using attributes from the attribute set. The predicate can be used to express the logical and temporal consistency requirements of the data stored in the object by referring to the value, time, and imprecision fields of the attributes in the set. The enforcement rule is executed when the predicate evaluates to false, and is similar to a method with a null argument list, and with a name which can be derived from the name of the constraint. Thus, an enforcement rule consists of a set of operations, a set of exceptions that may be signaled, and a set of operation constraints. The combination of the predicate and the enforcement rule in constraints can be used to specify a trigger, a feature proposed for constraint maintenance in databases.

The last component of an object is a compatibility function that expresses, for each ordered pair of methods, under what conditions the methods can execute concurrently while preserving system requirements. These conditions can include timing consideration, other currently active methods, affected sets of methods, and method arguments [3]. The use of the compatibility function in real-time concurrency control is discussed in Section 3.2.3.

3.2 Real-Time Extensions

Our real-time extensions to Open OODB affect its interface, basic underlying architecture, and its policy managers. These extensions are designed within Open OODB's original framework, and are shown as shaded components in Figure 1.

3.2.1 Extensions to the Open OODB Interface

In the alpha release of Open OODB, the schema is specified as a collection of C++ classes and transactions are specified as C++ programs, or as OQL programs that are compiled to C++ programs. Recall that objects and transactions in the RTSORAC model have additional features beyond those supplied by C++ classes and programs. To support these additional capabilities, we have added two additional interfaces to Open OODB: a graphic interface to specify classes for RTSORAC objects, and real-time extensions to the standard SQL query language to specify RTSORAC transactions.

Schema Specification. A schema in our extended Open OODB prototype is specified as classes for RTSORAC objects. The interface translates these classes to C++ code suitable to execute on the extended Open OODB system. Specification of these classes is done with a graphic interface programmed with X-windows and Motif. This interface produces a C++ class specification with certain "meta members", including a wait queue, compatibility function, POSIX mutual exclusion locks (mutexes) and condition variables, and member functions to lock and release the object. These meta members are used by the concurrency control mechanism described later in Section 3.2.2. The compatibility function defined by the interface tool is structured with a case for each pairwise combination of the class's methods. Each case is boolean expression involving the components of the compatibility function (Section 3.1). The case determines whether the methods may execute concurrently.

Transaction Specification. RTSORAC transactions are specified using extensions for real-time transactions that we have designed for the standard SQL query language. The extended language is called SQL/RT [7, 8]. The SQL/RT transaction structure is designed to allow decomposition of transactions into subtransactions that support more complex commit and abort semantics. SQL/RT provides three constructs for transactions that can be used to support the RTSORAC transaction model: one for defining transactions, a second for modifying characteristics of previously defined transactions, and a third for initiating transactions. Each of these constructs is discussed below.

3.2.2 Underlying Architecture Modifications

As shown in Figure 1, our prototyping uses a real-time operating system that is consistent with the POSIX standards and a real-time persistent storage manager. Both of these are changes to the original Open OODB underlying architecture.

Real-Time Operating System. The alpha release of Open OODB executes on a Sun Sparc architecture with the SunOS Unix operating system. Unfortunately, Unix has many well-known deficiencies for supporting real-time applications[9]. Fortunately, the next release of Open OODB executes on the Solaris 2.3 operating system which contains many of the real-time operating system features specified in the IEEE/ISO POSIX real-time operating system standards [10].

These features include shared memory, priority-based scheduling and priority-based semaphores. Our approach has been to implement Open OODB modules with real-time capabilities on a Solaris 2.3 operating system while maintaining an interface to the existing modules of the alpha release of Open OODB. Integration of our real-time modules with Open OODB will occur once the Solaris 2.3 version of Open OODB is available.

Real-time Persistent Store. Our second basic architecture modification is the addition of a real-time persistent store. The current Open OODB alpha version relies heavily on the Exodus storage manager as its persistent store. Exodus's unpredictable execution times, handling of requests in first-come-first-serve order rather than priority order, and conservative locking capabilities, render it unacceptable for a real-time data management system. Instead of relying on Exodus, we are incorporating another address space to Open OODB: a real-time persistent address space. Our current design uses this address space as checkpointed permanent storage for shared main memory RTSORAC objects (see Section 3.2.3) and for swap space if all objects can not fit into shared memory. This capability relies on the Zip Real-Time Database Management System (RTDBMS) from DBx Inc, which provides predictable execution times and bounded resource utilizations. In a joint effort with DBx Inc we have ported Zip RTDBMS to Solaris 2.3 operating system for use in our Open OODB prototype.

3.2.3 Object Management

RTSORAC database objects are designed to be kept in shared main memory for fast, predictable access. That is, instead of keeping objects in one of the current Open OODB address spaces, where they must be copied into a transaction's local address space for use, we keep objects in shared main memory. The *Object Policy Manager* (OPM) that we have added to Open OODB manages this shared memory and provides concurrency control for the objects. Figure 2 shows the implementation of object management.

Shared Main Memory Management. In our extended system, an *object keeper* process creates a shared main memory segment at system startup. This keeper process may load the shared segment with object instances, either by restoring old (persistent) objects, or by instantiating new objects. Transaction processes map the shared segment into their own virtual address spaces (see Figure 2), thereby gaining direct access to object instances. Transactions use an overloaded C++ `new` operator operator to dynamically place objects in the shared segment or to locate existing objects by name. To do this, part of the shared segment is reserved at a well-known offset for use by the system as an *object table*. The table associates each object's name with the object's offset from the shared segment's base address. The table also stores object type information. The special `new` operator automatically manages the object table and uses it to translate object names to offsets. From this offset, the `new` operator creates a properly typed pointer to the object in the shared memory segment and returns this pointer to the transaction. There is also an overloaded C++ `delete` operator for removing objects.

Semantic Locking Object Concurrency Control. Since each transaction may concurrently map objects in the shared memory segment into its own virtual address space, we must provide a concurrency control mechanism. Open OODB's current policy enforces serializability through exclusive locking of objects by transactions before a transaction makes a copy in it's own address space. Such techniques are inflexible and ignore transaction and data timing constraints.

We have developed a concurrency control technique called *semantic locking* for RTSORAC object management [3]. The semantic locking technique is capable of supporting logical consistency, temporal consistency, and the trade-offs between them as well as bounding any resulting imprecision. The technique utilizes the user-defined compatibility function (Section 3.1) of a RTSORAC object to determine the tradeoff and to define correctness for that particular object. In this technique, a transaction requests a semantic lock to invoke a method on an object. Semantic locks are granted based on the evaluation of a set of conditions and on the evaluation the compatibility function of the object.

When a transaction requests a semantic lock for a method invocation, it calls the meta member function `SLM_lock()` of the object specifying the method and the arguments for the requested invocation. The meta member function acquires the POSIX mutex for access to the object's meta data. When the mutex is granted, the `SLM_lock` meta member function attempts to acquire a semantic lock for the transaction. There are two possible outcomes when a transaction

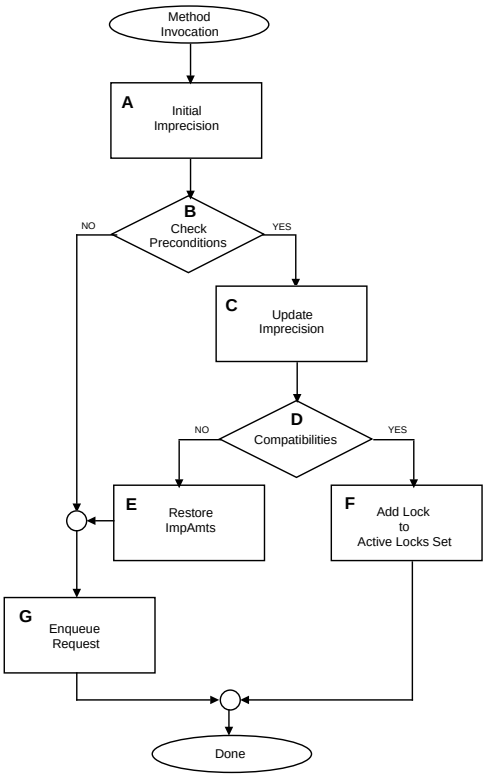

Figure 3: SLM_lock Meta Member Function Outline

process requests a semantic lock for a method invocation: the `SLM_lock` meta member function either grants permission to the transaction process to execute the requested method, or it suspends the requesting transaction. A suspended transaction will be awakened and will retry its lock request whenever a lock is released (discussed later). In either case, the transaction releases the mutex at the end of the `SLM_lock` meta member function. Note that the OPM uses mutexes to ensure mutual exclusion only for each object's meta members during the semantic locking mechanism execution; transaction access to object attributes is controlled with semantic locks.

Figure 3 shows the semantic locking mechanism that the `SLM_lock` meta member function performs when a transaction requests a semantic lock for a method invocation m_{req}. First, `SLM_lock` computes the maximum amount of imprecision that m_{req} could introduce into each of the attributes that it writes and into each of its own return arguments (Step A). It computes these values by using the amount of imprecision already in the attribute or return argument and calculating how m_{req} may update this imprecision through operations that it performs.

Next, the meta member function evaluates a set of conditional statements that determine if granting the lock would violate temporal or imprecision constraints. The first condition ensures that if a transaction requires temporally valid data, then m_{req} will not execute if any of the data that it reads will become temporally invalid during its execution time. The other two conditions test that m_{req} will not introduce too much imprecision into the attributes that it writes and into its return arguments.

If all of the above conditions hold, the `SLM_lock` meta member function updates the imprecision amounts computed in Step A and saves the old amounts in a data structure, in case the request is not granted (Step C). The meta member function then loops to evaluate the compatibility function for m_{req} with each currently locked method invocation and with each lock request in the wait queue for a method invocation with higher priority than m_{req} (Step D). If all tests in the loop succeed, the meta member function grants the lock for m_{req}, adds it to the active locks set and gives the transaction permission to execute the method. If any of the conditions or any compatibility test fails, the `SLM_lock` meta member function restores the original values of any changed imprecision amounts (Step E), places the lock request in the priority queue, and suspends the requesting transaction (Step G).

A transaction must explicitly release the locks that it is granted by calling the `SLM_release` meta member function on the object. This meta member function removes the method invocation from the object's active locks set. It then broadcasts on a real-time POSIX condition variable to awaken all of the suspended transactions in the object so they may retry their lock requests. Due to the newly-released lock, it may now be possible to grant some of these previously-denied locks. The use of a the real-time POSIX scheduler, discussed next, assures that the awakened transactions make their lock requests in priority order.

3.2.4 Transaction Management

Our Open OODB Transaction Policy Manager (TPM) provides for real-time scheduling of transaction processes, transaction timing constraint enforcement, and for maintenance of information about transaction processes.

The real-time transaction scheduling performed by the TPM is essentially a mapping of timing constraints expressed in RTSORAC transactions into real-time POSIX priorities for transaction processes. This mapping is designed so that the transaction process priorities realize Earliest-Deadline-First (EDF) schedul-

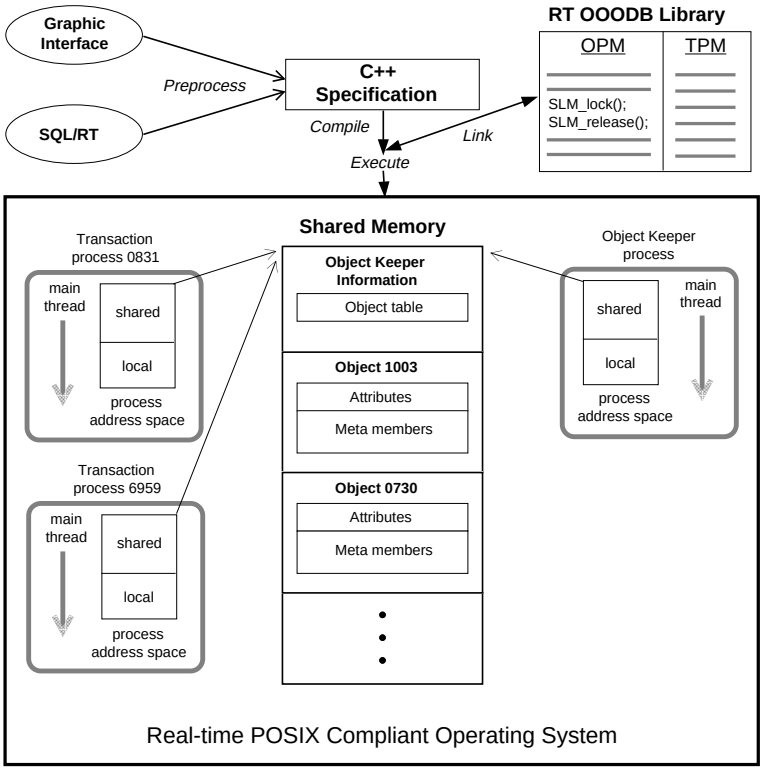

Figure 2: Object Management Implementation in Open OODB

ing. EDF scheduling has been shown to be effective in real-time databases [11], but implementing EDF scheduling using the capabilities specified by the real-time POSIX interface is non-trivial. The problem is that optimal EDF scheduling requires infinite priorities (one for each possible deadline), while POSIX mandates a minimum of only 32 priorities. Furthermore, real-time POSIX mandates a form of First-In-First-Out (FIFO) scheduling for processes of the same priority[3]. FIFO scheduling can adversely affect EDF scheduling since a later deadline may execute before an earlier deadline within same priority. Our TPM is designed to minimize the violation of EDF transaction scheduling order while using the capabilities of real-time POSIX. It does this by first mapping the deadline to a priority, then shuffling processes within the same priority to EDF order, and finally by increasing process priorities as their deadlines near [12].

In addition to EDF scheduling on the processor, the TPM is also responsible for mapping RTSORAC transaction timing constraints to real-time POSIX primitives for enforcing timing constraints. In particular, the TPM maps expressed earliest start times, deadlines, and periods into appropriate POSIX timer primitives.

[3]There are two other POSIX policies: *round robin* which is FIFO with a time quantum, and *other*, which is non-standard.

4 Summary of Real-Time and Object-Oriented Synthesis

This paper has shown, through the actual extension of an object-oriented system, how the object-oriented paradigm and the requirements of real-time systems can be combined to support data-intensive real-time applications. The object-oriented features are provided by the basic Open OODB system, which extends C++ with database capabilities so that objects can be persistent and concurrently shared.

The real-time features are provided by our enhancements to Open OODB. Our port of Open OODB to a real-time operating system allows for preemptive priority scheduling based on timing constraints of transactions. The new operating system also provides the ability to keep objects in shared memory for fast, predictable access. Our incorporation of a real-time persistent data store into Open OODB provides predictable storage and retrieval times for persistent

objects. Our extensions to SQL as the data definition and data manipulation language provide capabilities to express transaction decomposition, transaction timing constraints, and object temporal consistency constraints, among other things. Our Object Policy Manager provides the potential for increased data availability through the use of semantic concurrency control on the finer grained level of object methods, instead of on entire objects. This concurrency control technique is capable of expressing and enforcing the trade-off between temporal consistency of object data and traditional logical consistency of the data.

References

[1] P. S. Yu, K.-L. Wu, K.-J. Lin, and S. H. Son, "On real-time databases: Concurrency control and scheduling," *Proceedings of the IEEE*, vol. 82, pp. 140–157, January 1994.

[2] D. L. Wells, J. A. Blakely, and C. W. Thompson, "Architecture of an open object-oriented database management system," *IEEE Computer*, vol. 25, pp. 74–82, October 1992.

[3] L. B. C. DiPippo and V. F. Wolfe, "Object-based semantic real-time concurrency control," in *Proceedings of IEEE Real-Time Systems Symposium*, December 1993.

[4] M. Carey, D. J. DeWitt, J. E. Richardson, and E. J. Shekita, *Object-Oriented Concepts, Databases and Applications*. Addison-Wesley Publishing Company, 1989.

[5] J. Prichard, L. C. DiPippo, J. Peckham, and V. F. Wolfe, "RTSORAC: A real-time object-oriented database model," in *Proceedings of the International Conference on Database and Expert Systems Applications*, Sept 1994.

[6] V. Wolfe, S. Davidson, and I. Lee, "*RTC*: Language support for real-time concurrency," *Real-Time Systems*, vol. 5, pp. 63–87, March 1993.

[7] P. Fortier, V. F. Wolfe, and J. Prichard, "Flexible real-time SQL transactions," in *IEEE Real-Time Systems Symposium (to appear)*, Dec. 1994.

[8] P. Fortier, J. Prichard, and V. F. Wolfe, "SQL/RT: Real-time database extensions to the sql standard," 1994. To appear in *Standards and Interface Journal*.

[9] B. Gallmestier and C. Lanier, "Early experience with POSIX 1003.4 and POSIX 1003.4a," in *IEEE Real-Time Systems Symposium*, Dec. 1991.

[10] IEEE, *Portable Operating System Interface (POSIX); Part 1: System API; Amendment 1: Real-time Extension*. IEEE, 1994.

[11] R. Abbott and H. Garcia-Molina, "Scheduling real-time transactions: A performance evaluation," in *14th VLDB Conference*, Aug. 1988.

[12] J. Senerchia, "A dynamic real-time scheduler for POSIX 1003.4a compliant operating systems," 1993. Master's Thesis. Dept. of Computer Science, The University of Rhode Island.