Making an Object-Oriented DBMS Active: Design, Implementation, and Evaluation of a Prototype†

Sharma Chakravarthy
Computer and Information Science
department and
Database Systems Research and Development Center
University of Florida, Gainesville FL 32611
Email: sharma@ufl.edu

Susan Nesson
Lotus Development Corporation
55 Cambridge Parkway
Cambridge, MA 02142.

Abstract

Extant databases are passive in nature and offer little or no support for automatically monitoring conditions defined over the state of the database. In fact such a capability is central for a variety of applications requiring timely and time-constrained data management and processing (e.g., cooperative processing, process control, air traffic control, threat analysis).

Traditionally, the effect of condition monitoring has been realized either by encoding condition evaluation as part of the application program or by polling the database (periodically) for condition evaluation. This paper explores a third alternative, viz. active condition monitoring from an object-oriented design perspective. The focus of this paper is on the design and implementation of active condition monitoring functionality for an object-oriented Database Management System (DBMS) and its evaluation: performance comparison with polling, influence of implementation strategies on performance, and identification of opportunities for optimization.

1. Introduction

Although the concept of condition monitoring is not new (ON conditions in programming languages and early DBMS's, and signals in operating systems), there is a genuine need for providing a similar capability having well-defined semantics and satisfying the efficiency requirements of DBMSs. A large class of applications, such as process control, threat assessment and analysis, monitoring of intensive care units, air traffic control, and cooperative processing, that require database support need to react (often subject to timing-constraints) to a variety of conditions that are defined over the database state and events that change the state of the database. Traditionally, DBMSs are passive. Hence, the effect of condition monitoring is achieved either by encoding condition evaluation as part of the application program (equivalent to posing external queries) or by polling the database (periodically) to detect whether any of the conditions have become true. Neither of these approaches is satisfactory. Encoding condition evaluation within the application program not only transfers the burden -- of determining the conditions, formulating queries for these conditions, and the time of their evaluation -- to the application programmer but also interferes with

† This work was carried out when the authors were with Xerox Advanced Information Technology (previously Advanced Information Technology Division of Computer Corporation of America), 4 Cambridge Center, Cambridge, MA 02142. This work was supported by the Defense Advanced Research Projects Agency and by Rome Air Development Center under contract No. F30602-87-C-0029. The views and conclusions contained in this paper are those of the authors and do not necessarily represent the official policies of the Defense Advanced Research Projects Agency, the Rome Air Development Center, or the U.S. Government.
the application development. Furthermore, optimization of such conditions is extremely difficult. Polling results in wasted resources and in addition the user now has to determine the frequency of polling which is dependent on a variety of parameters including the frequency of update, timeliness (the time window within which the condition needs to be detected).

Recent research on active databases [STON86, STON87, DARN87, DITT86] is aimed at supporting new capabilities (e.g., alerters, triggers, situation-action rules) and techniques for their optimization (e.g., lazy, eager, overlapped execution). Active capability is being viewed as a unifying mechanism for supporting a number of DBMS functionality, such as integrity/security enforcement, maintenance of materialized (e.g., view) data, constraint management, and rule-based inferencing. On the other hand, AI systems have traditionally used daemons for asynchronous execution. LOOPS [BOBR83] and KEE [INTE85] have incorporated active values as a new paradigm that generalizes asynchronous rule processing. However, these systems do support sharing, consistency, and concurrent execution of transactions.

The goals of this effort are to design and implement active functionality for an object-oriented DBMS to facilitate the evaluation of this new approach and gain additional insights into its performance and optimization to feed into the HIPAC project [CHAK89, DAYA88b], investigated at XAIT. Specifically, the emphasis is towards answering basic questions, such as is active condition monitoring always better than polling?, should polling be retained as a viable alternative?, and what issues need to be considered when an existing DBMS is made active? than an in-depth performance evaluation. For this purpose, a prototype object-oriented DBMS was designed and implemented on a Symbolics machine using Symbolics Common Lisp and flavors. This prototype was modularly extended to include active objects for supporting automatic condition monitoring in addition to other DBMS functions. Needless to say that transaction processing and concurrency control were not included in the prototype.

The remainder of this paper is structured as follows. Section 2 briefly describes the implementation of an object-oriented DBMS using Symbolics Flavors. Section 3 discusses the design and implementation of active objects and the functionality of the resulting prototype. Section 4 describes the evaluation of the prototype. It also provides an analysis of the observed results. Section 5 contains conclusions.

2. Symbolics Implementation of a Prototype OODBMS

For the purposes of this effort, a subset of PROBE\(^2\), excluding spatial/temporal objects and recursion, was designed and implemented on Symbolics using Common Lisp and its object-oriented extension, Flavors. In this version stored and virtual relations are supported. A generalization of relational algebra is used as the query language. The algebra consists of relational operators (such as select, project natural join etc.) with suitable extensions (apply-and-append for example) to support the PROBE object classes. The system is composed of a hierarchy of objects and operations on objects. For example, relation is an object consisting of file objects, attribute objects and other instance variables. These objects are in turn built using other objects. Figure 2.1 shows the object hierarchy (including the objects introduced for active condition monitoring) used for supporting the

---

1The prototype has the functionality of a subset of PROBE — a passive, extensible DBMS developed at CCA [DAYA87a].

2PROBE [MANO86, ROSE86, DAYA87a, DAYA87b], supports a rich data model incorporating spatial/temporal objects as well as recursion at the kernel level. The Probe Data Model (PDM) provides objects and functions as basic constructs. An object is used to represent a real world object. Functions can be applied to objects to obtain properties of objects, invoke operations on objects, and describe relationships among objects. Objects and functions are manipulated through an algebra that is a generalization of the relational algebra.