Presented at DEXA'91 (Database and Expert Systems Applications), Berlin, Germany, 1991

Building Adaptive Applications using Active Mediators

Tore Risch*
Hewlett-Packard Laboratories
1501 Page Mill Rd.,
Palo Alto, CA 94303

Gio Wiederhold
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

We have extended current DBMS technology to support applications in dynamic and heterogenous environments. Our approach raises the level of software support available from DBMSs to include an intermediate layer of software to *mediate* between databases and their use by applications and users. In particular we are demonstrating *active mediators*, where the application instructs a mediator to actively monitor databases for change in information that the application depends on. We identify how mediators can support applications that are sensitive to change.

A prototype platform for these classes of mediators has been developed. As a uniform interface language throughout the system we use OSQL, a declarative object-oriented query language. OSQL statements are optimized using concepts extracted from Datalog and relational database research.

1 Introduction

Future computing environments will have large numbers of workstations connected via communication networks. Workstations will have powerful computation capabilities; server stations store, maintain, and do inferences over local data- and knowledge-bases, or *information bases*. Each information base is maintained locally by human experts and is likely to be autonomous from other information bases. Data resources, servers, and applications are heterogeneous. The environment needs to support frequent changes and additions to these data and computation resources. Information and control often has to be exchanged among different information bases. We will have a large distributed *information network*, through which it is possible to access data stored in a variety of local forms, integrate the data, and obtain information without the use of human intermediaries, as is common today.

Wiederhold [26] has proposed to introduce an intermediate *mediator* layer of software between databases and their applications and users. The mediator layer insulates individual users and servers from the necessity to maintain detailed models of the different and changing *data sources* within the information system.

We use the mediator concept to extend current DBMS technology to support such distributed, heterogeneous, and dynamic environments. Mediators make it easy to 'plug in' new data resources once there is a public interface protocol. In this work we focus on *active* mediators, where the application instructs a mediator to actively monitor databases for changes to information that the application depends on, and provide primitives for applications to adapt to these changes. We discuss what classes of mediator modules are needed to support active databases.

In Section 2 we motivate our work by describing a relevant real-life problem scenario. In Section 3 we describe the components of a mediator architecture serving as a platform to solve these real-life problems with references to related work. In Sections 4 and 5 we give an overview of the design of our mediator platform. Finally, Section 6 summarizes the work and indicates directions for future work.

2 A Scenario

To illustrate what new services are needed beyond what is provided by present DBMSs, it is fruitful to think of some possible problem scenarios. In this section we discuss one such scenarios, namely the problem of manufacturing production planning in a distributed envi-

*The work was done while visititing the Hewlett-Packard Stanford Science Center.

ronment with many independent production sites. The scenario mainly has grown out of discussions with people in HP Labs.

In our problem scenario we have a large corporation with production facilities distributed over many separate sites around the world. The computer environment is heterogeneous, different sites use different DBMSs. The hardware and software environments differ as well.

The corporation continuously receives orders that are composed of items that are produced at different sites. Often one has the choice to produce the same item at many sites, and one then makes choices depending on production costs, transportation costs, production capacity, etc. There are often dependencies between the production of order items, so that one site must wait for some other site to produce some subpart before the item can be made. The customers are promised delivery at some specific date, and it is important to neither deliver too early (because of cost of inventory) or too late.

When an order arrives, the possible production sites are polled to determine their capacities, costs, schedules, etc. Given this data and the customer's expectations, distributed production schedules for the order are produced for the various sites. We have implemented database support for executing such plans; we have not attacked the scheduling problem itself. The schedules are based on the polled data which occasionally change while the schedules are executed. Thus we have to be able to cope with world changes that may modify the plan while it is being executed. In the worst case some critical site may stop entirely and the plan will have to be completely redone. Because of the complexity of the planning problem, it is likely that the planning is broken down into many local interconnected planners, and normally not all of them have to be replanned at once.

Similar scenarios can be constructed for other related areas, e.g., computer network service planning systems, and project planning and tracking systems.

3 Mediator Classes

We will now continue by discussing a possible architecture covering the production planning scenario. architecture.

We have identified the following three classes of mediators (Figure 1):

Task Models

We break out domain knowledge now hidden in application programs and store them in a special kind of mediator which implements sharable and inspectable domain knowledge bases. We call such a domain knowledge base a *Task Model*. The task models allow us to maintain knowledge more easily by storing it in these limited-sized and specialized knowledge bases.

In our scenario, the actual task planners are application programs that generate plans for carrying out the distributed tasks as specified by users. The output of a task planner is a set of task models to be executed by the sites together. This interaction need not be completely automated; the plans could be done in cooperation with users, where the expert manually develops task models to carry out the plan.

While planners provide some of the most advanced and demanding applications for the mediator architecture, the architecture also supports ordinary application programs that retrieve distributed data through mediators, e.g. to produce weekly production summary reports.

We use extensions to OSQL, WS-OSQL, as the language to build task models. We have developed a platform for efficient representation of task models based on a main memory WS-OSQL compiler [15] interfaced with Iris. An overview of this platform is given in Section 5. Related work includes work on extracting complex objects from databases [2, 22], structuring knowledge bases [25], and storing business rules in knowledge base modules [8]. Unlike these system we use the same extended OSQL query language as a uniform representation of both persistent data and task models. Task model OSQL queries are optimized using techniques from optimizing relational databases and Datalog [7, 20]. Our research aims at developing small, modular, inspectable, easily maintainable, and well integrated knowledge modules rather than large scale complex knowledge bases [12, 16].

Monitors

We need some mechanism to handle the problem of dynamically changing contents and locations of data. For example, when an initial scheduling plan is made, the assumption is that the data, critical for the execution of the plan, is not changing. In practice these data are frequently invalidated. Therefore the planner is notified when data is updated that was assumed to be constant when the plan was made. Now the plan must be adapted as well.

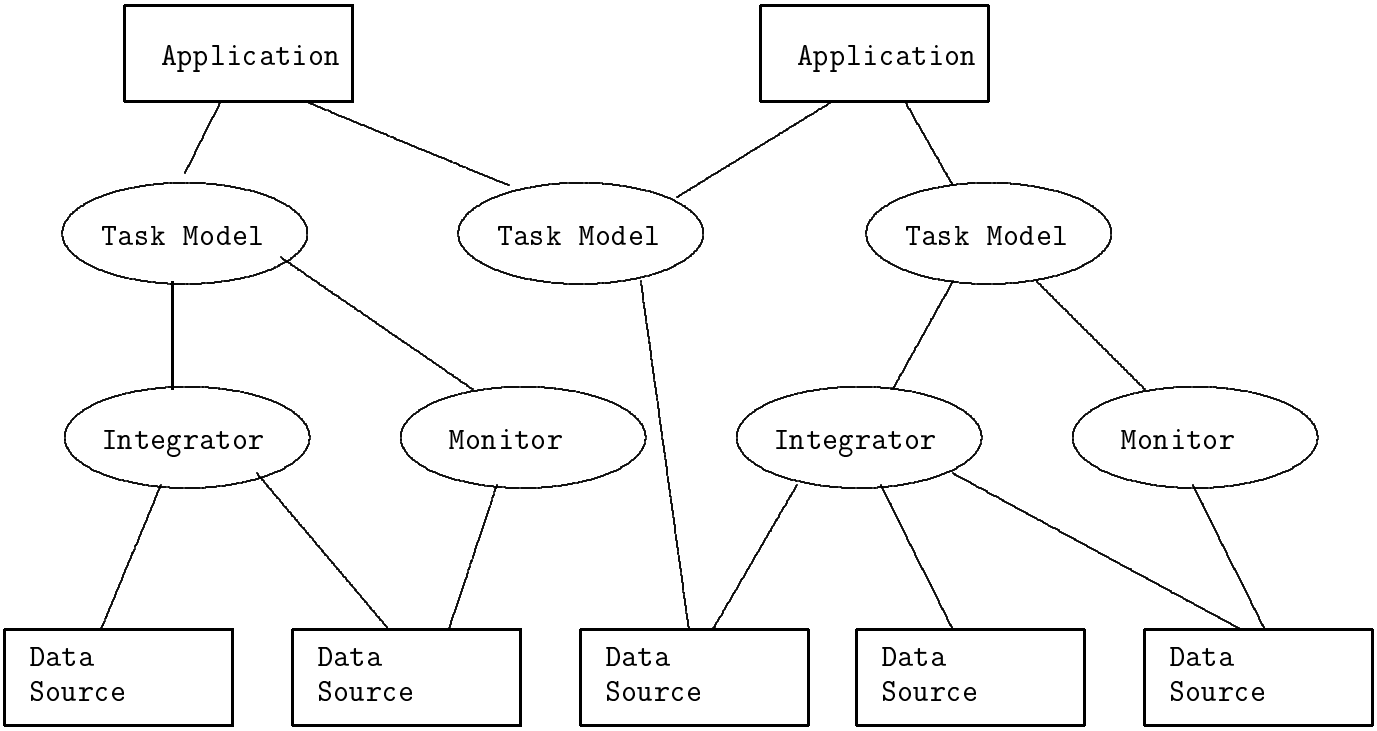

Figure 1: Examples of Mediators

Mediators continuously monitor these invariant data and notify the planner when the invariants change to a significant extent. Let's call these mediators *monitors*. Traditional DBMSs are passive since they passively respond to requests from application programs. With monitors the database becomes an *active database*, because procedures of application programs are invoked from the DBMS, when triggered by state changes in data sources. This technique provides a way to pass control between cooperating application programs through a database.

We have extended OSQL to include primitives for monitoring state changes in Iris databases and implemented the extension for the Iris prototype [12, 13]. Related research includes work on coordinating long running activities [5, 23], blackboard architectures [11], forward chaining rule based systems[4], and constraint propagation languages[18]. We use the same extended OSQL language both for passive and active queries and both within task models and for persistent databases. We describe our database monitors in Section 4.

Integrators

The knowledge sources can be represented in different ways. For example, different sites may use different DBMSs and data models, similar data may be represented with different data formats, etc. We therefore need mediators that retrieve and combine results from different knowledge sources, check their consistency, and present a higher level view to applications [10]. We call these mediators *integrators*. Integrators decouple the application from the necessity to maintain multiple data models for all knowledge sources. The Pegasus project at HP Labs [1] aims at using OSQL to support such integrator models. We include its concepts here to make the overall picture complete.

Interface Language

Finally we need to define formal languages between the various mediator levels. The application programs need a standard interface language. Lower level software also needs well defined interfaces. For example, we need some standard way to interface new knowledge sources so that mediators 'understand' them. We are using an extended version of the OSQL language [3, 6, 9] as such a uniform interface language.

4 Monitors

Database change monitors are computer programs that observe changes in the contents of database access queries, e.g., the current price of some commodity, the highest paid employee in a department, or the expenses of a department relative to its sales. We have implemented database change monitors [13] within the object-oriented DBMS Iris [6].

We use non-procedural extensions to OSQL as the language to express what type of changes are of interest for a given application. This is achieved by the programmer specifying a query (view) whose result is monitored for change. The programmer thus registers interest in a class of database state changes and delegates to the DBMS to generate a plan to infer when these value states change because of database updates. The algorithms to track such value changes are complicated and depend on information outside of the scope of the programs interest. Hence the programmer should be freed from having to specify their details. The system provides *monitor tuning parameters* that instruct the system to filter out insignificant state changes thereby decreasing the notification frequency.

The programmer must specify what happens when the value of a monitored view changes. A convenient way to do this is to specify a *tracking procedure* or *tracker*, which is a procedure of the application that is invoked by the DBMS when monitored data change. The DBMS thus keeps track of which tracking procedures monitor which object attributes and contains a mechanism to call the tracking procedures upon data changes. Monitoring processes are autonomous relative to updating processes, so that committing transactions need not wait for tracking procedures to finish, a distinction vis-a-vis POSTGRES triggering [19].

Possible tasks for tracking procedures include

- Notifying the end user that data have changed.
- Refreshing data browsers
- Modifying values in mediators.
- Changing processing heuristics in mediators.
- Changing stored abstractions in mediators.
- Informing applications that data views which the application depends on have changed.

We allow the programmer to specify the monitor tuning parameters *time intervals* and *tracking periodicity* Δt. If tuning parameters are specified the monitors are not triggered immediately when the database value state change happens, but will be delayed as specified. For example, in business situations, some monitors are best checked once a day or some time before the end of the week [23]. Some applications require the monitor to be notified every time there is a data change. Sometimes this checking must be done before committing the updating transactions, which can be very expensive. The programmer is given a primitive to make this choice.

A range interval, the *change significance* Δv, can be declared so that tracking procedures are not invoked unless the monitored value has changed more than the interval. The change significance is specified as absolute range limits, or as the variance relative to the magnitude of the value.

It is advantageous if the active DBMS can do this kind of *dynamic filtering* before notifying the application, in order to decrease the frequency of notification for intensively updated data. The application can dynamically change the difference interval to decrease the reactivity if it is performing some change-intensive task. Dynamic filtering is required, for example, by real-time monitoring AI systems where the tracker initiates time consuming reasoning activities [21].

5 Active Task Models

The DBMS architectures developed for conventional databases are not always feasible for supporting active databases. In particular, disk based databases are often too slow for applications requiring fast responses. Our architecture is a combination of traditional persistent data representation for conventional applications and main memory techniques for time critical mediators.

Task models represent domain knowledge bases. We represent our task models using the Iris data model whose lingua franca is the OSQL query language. Task models are presumed to contain moderate data volumes but complex structures. To obtain reasonable performance we need to represent task models in main memory. Therefore we have developed a main memory OSQL database implementation: WS-IRIS (Workstation IRIS) [15]. WS-IRIS is a fully operational DBMS for a workstation. WS-IRIS manages main-memory databases and, for better efficiency, it uses main memory data structures instead of the disk oriented data structures of classical DBMSs. It also provides transac-

tions, limited concurrency control, logging, and recovery.

WS-IRIS contains an extensible cost-based query optimizer that translates queries in an extended OSQL dialect, WS-OSQL, into Datalog [20] programs for subsequent interpretation. It incorporates techniques for conventional query optimization, where effective for our main memory storage management. We also use optimization techniques tailored for the object-oriented nature of WS-OSQL.

The task model platform is interfaced with Iris so that data can be extracted from Iris databases into task models. With the Pegasus [1] extension to Iris we will be able to access heterogeneous data sources.

In order to support our scenario we also need to represent task models that are sensitive to changes both in underlying data sources and in local data. We therefore have implemented database change monitors both within WS-IRIS and in the IRIS prototype. Primitives have been added to OSQL to monitor changes of views of both local and global data. For example, a task model may have a set of rules for selecting a preferred part based on some local thresholds, e.g., maximum delivery time and failure rate. Using monitors the task model can instruct the system to execute some rule whenever some other part becomes the preferred one. The tuning parameters Δt and Δv add some stiffness to the system by filtering out temporary and insignificant data changes.

6 Discussion

We have described an architecture which has been implemented to support database applications that are sensitive to changes in an underlying heterogeneous data sources. Using such a system one writes applications that adapt to change rather than prohibit change as is customary in traditional database transaction systems. We used the mediator concept [26] to identify three kinds of mediators which we found important in such an architecture:

1. *Active Task Models* are domain knowledge bases which contain both conventional rules and rules sensitive to change in both local and external data.

   A planned generalization is to include the notion of time in the system in a coherent way in order to support rules that access timing information about data and knowledge [17, 24].

2. *Database Change Monitors* are mediators which detect changes in underlying data source by tracking changes in derived views. The monitors will notify the active task models when significant data changes are detected. The programmer specifies declaratively what constitutes significant change so that the monitors can filter insignificant notifications.

3. *Integrators* are meditators which, using schema integration and data conversion facilities [10], integrate heterogeneous data representations to be presented uniformly to other mediators.

As a common interface language to the different mediator modules we use extensions to OSQL [6], WS-OSQL [12]. The approach has been demonstrated by a prototype implementation, WS-IRIS (Workstation Iris) [12].

An important application area for active mediators is in the support of systems where persistent data is checked out into main memory *work areas*, which are here represented as active task models. During long sessions the user would work directly with specific and hence efficient active task models and only occasionally check data back into the central database. This complements the view object paradigm proposed by Wiederhold [22] and implemented in the PENGUIN project [2]. Here we provide the same uniform query language, OSQL, both for the shared database, the extraction of data into work areas, and for representing the work areas themselves.

Active task models also have applications in CAD/CAM and CASE systems, where a design is stored persistently in a database. A group of designers intend to work cooperatively on a specific design. They therefore check out the design into an active task model. The designers would work concurrently on the design in the active task model using some supporting software tools. The tools would use monitors to notify designers when some other designer modifies some data that is of interest. Winslett et al [27] describe an architecture for consistency maintenance in a design database, which could be supported by active task models.

An advantage with having data communicated between applications stored in a database is that the data then also is accessed using a query language instead of being hidden inside data structures within the applications. Having a query language permits a level of integration that is hard to obtain with direct use of communication protocols.

ACKNOWLEDGEMENTS:

The KSYS group at Stanford (supported partially by DARPA contract N39-84-C211) provided insights into the mediator concept. Peter Lyngbaek and the DTD group at HP Labs helped us understand Iris. Gio Wiederhold acknowledges support from HP Stanford Science Center and from NSF contract DMC 86-19595-A1.

References

[1] R.Ahmed, P.Desmedt, W.Kent, A.Rafii, W.Litwin, M.Shan: The Pegasus project: Information Management in a Heterogeneous Database Environment, *IEEE COMPCON*, March 1991.

[2] T.Barsalou, G.Wiederhold: Complex objects for relational databases, *Computer-Aided Design*, 22(8), Oct. 1990, pp. 457-468.

[3] D.Beech: A Foundation for Evolution from Relational to Object Databases, *Advances in Database Technology - EDBT '88*, Lecture Notes in Comp. Sc., Springer-Verlag, 1988, pp. 251-270.

[4] L.Brownston, R.Farell, E.Kant, N.Martin: *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, Mass., 1985.

[5] U.Dayal, M.Hsu, R.Ladin: Organizing Long-Running Activities with Trigger and Transactions, *Proc. SIGMOD*, May 23-25, Atlantic City, 1990, pp. 204-214.

[6] D.Fishman et al: Overview of the Iris DBMS, in W.Kim, F.H.Lochovsky (ed.): *Object-Oriented Concepts, Databases, and Applications*, ACM press, Addison-Wesley Publ. Comp., 1989.

[7] R.Krishnamurthy, S.Zaniolo: Optimization in a Logic Based Language for Knowledge and Data Intensive Applications, *Advances in Database Technology - EDBT '88*, Lecture Notes in Comp. Sc., Springer-Verlag, 1988, pp. 16-33.

[8] P.Lucas, T.Risch: Representation of Factual Information by Equations and their Evaluation, *Proc. Intl. Conf. on Software Eng.*, Tokyo, Japan, Sept. 13-16, IEEE, New York, 1982, pp. 153-167

[9] P.Lyngbaek and the OODB Team at CSY: *OSQL: A Language for Object Databases*, Technical report, HP Labs, HPL-DTD-91-4, 1991.

[10] L.deMichiel: Performing Operations over Mismatched Domains, *Proc. of IEEE Data Eng. 5*, Los Angeles, Feb. 1989.

[11] P.Nii: The Blackboard Model for Problem Solving, *AI Magazine*, Vol. 7, No. 2, Spring 1986, pp. 38-53.

[12] T.Risch, R.Reboh, P.Hart, R.Duda: A Functional Approach to Integrating Database and Expert Systems, *Communications of the ACM* 31, 12 (Dec. 1988), pp. 1424-1437.

[13] T.Risch: Monitoring Database Objects, *Proc. VLDB*, Amsterdam, the Netherlands, 1989.

[14] T.Risch: *Tuning the Reactivity of Database Monitors*, Technical Report, HP Labs, HPL-90-17, 1990 (also part of [26]).

[15] T.Risch: *The Translation of Object-Oriented Queries to Optimized Datalog Programs*, Technical Report, HP Labs, HPL-DTD-91-9, 1991.

[16] E.H.Shortcliffe: *Computer-based medical consultations: MYCIN*, American Elsevier, New York, 1976.

[17] R.Snodgrass, I.Ahn: Temporal Databases, *IEEE Computer*, Vol. 19, No. 9, Sept. 1986, pp. 35–42.

[18] G.L.Steele Jr., G.J.Sussman: CONSTRAINTS, in *APL79 Conf. Proc.*, (Rochester, USA), pp. 208-225.

[19] M.Stonebraker: The design of POSTGRES, *Proc. SIGMOD*, 1986, pp. 340-355.

[20] J.D.Ullman: *Principles of Database and Knowledge-Base Systems*, Volume I and II, Comp. Sc. Press, 1988.

[21] R.Washington, B.Hayes-Roth: Input Data Management in Real-Time AI Systems, *11th Intl. Joint Conf. on Artificial Intelligence*, 1989, pp. 250-255.

[22] G.Wiederhold: Views, objects, and databases, *IEEE Computer*, 19(12), 1986, pp. 37-44.

[23] G.Wiederhold, X.Qian: Modeling Asynchrony in Distributed Databases, *3rd Intl. Conf. on Data Eng.*, Los Angeles, CA, Feb. 3-5, 1987, pp. 246-250.

[24] G.Wiederhold, S.Jajodia, W.Litwin: Dealing with Granularity of Time in Temporal Databases, *Nordic Conf. on Adv. Inf. Syst. Eng.*, Springer, 1991.

[25] G.Wiederhold, P.Rathmann, T.Barsalou, B.S.Lee, D.Quass: Partitioning and Composing Knowledge, *Inf. Systems*, Vol.15, No.1, 1990, pp. 61-72.

[26] G.Wiederhold,
T.Risch, P.Rathmann, L.DeMichiel, S.Chaudhury, B.S.Lee, K.H.Law, T.Barsalou, D.Quass: *A Mediator Architecture for Abstract Data Access*, Stanford Comp. Sc. Dept., STAN-CS-90-1301, 1990.

[27] M.Winslett, K.Hall, D.Knapp, G.Wiederhold: Use of Change Coordination in an Information-rich Design Environment, *IEEE Design Automation Conf.*, Las Vegas, June 1989.