Third Generation *TP* Monitors: A Database Challenge

Umesh Dayal* Hector Garcia-Molina† Mei Hsu‡ Ben Kao§ Ming-Chien Shan¶

Background

In a 1976 book, "Algorithms + Data Structures = Programs"[15], Niklaus Wirth defined programs to be algorithms and data structures. Of course, by now we know that man does not live from programs alone, and that there is a second fundamental computer science equation: "Programs + Databases = Information Systems."

Database researchers have traditionally focused on the *database* component of the equation, providing shared and persistent repositories for the data that programs need and produce. As a matter of fact, a lot of us have worked hard to hide or ignore the *programs* component. For instance, *non-procedural* languages like *SQL* and relational algebra have been the holy grail of the database field, letting us describe the data in the way we want without need to write messy *programs*. The magic wand of transactions makes programs that execute concurrently with our non-procedural statements suddenly disappear: these other programs appear as *atomic actions* that are either executed before we started looking at our data, or will be executed after we are all done with our work. The wonders of fault tolerance and automatic recovery guarantee that we never have to concern ourselves with our statements failing or being interrupted. The data we need will always be there for us, and our statements will always run to completion.

Unfortunately, the real programs that operate on databases are in many cases more complex than the classical ones like "withdraw 100 dollars from my account" or "find me all my blue eyed great-grandfathers." For one, programs may be much longer, requiring many database interactions. Furthermore, programs need to interact with other concurrent programs, getting results from and to them. They may also need to be aware of their environment, perhaps monitoring the execution of another program, or taking corrective action when some system components fail. Of course, this is not to say that transactions and non-procedural query languages have not been great contributions. In many cases, they are all that is needed to program one's application. But beyond that there are many cases when one must deal with multiple concurrent applications. Indeed, a critical problem facing complex enterprises is the automation of complex business processes. Enterprises today are drowning in an ocean of data, with a few isolated islands of automation consisting of heterogeneous databases and legacy application programs, each of which automates some point function (e.g., order entry, inventory, accounting, billing) within the enterprise. As the enterprise attempts to automate its business processes, these isolated islands have to be bridged: complex information systems must be developed that need to span many of these databases and application programs. Traditional database systems do not provide the supporting environment for this.

Our programming languages colleagues have been working on the programs component of our fundamental equation, but the database component has traditionally been ignored or hidden. There has been a lot of recent interest on languages that support persistent objects, but often the goal is to make the database that holds the objects look as little as possible like a database. That is, the persistent objects are to be handled just as if they were volatile objects, even though they are not. Also, the programming languages researchers have borrowed the notions of transactions and serializable schedules to hide as much as possible concurrent

*Hewlett-Packard Laboratories, Palo Alto, CA. e-mail: dayal@hplud.hpl.hp.com

†Stanford University, Department of Computer Science, Stanford, CA. e-mail: hector@cs.stanford.edu

‡Digital Equipment Corporation, Mountain View, CA. e-mail: hsu@ocean.ucs.dec.com

§Princeton University, Department of Computer Science, Princeton, NJ. Current e-mail: kao@cs.stanford.edu

¶Hewlett-Packard Laboratories, Palo Alto, CA. e-mail: shan@hplmcs.hpl.hp.com

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0393...$1.50

execution and failures of programs. Finally, traditional programming languages (there are exceptions[4, 13]) have focused on "programming in the small," as opposed to "programming in the large." The goal of the former is to program single applications or to solve single problems, as opposed to programming an entire enterprise and all of its interacting applications.

Researchers from both camps have recently been addressing both components of the "Programs + Databases" equation. For example, database researchers have been adding triggers and procedures to database objects[2], resulting in so called *active databases*. These are important steps in the right direction (other related steps are listed below), but still do not address the full *programming in the large* problem.

In our opinion, the only software providers that have tackled both components of the "Programs + Databases" equation, and have a proven track record with real applications, are the *Transaction Processing Monitor (TPM)* builders[9].

TP Monitors

A *TPM* provides the support and "glue" that enables an enterprise to develop data-intensive applications. It provides facilities for running processes, for handling persistent communications among these processes, for communicating with users, and for accessing multiple database systems.

First generation *TPM* (e.g., IMS/DC, CICS[9]) provided all these facilities within a single monolithic system. They usually assumed users were at dumb terminals. Second generation *TPM*s are *open* to software from multiple vendors, provide support for smart clients, and allow a single transaction to interact with various database systems, coordinating via a two-phase commit protocol (e.g., Tuxedo, Encina[9]). We believe that the time is ripe now for third generation *TPM*s.

But before discussing third generation *TPM*s, it may be beneficial to make an analogy with database processing in the late 60's and early 70's. In those years, database processing (or simply data processing) represented a major portion of the computer market, yet there was very little academic or industrial research done on the topic. Practitioners were way ahead of researchers in terms of knowing how to model data and how to write application programs. Yet, they lacked a good understanding of the key modeling abstractions and execution models. After two decades of strong database research, industry now tends to follow the major database laboratories.

In the *TPM* area, the situation is currently analogous to that of the late 60's. *TPM*s represent a significant market share, and there even are start up companies (e.g., Transarc) building products. However, it is very hard to list more than a handful of researchers working on *TPM*s. Yet, just like databases in the 60's, there is a critical need to develop clean modeling abstractions, languages, and execution models for describing complex acitivites that involve interactions among many components and applications. The third generation *TPM*s we advocate will provide these; they will have to be constructed jointly by researchers and developers.

Requirements for Third Generation *TPM*s

Third generation *TPM*s (*3GTPM*s) will need to include facilities for modeling and managing the execution of business activities. Such activities typically consist of many steps, are of long duration, may require access to one or more shared databases, and may involve interaction with multiple individuals and multiple organizational units of the enterprise. An activity may start with a step that may in turn trigger other asynchronous and deferred steps. Each step may invoke applications that execute transactions over one or more databases. The flow of control and data among the steps has to be specified and orchestrated. Because the steps may access shared databases, they must be synchronized. Steps (or the systems on which they are executed) may fail; activities must be made reliable against such failures.

To illustrate these requirements, consider the following example of order processing in a highly automated manufacturing enterprise. An incoming customer request causes an order to be created and inserted into an order entry database. Then, the customer's address and status are validated (in one or more separate transactions) against the customer database. If invalid, the order may be rejected or returned for reprocessing. If valid, a transaction to check the inventory database is initiated. If the order cannot be satisfied with the current inventory, a manufacturing activity has to be initiated. Several orders may be batched before being sent to the manufacturing shop. Manufacturing itself is a complicated activity, involving many steps that may be performed by many different humans, robots, or machines. Further downstream activities include testing, quality control, packaging, shipping, accounting, billing, and customer service. Some of the downstream activities may involve external organizations such as subcontractors and suppliers of materials and services. An activity like this may involve tens of organizations within and outside the enterprise, hundreds of humans or machines, hundreds of databases, and thousands of interconnected applications.

With existing *TPM*s, the linkage among related transactions (i.e., workflow) is encoded in the applications themselves, documented in operational manuals to be enforced manually, or (worse still) carried in people's heads.

A business activity often requires multiple users (re-

sources). Our order processing example involves order entry clerks, inventory clerks, managers, engineers, technicians, accountants, and a host of other resources. Today, the policies for authorization, allocation and notification of resources to perform related tasks in a timely fashion are also embedded and scattered in applications themselves or enforced manually.

Furthermore, every enterprise has business rules that typically serve to differentiate it from its competitors. For example, special relationships may exist with preferred customers or suppliers. Inventory may be distributed among multiple warehouses based on projected regional and seasonal demands. Today, these rules are also embedded in applications or implemented manually.

As the business activities grow in complexity, the workflows, policies, and business rules embedded in the applications become difficult to understand, and, in some cases, close to impossible to maintain. Today's generation of *TP* monitor technology, while crucial in the synchronization and management of single transactions, provides relatively little help in managing applications that are distributed and interconnected in time and in space.

We believe that activities like the one we have illustrated occur in many application domains. They range from small-scale "office information systems" where the activities of a single office are captured, to large scale intra- and inter-enterprise information systems such as for service order provisioning in the telecommunications industry, insurance claims processing, patient tracking in hospitals, and controlling scientific experiments and simulations.

Third generation *TPMs* will provide a complete environment for organizing, developing, and executing such integrated database-intensive applications. In addition to conventional *TPM* facilities for process management, communication, and coordination, *3GTPMs* will provide facilities for specification, management, communication, and coordination of complex activities.

Research Issues

We briefly discuss some of the key open issues in the development of *3GTPM* technology.

Work Flow Model and Language

A high level workflow language is needed to specify how activities can be composed from base applications and recursively from other activities. In many ways a workflow language is like a regular high level programming language. For example, it needs to provide a way to specify that if steps A and B are completed, then step C should follow, or that if application X fails, then program Y needs to run. However, a workflow language must be *declarative* to facilitate the design, analysis, and maintenance of flows. (Graphical interfaces may be preferable to textual specification languages.) In addition to specifying the flow of control and data within an activity, a workflow language also needs to provide the following features.

1. It must allow for persistent execution (e.g., programs or applications may have to be retried in case of failures); error handling; rollback with compensation (since some transactions within an activity may have been committed, they may have to be compensated when failures occur); and rollforward (since activities are of long duration, it is typically unacceptable to roll them all the way back to the beginning).

2. It must allow for legacy applications and heterogeneity. A program should be able to request a service in a foreign system or from an existing application.

3. It must allow for execution by multiple end users. The programs of different end users should be allowed to interact.

4. It must allow for monitoring, tracking, and querying the status of activities. For example, It should be possible to request that a particular action be executed when another activity terminates abnormally or signals some event, or to request information on the status of an activity or on how it entered its current state.

Methodologies for designing flows and analyzing their behavioral properties are needed.

Also needed is an execution model that governs the rules for breaking the execution of an activity into discrete events that can be reliably recorded, tracked and monitored.

Extended transaction models and algorithms

Transactions are short-duration, single-user tasks that preserve the ACID properties. New abstractions are needed for characterizing the properties for long-running, multi-user, complex flows of tasks.

With the new abstractions, new algorithms and protocols are needed for managing constraints and dependencies among sub-units of work, for ensuring reliability and recoverability, and for ensuring consistent database images under concurrent access.

Architecture

A *TPM* funnels requests and results between clients and application servers. It offers services, such as reliable queueing services and transaction management services, to ensure recoverability and to optimize for performance. One of the challenges for *3GTPM* lies in how advanced service components and heterogeneous application and database servers can be included, and how complex workflow can be managed.

For example, the Common Object Request Broker (*ORB*)[12] service may provide the backbone for heterogeneous application integration. The *ORB* service allows the functions ("methods") provided by application servers to be organized in an object oriented framework. It enhances the manageability and extensibility of the functions of application servers.

Database integration, translation, and query processing services that allow transparent access to heterogeneous database systems may also be supported.

As another example, a variety of extended transaction models, such as nested transactions and sagas[8], may be supported by an extensible transaction manager. Incorporating an extensible transaction manager allows new transaction models to be dynamically defined in a *3GTPM*.

In addition, application and database servers in *3GTPM* may be *active*. Active servers are capable of monitoring events of interest to other components in the system, and provide event notification when they occur.

A *workflow manager* service manages the events generated by the cooperating active servers, and channels events reliably based on high-level workflow specifications. The workflow manager allows individual application servers to focus on specific, modular tasks, while it assumes the responsibility of overseeing the flows among tasks. It also manages a variety of resources such as people or manufacturing robots. In contrast, second generation *TPM*'s only know about two resources: memory and CPU cycles.

A rational framework must be developed to allow a *3GTPM* to aggressively accommodate extended service components such as the *ORB* service, extensible transaction management services, transparent multidatabase access services, and workflow services, as well as active servers.

Performance and security

Performance optimization in traditional *TPM*s is well developed. For example, a mature *TPM* is quite effective in dynamically allocating instances of the application servers to provide for load balancing. System security and protection capabilities are also well developed in mature *TPM*.

The performance and system security issues in the *3GTPM* are yet to be understood and characterized. The distributed and heterogeneous nature of the environment, and the scale and number of components involved, will require new techniques to be developed.

Organizational resource management

A *3GTPM* manages multi-user workflow in an activity. Organizational policies are used to determine the assignment of organizational resources (users, robots, machines) to steps of an activity. For example, the policies might dictate which managers must approve a work order before it is executed. Abstractions, algorithms and services (e.g., a "role resolution service") are needed to capture and implement these policies.

Summary

In summary, the goal of third generation *TP* monitors is to allow significantly improved specification and management of complex processes and their interactions. This will require that current database and *TP* monitor abstractions, architectures, and technologies be expanded. These technologies will need to scale to huge numbers of processes and application programs that span many organizations and involve many non-database resources such as humans, processors, and services.

There is already some work addressing the open issues we have outlined above, including new transaction and workflow models (e.g. [5, 8, 7, 3, 1, 11, 2, 6, 14, 10]). What is lacking is the *integration* of these ideas into a coherent transaction processing monitor, and the *demonstration* of these novel ideas on real applications. Herein lies the challenge!

Acknowledgement. The authors thank Dieter Gawlick for insightful discussions.

References

[1] M. Ansari, L. Ness, M. Rusinkiewcz, and A. Sheth. Using flexible transaction to support multi-system telecommunication applications. In *Proc. of the 18th Int. Conf. on VLDB*, 1992.

[2] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, 1990.

[3] U. Dayal, M. Hsu, and R. Ladin. A transaction model for long-running activities. In *Proc. of the 17th Int. Conf. on VLDB*, 1991.

[4] F. DeRemer and H. H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Enginerring*, SE-2(2):80–86, June 1976.

[5] A. Elmagarmid. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers Inc., 1992.

[6] A. Elmagarmid, J.Chen, W. Du, O. Bukhres, and R. Pezzoli. InterBase: An execution environment for global applications over distributed, autonomous, and heterogeneous software systems. Technical Report CSD-TR-92-016, Purdue University, 1992.

[7] H. Garcia-Molina, D. Gawlick, J. Klein, and and K. Salem K. Kleissner. Coordinating multi-transaction activities. In *IEEE COMPCON*, 1991.

[8] H. Garcia-Molina and K. Salem. Sagas. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, 1987.

[9] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., 1993.

[10] Y. Halabi, M. Ansari, R. Batra, W. Jin, G. Karalbatis, P. Krychniak, and M. Rusinkiewcz. An environment for specification and execution of multi-system applications. In *Proc. of the 2nd Int. Conf. on System Integration*, 1992.

[11] M. Hsu, A. Ghoneimy, and C. Kleissner. An execution model for an activity management system. In *Proc. of Workshop on High Performance Transaction Systems*, 1991.

[12] Object Management Group, Framingham, MA. *The Common Object Request Broker: Architecture and Specification. OMG Document No. 91.12.1*, Dec. 1991.

[13] J. M. Purtilo. A software interconnection technology. Technical Report CS-TR-2139, Dept. of Computer Science, University of Maryland, 1988.

[14] A. Sheth and L. Kalinichenko. Information modeling in multidatabase systems: Beyond data modeling. In *Proc. of the 1st Conf. on Information and Knowledge Management*, 1992.

[15] N. Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, 1976.