Caprera: An Activity Framework for Transaction Processing on Wide-Area Networks

Suresh Kumar
Vice President, Engineering
Tactica Corporation
suresh@tactica.com

Eng-Kee Kwang
Chief Technology Officer
Tactica Corporation
engkee@tactica.com

Divyakant Agrawal
Associate Professor
UC Santa Barbara
agrawal@cs.ucsb.edu

Caprera is an open framework for designing client/server applications that operate over a wide-area network. The activity model of Caprera used to extend transaction processing and transaction-oriented application in an open environment including mobile and remote clients connected by wireless, phone lines, or Internet is described here. Since Caprera enables off-line users on mobile platforms to interact with corporate transaction processing systems, Tactica Corporation markets its product as a programmable server software for off-line transaction processing (OFTP). This paper describes the design rationale and the product architecture of Caprera.

Introduction

Under the existing model of data processing, referred to as on-line transaction processing (OLTP), all user interactions, called transactions, are executed directly on the machine that stores the enterprise database. The OLTP model is extremely rigid in the way in which it provides access to the information for its users. There are at least two important changes which require a complete overhaul of the traditional OLTP model. The first is the increasing reliance on re-engineered business processes and workflow automation for transaction processing. The second is extending the enterprise database connectivity to the employees in the field. Current OLTP systems provide excellent support to execute a high volume of independent, short-duration transactions.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

Unfortunately, minimal support is available in most OLTP systems for controlling the execution of inter-dependent activities within a business process. Systems that provide support to manage business processes are referred to as workflow management systems (WFMS). Most WFMSs provide a tool to model workflow and a run-time system to control the execution of activities within the workflow. However, none of the WFMSs provide significant support or tools to design the activities themselves.

Due to advances in networking technology, most employees of an enterprise have some level of connectivity to the enterprise-wide information systems. In spite of this connectivity, inefficiencies result under the current OLTP paradigm because only those users who are directly connected can access the enterprise-wide databases. Tactica's goal is to develop a family of products that will allow transaction processing from mobile/remote platforms. This model of operation, which can be loosely termed as off-line transaction processing (OFTP), will enable remote and mobile users to integrate with the enterprise-wide information infrastructure. In this paper, we describe the design and implementation of Caprera, software primarily developed to extend database connectivity to mobile and remote users.

Caprera Activity Model

The Caprera system consists of an open, standards-based software framework that provides a complete set of software tools to efficiently build, deploy, and manage transaction-oriented applications that operate over wide-area networks. In this section we describe the Caprera activity model, as well as the rationale behind various design choices.

Design Rationale

A key goal of Caprera is to provide transaction processing capability from mobile/remote platforms on host or corporate databases. Since there exist billions of dollars worth of investments in legacy applications in such databases, it is clear that Caprera goals must be achieved without any modifications to either the host

DBMS software or the host databases. The most widely used OLTP solution to ensure data integrity is to lock data objects until transaction termination (often referred to as strict two-phase locking [1,3]). Although locks and two-phase locking are universally used in commercial DBMSs, it is commonly agreed that long duration locking may result in a significant performance degradation. Since Caprera user interactions are expected to be long duration, due to slow network links as well as asynchrony due to mobility, Caprera cannot afford to use locking as the only means to solve the data integrity problem. Another requirement for the Caprera design is to provide a uniform solution to fulfill the needs of all classes of Caprera users, i.e., mobile (or intermittently connected) as well as remote (connected over slow links) clients.

Since user interactions in Caprera are long-duration, the term *activity* is used to distinguish Caprera interactions from the standard transaction concept in databases. An activity enables Caprera users to execute applications with transactional guarantees from mobile and remote client platforms. Traditionally, the transaction paradigm has been used to ensure atomicity of user interactions with the database. Concurrency control protocols such as two-phase locking [1,3] are used to ensure execution atomicity of transactions, and recovery mechanisms typically based on write-ahead logging [5] are used to support failure recovery in database systems. A general consensus among database system researchers and practitioners is that ensuring atomicity of long-lived transactions [2,6] at the system level may result in significant performance degradation. Also, it is commonly agreed that execution atomicity and failure recovery (to a lesser extent) for long-lived transactions can be handled best at the application level itself. Caprera is designed to provide a framework that can be used to control the degree of atomicity at the application level.

Concurrency Control in Caprera

Figure 1 illustrates the layered architecture in which the Caprera system is deployed. As shown in this figure, the host database is now configured so that OLTP users interact with the database via the standard transactional interface using TP monitors and WAN users interact with the host database via activities that execute in a distributed manner on Caprera clients and servers. Figure 2 shows a distributed multi-tier environment for OFTP using Caprera. In a multi-tier environment, rules and events can be used to create the workflow necessary for complex transactions in a business process. From the correctness point of view, the Caprera framework must ensure execution atomicity not only of concurrent Caprera activities but also with respect to the transactions that are being executed at the host database. Caprera uses two different approaches to control the concurrent execution of transactions and Caprera activities. These approaches can be classified as pessimistic and optimistic approaches to concurrency control [1,3].

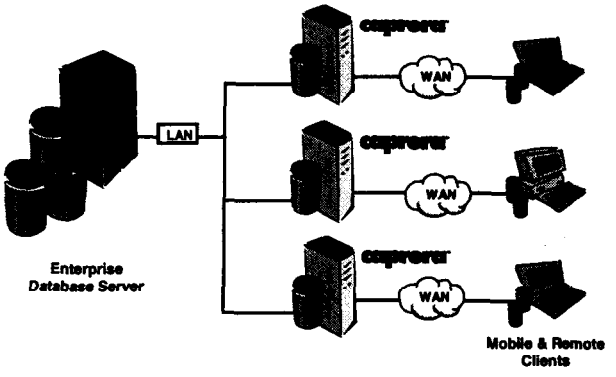

Figure 1

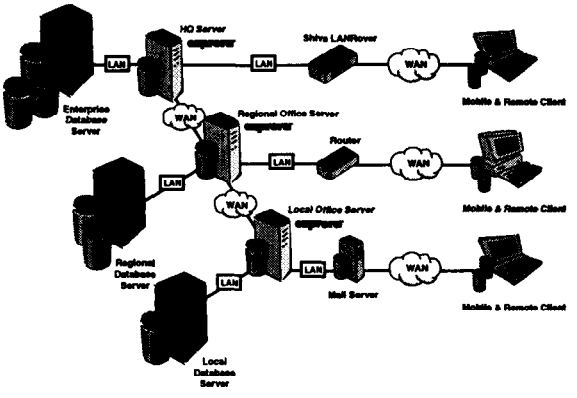

Figure 2

In the pessimistic approach, Caprera activities are encapsulated in a transaction block. The semantics of a transaction block are such that all interactions (i.e., read and write operations) from the activity to the host database are done by conforming to the host concurrency control mechanism. This ensures that the Caprera activity is serializable with respect to host database transactions.

In the optimistic approach, Caprera activities are executed without performing any synchronization with respect to the concurrent host transactions. A certification check is performed before committing the updates of a Caprera activity. Unlike in optimistic concurrency control protocols [4], failed certification does not necessarily result in the abortion of Caprera. An exception is raised and it is up to the application designers to appropriately handle the exception at the application level. For example, the application may choose to ignore the exception, may take some action to resolve the conflicts[1], may redo the activity, or may abort the activity.

[1] Caprera provides a conflict resolution editor.

In order to accommodate both approaches, Caprera activities are designed so that updates to the host database are deferred until commit time. An advantage of this approach is that when an activity is executed as a transaction block, only read locks are held during the life-time of the activity. Write locks are needed for relatively short duration.

Updates from Caprera activities are incorporated in the host database either when there were no conflicts (hence, the overall execution is serializable from the concurrency control point of view) or when the update conflicts have been resolved (indicating that there are no data inconsistencies at the application level even though the execution is not serializable). The certification check is the standard validation test from the optimistic concurrency control method [4]:

> *The read set of a Caprera activity is validated against committed writes in the host database.*[2]

In order to avoid execution anomalies after the certification, an application designer must execute the certification and the commitment of a Caprera activity as a transaction block. In effect, this results in the Caprera activity obtaining read and write locks during the termination phase which is significantly shorter than the entire life span of Caprera activities.

Failure Recovery in Caprera

In contrast to the problem of concurrency control, where an unconstrained execution of a Caprera activity may potentially leave the host database inconsistent, the impact of failed Caprera activities on the host database is not critical. A recovery mechanism is used in database management systems to eliminate the effects of partially executed transactions that failed due to system crash or transaction aborts. Write-ahead logging [5] is the most common technique to implement failure recovery in database management systems.

Caprera activities are structured so that all updates to the host database are deferred until termination[3]. Furthermore, a Caprera activity is either completely executed as a transaction block or the certification and update phase is executed as a transaction block. From the point of view of the host database, a Caprera activity is vulnerable to failure only after it performs its first update to the host database. Until then, failure of a Caprera activity has no effect on the host database. If transaction blocks are used to encapsulate the updates within an activity, then the updates are subject to the concurrency control and failure recovery mechanism of the host database. Hence, the recovery mechanism of the host database is sufficient for undoing the updates of aborted Caprera activities, as well as redoing the missing updates of committed Caprera activities during crash recovery.

Although the failure of a Caprera activity prior to the update phase does not impact the host database, it results in lost work for Caprera clients. In order to minimize communication between Caprera clients and servers, it is expected that Caprera activities will be designed to accomplish a significant amount of work as compared to host transactions. Hence it is necessary to provide a mechanism in Caprera to facilitate *forward recovery*[4] of failed Caprera activities. Furthermore, since Caprera activities are long-lived, it may be useful to support partial backout or *backward recovery*[5] of Caprera activities.

Viewing a Caprera activity as a sequential program (for example, a shell script or a C++ program), the progress of an activity should be logged at every primitive statement level in this sequential program. Logging at such a fine granularity will enable the Caprera users to minimize the amount of lost work as a result of failures or rollbacks of activities. Unfortunately, statement-level logging results in a significant overhead, since the program state must be saved on stable storage after the execution of every statement. Caprera strikes a balance in the tradeoff between logging overhead versus lost work due to failures and rollback by structuring Caprera activities as a sequence of coarse logical stages.

A Caprera activity is a distributed object that embodies business rules and data. It is distributed because the execution of the Caprera activity involves multiple sites in the network. For example, a typical activity may be initiated at a Caprera server, followed by data collection or extraction from the host database. After the activity is ready, it may then migrate to an assigned Caprera client, which may be a mobile or remote platform, where the associated tasks are performed by the Caprera user. Once the tasks are completed by the user at a Caprera client site, the activity migrates back to the server for termination. The Caprera server terminates the activity by incorporating the updates into the host database.

The various stages of an activity in Caprera are as follows:

[2] The current implementation ensures that the data read from the host database is indeed the same at commit time.

[3] Although it is expected that most Caprera activities will be structured in this manner, the current implementation is not constrained to adhere to this design philosophy. If an application chooses to update the host database in the middle of an activity it is permitted in the current Caprera implementation. This approach, however, weakens the failure recovery guarantees provided by Caprera and must be strengthened at the application level.

[4] A resumption of a failed transaction from an intermediate point of execution is referred to as a forward recovery.

[5] When an execution of a transaction is rolled back to a prior control point - it is referred to a backward recovery. Most database management systems support a simple notion of backward recovery by backing out a transaction completely when it is aborted.

- Prolog: This stage of the activity involves initializing the state and the meta-information associated with the activity.
- Assignment: At this stage the input parameters, the business rules, and the initial state of the activity are used to assign the activity to an appropriate Caprera client. The assignment determines the client where the task associated with the activity will be performed.
- Extraction: The input parameters, the assignment, and the business rules are used to extract the data that will be needed from the host database to execute the activity.
- Client Task(s): At this stage, the activity migrates to the client machine where the client tasks are performed.
- Update: The activity migrates back to the Caprera server and business rules are used to update the data in the host database using the data that was returned from the client machine.
- Epilog: The activity is completed after performing all the necessary clean up.

Failure recovery in Caprera involves providing resiliency to both site and communication failures. A local persistent store (LPS) is used to log the run-time information associated with Caprera activities. Transactional queuing mechanisms are used to deal with communication failures between Caprera clients and servers. In addition, control mechanisms are available to partially or completely rollback an activity. A user-initiated abort or rollback request is admitted either to rollback an activity completely or to the beginning of any of the six stages in the activity.

Caprera Architecture

Caprera is a client/server application-development environment for building transaction-oriented applications. An activity is the basic building block of a Caprera application. Applications are built as a collection of persistent objects in the Caprera system. These objects are activities, subset tables, views, rules, and events. Subset tables define the data subsets that are selectively replicated to the clients using the activity framework. Views are the form definitions for presenting the subset data on the client. Rules and events are the triggers for initiating activities on the client or the server and can be used to design workflows in the application. Activities are transactional objects that encapsulate the business rules using a Java-like scripting language called CapreraScript. Activity objects use subset tables, views, rules, and event objects to represent a business process.

Figure 3 represents the layered architecture of a Caprera server. Each of the blocks in the diagram represents a major functional module of Caprera. The persistent store for all of the Caprera objects is a relational database called Local Persistent Store (LPS). LPS is also the keeper of the activity log on the server as well as the client. All six stages of an activity are logged into the LPS, so that each stage can be recovered or rolled back in case of failure. All interactions with the LPS, as well as the host database, are handled by Database Manager. Database Manager provides a number of native and ODBC adapters to the host databases. Database Manager makes use of the concurrency control and failure recovery mechanisms provided by the LPS as well as the host database. All the Caprera objects, like activities, subset tables, views, rules, and events, are maintained as persistent objects in the LPS through the Persistent Object Manager.

CapreraServer

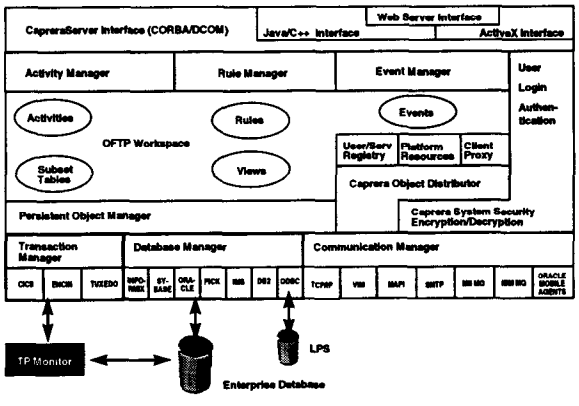

Figure 3

Activity Manager is primarily responsible for scheduling the activities on the server. Each stage of the activity is executed in its own process space. Activity Manager can perform load balancing by distributing activity execution to multiple servers on the network using a scheme based on the owner of the activity. Dynamic load balancing is achieved by distributing the activities to other servers based on the load of the currently executing activities. Activity Manager supports forward recovery by providing a framework for executing compensating activities in case of data conflicts while checking data into host DBMS from the client.

Caprera Object Distributor is responsible for distributing the activity execution between the server and client. After the execution of the Prolog, Assignment, and Extraction stages, the activity is handed over to Caprera Object Distributor for distribution to the client. Caprera Object Distributor uses the Security Manager to encrypt activity objects using RSA public/private encryption algorithms and then hands over the secure activity object

to Communication Manager. Communication Manager transports the secured/compressed activity object to the client using a transactional messaging queue built on top of SMTP, MAPI, VIM, or TCP/IP. Communication Manager provides adapters for a number of communication, as well as transactional messaging, protocols.

CapreraClient

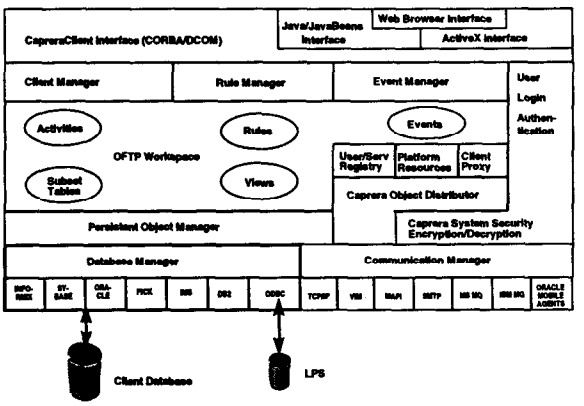

Figure 4

Figure 4 shows the layered architectural diagram of a Caprera client. There are a number of modules that are common between the server and the client. The counterpart of Activity Manager on the client is Client Manager. Client Manager receives the activity object from client Communication Manager, uncompresses and decrypts the activity object and brings the activity object back to life. Then the task stage of the activity is executed on the client. On completion of the client task, the activity is shipped back to the server securely. At the server, the Update and the Epilog stages of the activity are scheduled for execution by the Activity Manager.

The CapreraScript engine is a virtual machine that executes the code contained in the activity object. CapreraScript is a Java-like scripting language that provides constructs for developing transactional applications.

Security Manager uses algorithms from the RSA suite to provide extensive security mechanisms for transporting activities over a public data network. Caprera supports user login authentication while invoking the server or the client.

Transaction Manager provides the interface support for TP monitors with applications that need the additional transactional guarantees of a TP Monitor.

Inter-application communication is provided through DCOM and CORBA. Caprera client components can be integrated into applications using ActiveX and JavaBeans.

Future releases of Caprera will also provide a Java-based interface for embedding Caprera client functionalities in a Web browser. The Web browser will obtain the Web contents from Caprera server through CGI hooks between Caprera server and the Web Server.

Concluding Remarks

In summary, Caprera is a commercial product that provides host database connectivity to mobile and remote users over a wide-area network. We refer to this as OFTP (*off-line transaction processing*) in contrast to the traditional OLTP. The foundation of Caprera transaction and recovery architecture and the activity model is based on traditional database theory and concepts.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems,* Addison Wesley, Reading, Massachusetts, 1987.

[2] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pages 249--259, May 1987.

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

[4] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems,* 6(2):213-226, June 1981.

[5] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A Transaction Recovery Method supporting fine-granularity Locking and Partial Rollbacks using Write-ahead Logging. *ACM Transactions on Database Systems,* 17(1):94-162, March 1992.

[6] A. Reuter and H. Wachter. The ConTract Model. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications,* pages 219-263, Morgan Kaufamann Publishers, 1992.