OSF DCE SIG | S. Dietzen (Transarc)
Request For Comments: 53.0 | R. Fleming (DEC)
January 1994

REQUIREMENTS FOR TRANSACTION PROCESSING WITH DCE
(Report of the DCE SIG Transaction Processing Working Group)

1. INTRODUCTION

This paper specifies a set of requirements for the support of distributed transaction processing built on the Distributed Computing Environment (DCE). It reflects the discussions conducted within the Transaction Processing Working Group of the DCE SIG and the mail exchanged by the members of this group. The scope of these requirements is limited to an initial offering, but it also lays out the framework for other work in this area.

2. JUSTIFICATION

In order for DCE to be successful in the commercial marketplace it must address the requirements for building distributed transaction processing (DTP) systems. Transaction integrity is important to those commercial applications (like banking, financial, airline, etc.) which cannot afford any loss or inconsistency in data. Even in a centralized application, TP methodologies are necessary to these applications.

As applications become more distributed and the number of points of failure increase, transaction integrity becomes even more important. In the absence of DTP techniques, it is left to the application designers and implementors to worry about data integrity. This adds an unacceptable burden in a small system, and makes the construction and maintenance of large ones infeasible.

While DCE provides an excellent base of technology, it is by no means sufficient for DTP applications. Large commercial systems in the key industries will not be able to give up the transaction protection they currently have and hence will be unable to implement distributed computing until DTP is available. Through the adoption of core transaction processing support, the OSF enables independent application developers and TP product vendors to build and interconnect transactional clients and servers, resource managers, logs, TP monitors, alternative communication gateways, additional higher-level programming interfaces, management tools, etc. The incorporation within the DCE of the basic DTP extensions proposed herein facilitates interoperability for the broad range of DCE-based DTP technologies that will evolve over time.

This paper describes how to augment existing DCE systems with transaction semantics in a way that is consistent with the relevant standards in this area, in particular those of X/Open (see [TxRPC]). This paper recommends use of the X/Open TxRPC API and protocols as described in Section 3.2.

3. REQUIREMENTS

3.1. Transactional Semantics

The primary purpose for extending DCE RPC is to provide transactional semantics, thus creating a transactional RPC (TxRPC). That is, the At-Most-Once semantics, currently supported by DCE RPC, will be augmented so that a unit of work can possess ACID properties when distributed using RPCs. ACID properties are defined as follows:

(a) Atomicity — Transactions are ‘‘all or nothing’’. In this way, both programmers and users need not be concerned that failures could cause data to be left in an inconsistent state.

(b) Consistency — The data updates made by transactions preserve the integrity of data by mapping one consistent state to another.

(c) Isolation (or serializability) — Concurrent transactions behave as if they were executed in series. This means that application developers are freed from considering potential complex interleavings of their concurrent computations, and users can be more confident in the correctness of the resulting software.

(d) Durability (or permanence) — Once done, transactions are not undone. Therefore, users and developers are assured that critical modifications to data will not be lost by subsequent system failures.

These properties are preserved across many types of failures, including: communications outages, system crashes and application failures.

Under a typical scenario for DCE DTP, a client application begins a transaction, performs multiple accesses to some number of servers using transactional RPCs, and then ends the transaction. The client and participating servers may also access local resource managers. When the client application ends a distributed transaction, the system polls each of the participating servers (via a commit protocol) to determine whether updates associated with that transaction should be made permanent. If any participant is unable to do the required computation, all the work is undone (i.e., aborted, or rolled back). Otherwise, the transaction and the associated updates are committed.

Note that the use of transactional semantics are an option, not a requirement. Programmers can choose to use the existing At-Most-Once semantics as opposed to transactional semantics if they so desire.

3.2. X/Open TxRPC and DCE RPC

The X/Open Distributed Transaction Processing Group (XTP) has specified a transactional RPC consisting of:

(a) An API that is OSF DCE RPC with extensions to the IDL.

(b) Use of the OSI TP protocol.

Complete details of the API and protocol may be found in [TxRPC].

Because of the large and growing installed base of DCE customers, we also need a DCE-based protocol for transaction processing. Some customers will accept an OSI-based solution, but existing customers would prefer a DCE-based protocol for transaction processing that would work with all existing DCE implementations. A DCE-based protocol leverages DCE RPC, security, and naming to flow transactions. There are currently examples of such implementations in the market.

The OSF DCE requirements for distributed transaction processing are:

(a) An API as specified in the X/Open TxRPC specification.

(b) Support for a DCE-based protocol for transactional RPC.

(c) When market demand and resources exist, support for an OSI TP-based protocol for transactional RPC as specified in the X/Open TxRPC specification.

3.3. Structure and Completeness

The extensions to DCE RPC to make it transactional must be complete and modular. It must be possible to use the DCE TxRPC with a an X/Open Distributed Transaction Processing (DTP) compliant Resource Manager (RM) without having to develop any additional software, except, of course the application. Alternatively, it must be possible to integrate the DCE TxRPC with existing Transaction Processing (TP) systems. To achieve the latter the design must be modular and the interfaces to the modules must be rigorously specified. Figure 1 shows a diagram of a system structured to meet these requirements. Shown in Figure 1 are:

(a) Application (AP) — The user provided application which leverages the transaction manager to define transactions and which uses DCE RPC facilities to ship requests to servers.

(b) Resource Manager (RM) — The manager of resources (e.g., a database system) provided by the local system.

(c) RPC Stubs — The DCE RPC stubs generated from the IDL source code. These stubs are augmented with provisions for shipping transactional context over TxRPC interfaces. Such changes are upwardly compatible with the present DCE RPC.

(d) RPC Runtime — Runtime support for the RPC stubs.

(e) Transaction Manager (TM) — The manager of transactions guaranteeing the ACID properties over distributed applications. The TM provides the transaction context that is piggybacked on transactional RPCs, as well as that which is included within the two-phase commit flows.

(f) Communications Manager (CM) — The manager of communications. The CM provides for the sending of transactional RPCs and is responsible for shipping the two-phase commit protocol among transactional participants. Note: The TM is completely independent of the communications mechanism used, while the CM is independent of the transaction state information it ships. (As an aside, the TM should be able to operate with multiple CMs so that alternative communication technologies (e.g., peer-to-peer) can be easily integrated.)

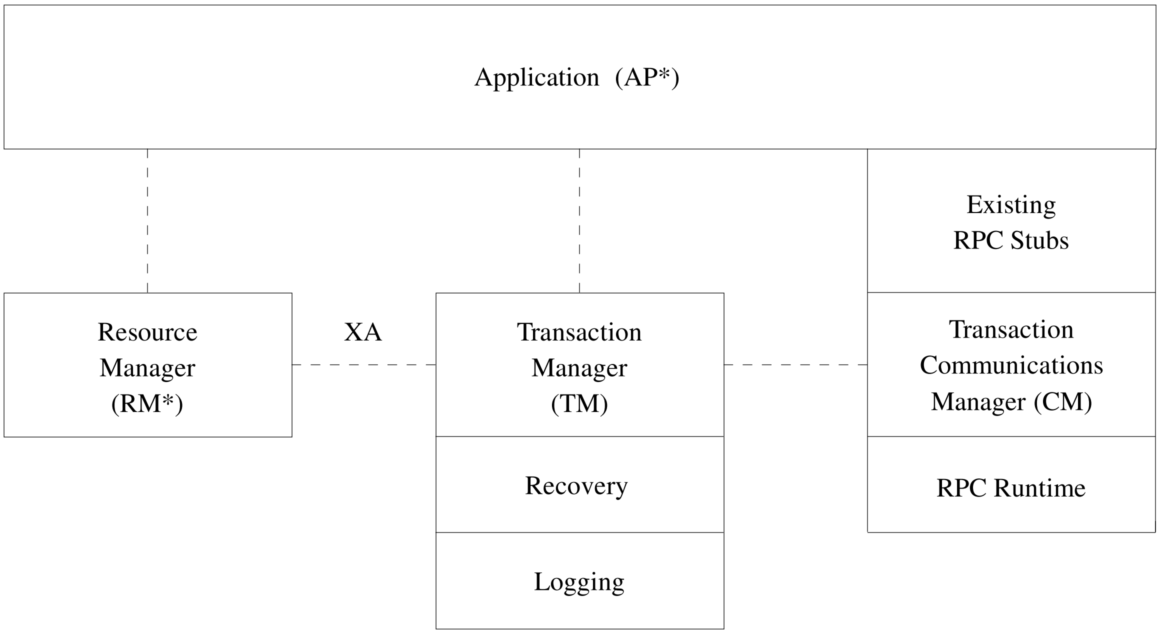
(g) Logging component — The stable storage abstraction for recording transaction state and outcomes. The logging facility must be optimized to provide state-of-the-art performance on both light and heavily loaded systems. It is not necessary that this logging facility be capable of providing RM logging. However, the design should be open in that it permits ‘‘common’’ logging, wherein another log (e.g., an RM log) is used in place of the DCE log for efficiency.

(h) Recovery service — The manager for system restart and abort processing that replays the log to invoke the appropriate undo/redo logic within the participating resource managers. The recovery manager may be viewed as an internal component of the TM, but a clearly defined TM/recovery interface permits the integration of specialized recovery mechanisms (beyond the provisions of the XA interface, which is the name for the transactional interface between the RM and TM).

The structure just described meets the requirement of being complete but also of lending itself to being integrated with an existing TP system. Other structures are possible.

In Figure 1, the components that are provided by applications are marked with a ‘‘*’’ — all other components are provided by the system.

FIGURE 1



3.4. Interface Specifications

The interfaces in Figure 1 that must be formally documented are discussed in this section. Specified interfaces allow DCE TxRPC to be integrated into a system as a single unit, and it allows DCE TxRPC to be integrated with an existing system by augmenting DCE with the new components illustrated in Figure 1.

The interfaces requiring formal specifications are:

(a) TM/RM interface (see [XA]).

(b) TM/AP interface (see [TX]).

(c) TM/CM interface.

(d) RPC Stubs interface to CM and RPC Runtime.

(e) Selected interfaces within the TM. If the TM is structured with a separate Recovery Service (RS), the TM/RS and the RS/Logging service interfaces must be specified. If the RS is embedded within the TM and/or logging service, it is necessary to specify the TM/Logging service interface.

(f) The distributed transaction service should provide a management interface for querying transaction state and for resolving blocked transactions by making heuristic decisions. The TM should also be extensible in that systems can associate special computations (e.g., application callbacks) with transaction state transitions (prepare, commit, and abort).

4. DESIRABLE FEATURES

The following are enhancements which go beyond what is specified by XTP. Since these enhancements are also under consideration in XTP and since the desire is to have a standards-based solution, OSF is encouraged to work with XTP to address the features described below.

4.1. Nested Transactions

In programming multi-threaded applications on top of the DCE, the developer must ensure that concurrent threads do not conflict in their access to shared data. This suggests a synergistic relationship between the transaction construct and threading primitives, since the transaction model defines concurrency control constructs that support the ACID properties. By assigning a transaction (or multiple, sequential transactions) to each individual thread, intra-application resource contention is handled in the same manner as inter-application.

The use of the nested transaction model provides protection for multi-threaded applications that are cooperating in the work of a concurrently executing transaction. Consider that a given multi-threaded procedure may itself be an atomic unit of work concurrently sharing data with other applications. What is needed in this situation is the notion of a hierarchy of transactions in which sub-transactions individually contend for resources. Nested transactions provide that hierarchy: a nested transaction only commits relative to its parent; if the parent aborts, any work associated with a child is rolled back as well.

Another reason that nested transactions should be supported is to provide failure isolation. Consider that as TP applications become highly distributed, the likelihood is increased that a failure within a particular server or communication will result in a global rollback of computation. To avoid such scenarios, nested transactions are employed to allow particular servers to locally isolate and recover from failures without effecting other transaction participants. Through nested transactions, developers have a uniform means by which to trap exception conditions. This can most clearly be seen for the case in which servers are themselves the clients of other servers: without nested transactions, the programmer of such server applications is prohibited from using the transaction concept, unless he guarantees that any invoking client is non-transactional. But even then, this same restriction would apply to other servers invoked by a transactional server.

As the above discussion illustrates, nesting becomes very important as distributed transactions become ‘‘deeper’’ — i.e., when servers act as the clients of other servers — and ‘‘broader’’ — i.e., as greater numbers of servers participate. Without nesting, alternative ad hoc failure recovery mechanisms must be developed by the designer. Nested transactions offer a common facility for addressing the complexities associated with concurrency and failure recovery in a distributed environment.

4.2. Coordinator Migration

There exists a window of vulnerability in the two-phase commit protocol: once a server has prepared but before it is informed of the transaction outcome, it gives up its ability to abort the transaction, and thereby gives up control over its data (i.e., it agrees to keep all relevant data locked pending the transaction outcome). Should the transaction coordinator become unreachable during this window, access to potentially critical data is blocked.

This vulnerability can be substantially reduced by migrating the role of transaction coordinator from clients to more reliable servers. In this way, we reduce the likelihood that the coordinating machine will be inadvertently shut down or rebooted at an inappropriate time. Moreover, this allows ‘‘ephemeral’’ clients — those without any facility for logging transaction results (such as diskless PCs and workstations) — to begin and end transactions. Additionally, through coordinator migration, transaction participants can suggest or require that a particular server act as the coordinator. And hence, a server managing critical data can demand to serve as the coordinator so that it need never relinquish control of that data.

5. ACKNOWLEDGEMENTS

Members of the DCE SIG Transaction Processing Working Group contributed substantially to this document, both at SIG meetings and over the sig-dce-tp@osf.org mailgroup. In fact, the named authors of this document are largely place-holders: although we produced the initial draft and took care of editorial matters, the members of the Working Group were largely responsible for the substance of the final version.

REFERENCES

[TX] Distributed Transaction Processing: The TX (Transaction Demarcation) Specification, X/Open Preliminary Specification, ISBN 1-872630-65-0, P209, October 1992, X/Open Company Ltd. (Apex Plaza, Forbury Rd., Reading, Berkshire, RG11AX, UK; Internet email: XoSpecs@xopen.co.uk).

[TxRPC] Distributed Transaction Processing: The TxRPC Specification, X/Open Preliminary Specification, ISBN 1-85912-000-8, July 1993, X/Open Company Ltd. (Apex Plaza, Forbury Rd., Reading, Berkshire, RG11AX, UK; Internet email: XoSpecs@xopen.co.uk).

[XA] Distributed Transaction Processing: The XA Specification, X/Open CAE Specification, ISBN 1-872630-24-3, C193, October 1991, X/Open Company Ltd. (Apex Plaza, Forbury Rd., Reading, Berkshire, RG11AX, UK; Internet email: XoSpecs@xopen.co.uk).

AUTHORS' ADDRESSES

Scott Dietzen
Transarc Corp.
707 Grant Street
Pittsburgh, PA 15219
USA
Internet email: dietzen@transarc.com
Telephone: +1-412-338-4439

Robert Fleming
Digital Equipment Corp.
151 Taylor Street
Littleton, MA 01460
USA
Internet email: fleming@olcrow.enet.dec.com
Telephone: +1-508-952-4267