Open DECdtm: Constraint Based Transaction Management

Digital Equipment
Activity Management Group
Johannes Klein, Francis Upton IV[1]

Open DECdtm offers portable transaction management services layered on OSF DCE which support the application (TX), resource manager (XA), and transactional DCE RPC (TxRPC) interfaces specified by X/Open. Open DECdtm also provides interoperability with OSI Transaction Processing (OSI TP) and OpenVMS systems using the DECdtm OpenVMS protocol. Protocols executed by Open DECdtm are specified by constraints. This simplifies the development of transactional gateways between different data transfer protocols and transaction models.

1. Introduction

Open DECdtm offers portable transaction management services layered on OSF DCE which support the application (TX), resource manager (XA), and transactional DCE RPC (TxRPC) interfaces specified by X/Open[6,7,8]. Open DECdtm also provides interoperability with OSI Transaction Processing (OSI TP)[5] and OpenVMS systems using the DECdtm OpenVMS protocol[4]. Protocols executed by Open DECdtm are specified by constraints. A constraint based implementation simplifies the development of transactional gateways between different data transfer protocols and allows for support of different transaction models in a heterogeneous environment.

Transaction management protocols, application demarcation services and resource manager interfaces are well understood and available in a variety of products. This is documented by formal and industry standards as well as implementations such as Tandem's TMF, Digital's DECdtm and RTR, etc [2]. Most transaction management protocols are defined in the context of particular data transfer protocols. Today no agreement has been reached of how to manage transactions across different data transfer models (i.e. request/response, peer-to-peer, queued) and supporting different transaction coordination models.

Most transaction management implementations offer private interfaces to integrate with data transfer protocols. No general agreement has yet been reached on how those interfaces should look.[2] These issues have to be resolved to allow applications to use different data transfer protocols when executing a transaction. To address those issues and allow applications to use a mix of different data transfer models, Open DECdtm supports synchronization between transaction state and data transfer state.

Synchronization issues particularly arise if supporting peer-to-peer data transfer across multiple transaction protocols. For applications using these services transaction management services have to ensure that on commitment all database updates are recorded on stable and all messages sent are consumed. For that reason the transaction manager may need to reject commit requests or abort transactions.

Take as an example polarized control (half-duplex) dialogues (conversations) as defined by OSI TP or IBM's SNA LU 6.2[5]. At any time only one application is allowed to send data on a dialogue. An application issuing the request to commit must have the right to send on all its dialogues. This ensures that all data is received before an application commits. The transaction manager should reject any commit request if issued prematurely. If this "state check" is not handled by the transaction manager transactions would have to be aborted to ensure message integrity.

[1] The views presented in this paper are those of the authors. The paper does not constitute a commitment by Digital to any specific product.

[2] Although X/Open is working on extending the resource manager interface to handle transactional communication protocols, this extension is done in the context of supporting only OSI TP.

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0430...$1.50

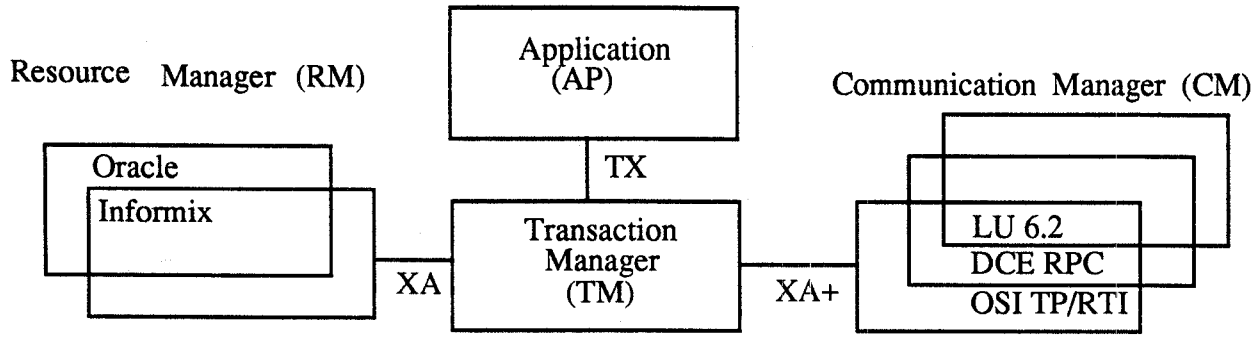

Figure 1 X/Open System Model

Current proposals for communication manager interfaces such as XA+ do not account for those issues. They only address synchronization issues between transaction branches.

Today most transaction management protocols only provide support for flat transactions. Those providing support for nested transactions cannot use their extended functionality when coordinating transactions in a heterogeneous environment. Open DECdtm provides nested transaction support in a manner that interoperates with flat transaction protocols by representing each nested transaction as a flat transaction and providing the necessary coordination locally. This is as a first step towards support for different transaction models. Currently concurrency control issues cannot be solved in a heterogeneous environment and therefore are not specifically addressed by Open DECdtm.

The following sections provide an overview about the architecture of Open DECdtm, a brief introduction to the specification of transaction management protocols and issues in synchronizing with data transfer protocols and transaction management protocols.

2. Model and Protocols

Transaction processing systems distinguish between four different components as shown in Figure 1. Applications (AP) execute user programs. Applications access local resource managers (RM) using languages such as SQL. Applications access remote resource managers through services offered by remote applications. Invocation of remote services is supported by communication managers(CM). Transaction managers (TM) ensure that local and remote resource managers agree on the same outcome of a transaction. Transaction managers must also ensure that all messages sent by participating applications are processed before allowing a transaction to commit. Protocols are specified to define the correct behavior of each component.

Internally, Open DECdtm models transaction participants as resources. Each resource is managed by one of the above components. Examples of resources are an AP thread, an RM instance, or a specific branch managed by a CM. Resources change state when processing a transaction. Those state transitions are called events. Protocols define conditions on the occurrence of events. Open DECdtm uses constraints to define these conditions[3]. Messages are exchanged between components to indicate the state of a given event and resolve conditions. Messages indicate that events have become TRUE, have become FALSE, or are promised to become TRUE at some later point. Examples of events include *commit*, indicating the transaction has committed, *request*, indicating the desire to begin transaction termination, and *token*, indicating the right to send data on a given communication link.

Constraints are defined using *when-then-enable* statements

when <condition>
then [enable] <action>.

Conditions and actions refer to events. All conditions associated with an event must be satisfied before the event becomes enabled, i.e. the event may only become TRUE once it has been enabled, but being enables does not imply the event will become true. A set of sample constraints used by Open DECdtm are shown below.

When starting a transaction a constraint is established between application and transaction manager. The application is identified as *superior*, the transaction manager is identified as *subordinate*.

```
constraint Application(Superior,Subordinate)
  {
    // start termination after issuing tx_commit
    when done(Superior) then
          request(Subordinate);
    // on commit enable "ok"
    when commit(Subordinate) then
          ok(Superior);
  }
```

Constraint *Application* defines that commitment starts after the application completed processing. On commit the application event *ok* becomes enabled indicating the return of control to the AP.
When invoking a resource manager or a remote service the *Participant* constraint is created. Resource managers or remote transaction managers are called *subordinates*.

```
constraint Participant(Superior,Subordinate)
  {
  // start termination when receiving a request
  when request(Superior) then
            request(Subordinate);

  // commit when requested and superior agreed
  when request(Superior) and commit(Superior)
        then commit(Subordinate);

  // commit when subordinate agreed
  when commit(Subordinate) then
            commit(Superior);
  }
```

The constraint *Participant* identifies that a subordinate may only commit after requested by its *superior*. Both commitment of the *superior* as well as the commitment of the *subordinate* are mutually dependent. Messages necessary to achieve common agreement are automatically derived from this constraint specification.
If at runtime state transitions cause updates to recoverable resources journaling actions are automatically triggered after analyzing the state of affected constraints. Optimizations such as *read-only, volatile, last-participant and one-phase* are automatically derived ensuring optimal behavior of Open DECdtm.

3. Communication Management

When dealing with transactional communication, there are two important entities:

1. The transaction branch which represents the state of the remote transaction participant.
2. The connection which represents the state of the communication link to a remote application.

Current proposal for communication manager interfaces such as XA+ only deal with issues related to branch management. However, it is useful to treat connection and branch management separately because of the different nature of their events and their different lifetimes. Failure of a connection may cause abort of ongoing local branches but causes recovery for local branches which are promised to *commit* or *committed*. The current state of a connection also determines when commitment can be started or whether premature request for commitment should cause the transaction to be aborted or cause a state check.
Branches have the traditional transaction events, e.g. *commit, request* and live until the transaction termination protocol has completed. Connections have events associated with them such as *com_fail*, indicating the connection has failed, and *token*, indicating that data may be sent on the connection. Some connections are only associated with a single branch[5] others are multiplexed between transaction branches[4].
Another important aspect in the relationship between connections and branches has already been identified. In case of polarized control [5] branches are only allowed to start commitment if all connections are in the *send* state. The transaction manager has to prevent any attempt by the AP to commit a transaction if any of its associated connections are not in send state. This ensures that all messages sent are processed before commitment starts.

4. System Architecture

Open DECdtm provides portable transaction services integrated with OSF DCE RPC and threads. When designing a transaction manager the following questions have to be answered, (1) whether to maintain transaction state in the application process or a separate server process and, (2) whether to piggypack coordination messages on user requests or to establish separate communication links. Each approach has its advantages and disadvantages. Over time transaction managers must be able to accommodate all of them either for interoperabiliy, security or performance reasons.
Open DECdtm uses a toolkit-like approach. Library routines are provided for constraint management. Shells interfacing to constraint management services offer interfaces as required by TX, XA, DECdtm for OpenVMS protocol, and OSI TP protocol specifications. Constraint management services are integrated with a common journal server[1]. Thus the same code base is used to support different approaches.
Open DECdtm maintains transaction state in the address space of individual applications and piggypacks coordination messages on RPC requests exchanged between client and server application. This avoids process switching for maintaining transaction state

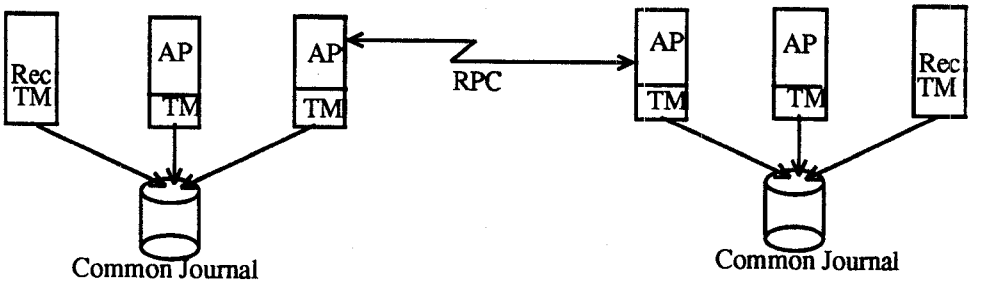

Figure 2 System Architecture

and allows Open DECdtm to take advantage of ongoing application communication.

Each Open DECdtm node has a journaling and recovery daemon. The recovery daemon resolves transactions after process and node failures. During normal processing transaction state is spooled to a common journal service. The common journal service allows the recovery daemon to retrieve information about in-doubt transactions and decided but unacknowledged transactions ensuring their timely resolution.

5. Summary

Open DECdtm offers portable transaction services across OSF and VMS platforms. Open DECdtm is layered on OSF DCE with support for TxRPC, XA, and TX as specified by X/Open. Open DECdtm supports the DECdtm OpenVMS for VAX protocol and provides interoperability with OSI TP. Open DECdtm's protocols are specified using constraints. This simplifies the task of interoperating with different communication protocols and provides an extensible infrastructure for new transaction models and work flow systems.

Acknowledgments

The authors would like to thank Dieter Gawlick for creating an exciting working environment, Bob Taylor for guiding us through difficult times, Wayne Duquaine for doing a first prototype, Edward Cheng and Mike Depledge for providing a journal service and Ron Obermarck for implementing the first specification compiler.

References

[1] Cheng, E. *A High-speed Open Journal in a Distributed Computing Environment*, Proceedings ICSC, December 1992.

[2] Gray, J., Reuter, A., *Transaction Processing : Concepts and Techniques*, Morgan Kaufman. 1992.

[3] Klein, J., *Advanced Rule Driven Transaction Management*,, IEEE COMPCON, February 1991.

[4] Laing, W., Johnson, J.E., Landau, R.V, Transaction Management Support in the VMS Operating System Kernel, *Digital Technical Journal*, Winter 1991.

[5] Upton IV, Francis, OSI Distributed Transaction Processing, an Overview, International High Performance Transaction Processing Workshop, 1991.

[6] X/Open Company, *Preliminary Specification, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification*, 1992.

[7] X/Open Company, *Preliminary Specification, Distributed Transaction Processing: TxRPC Specification*, 1993.

[8] X/Open Company, *CAE Specification. Distributed Transaction Processing: The XA Specification*, 1991.