APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

Training

ANSAwise - CORBA Transactions

Chris Mayers

Abstract

Organizations require that new systems based on CORBA distributed object technology provide the same kinds of transactional guarantees as those provided by traditional transaction processing monitors (TPMs) and database systems.

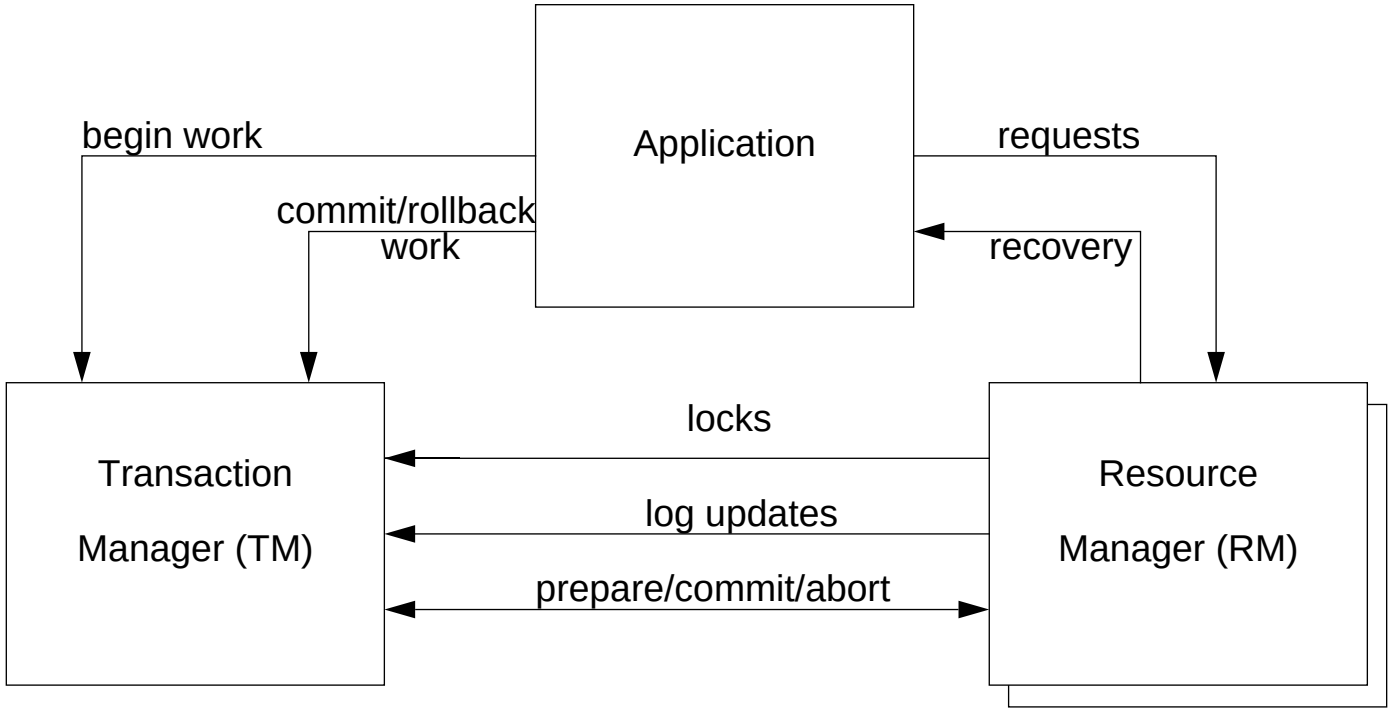
This module of the ANSAwise training programme briefly describes the X/Open distributed transaction processing (DTP) model, then outlines the CORBA Object Transaction Service (OTS).

[This is a variant of APM.1629 and APM.1461. It does not cover traditional database systems.]

APM.1802.01	**Approved**	9th July 1996
	Briefing Note	

Distribution:
Supersedes:
Superseded by:

CORBA Transactions



In this session

- **Clarify the significance of transactions in structuring applications**
- **Examine open standards in transaction processing**
- **Describe the CORBA Object Transaction Service (OTS)**

Transaction Processing Requirements Review

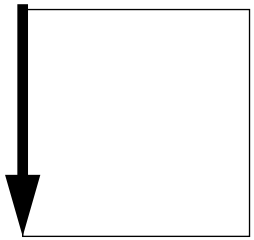
- The world's first large distributed systems were TP systems
- Scope is the end-to-end system, not just data
- High availability (fast recovery)
- High performance (soft real-time)
- Online and batch transactions concurrently
- ACID (all-or-nothing, serializable) properties

Transactions - the ACID properties

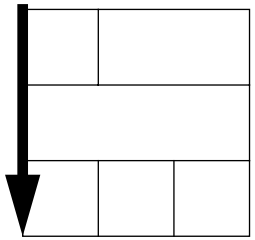
- ***A*tomicity**
  - **all-or-nothing**
- ***C*onsistency**
  - **transactions must be self-contained logical units of work**
- ***I*solation**
  - **concurrent transactions must not affect each other**
- ***D*urability**
  - **what's done must not be undone**

Flat and Nested Transactions

- **Flat Transaction**



- **Nested Transaction (with subtransactions)**

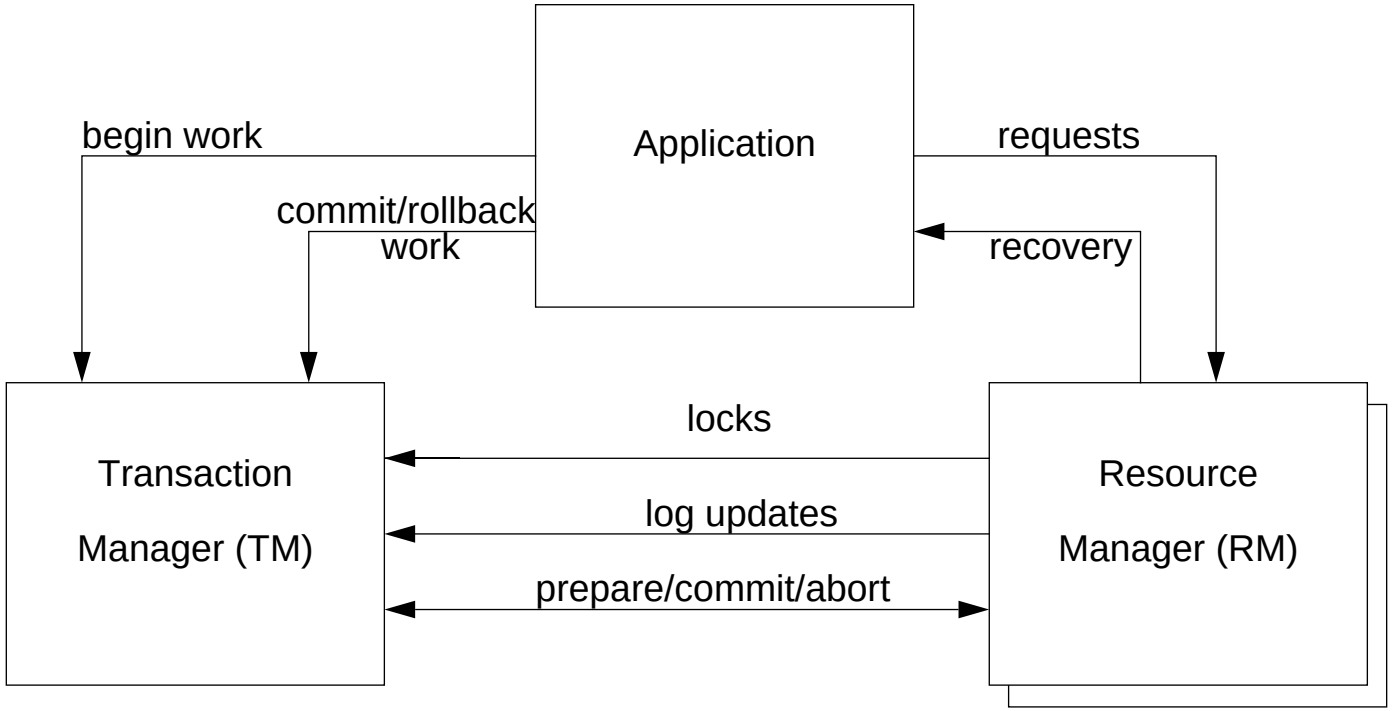


Distributed Transaction Processing

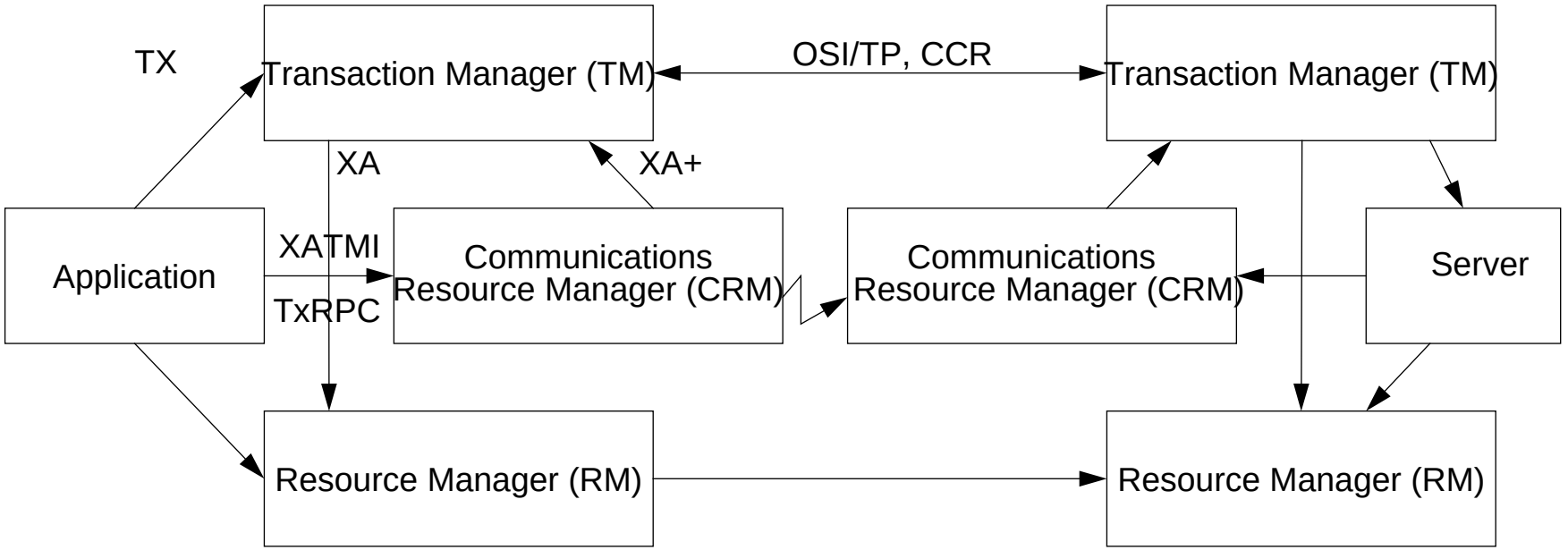
- **In a distributed system, how are transactions involving multiple databases coordinated?**

- **What about transactions involving operations other than database update?**

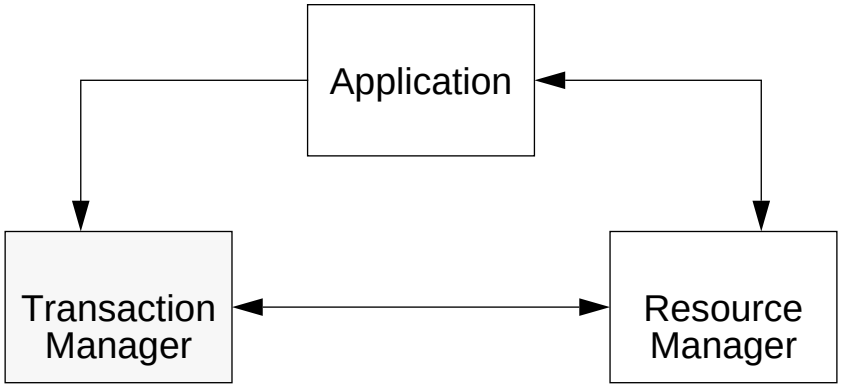
Typical Open TP Core Components (much simplified)



X/Open and ISO TP Model



Example Transaction Managers



- **Encina (Transarc, now owned by IBM)**

- **Tuxedo (Unix Systems Laboratories, now owned by Novell)**

Encina (IBM/Transarc)

- **Application Interface - Transactional C (proprietary language extension)**
- **Transaction Model - Nested**
- **Communications - Transactional RPC (TRPC)**
- **Infrastructure - OSF DCE**

Tuxedo (Novell/Unix Systems Laboratories)

- Application Interface - ATMI (C, C++, COBOL, 4GLs)
- Transaction Model - Flat
- Communications - Tuxedo
- Infrastructure - Most Unix

The ACID properties in CORBA

- ***A*tomicity**
  - **supported by the Transaction service**
- ***C*onsistency**
  - **...is the responsibility of your application**
- ***I*solation**
  - **supported by the Concurrency service**
- ***D*urability**
  - **supported by the Persistence service**

Using the CORBA Transaction service

- You can use the Transaction service without the Concurrency service or Persistence service
  - and vice-versa
- What might you use as alternatives to the Concurrency service and Persistence Service?
- When might you use the Concurrency service, but not the Transaction service?

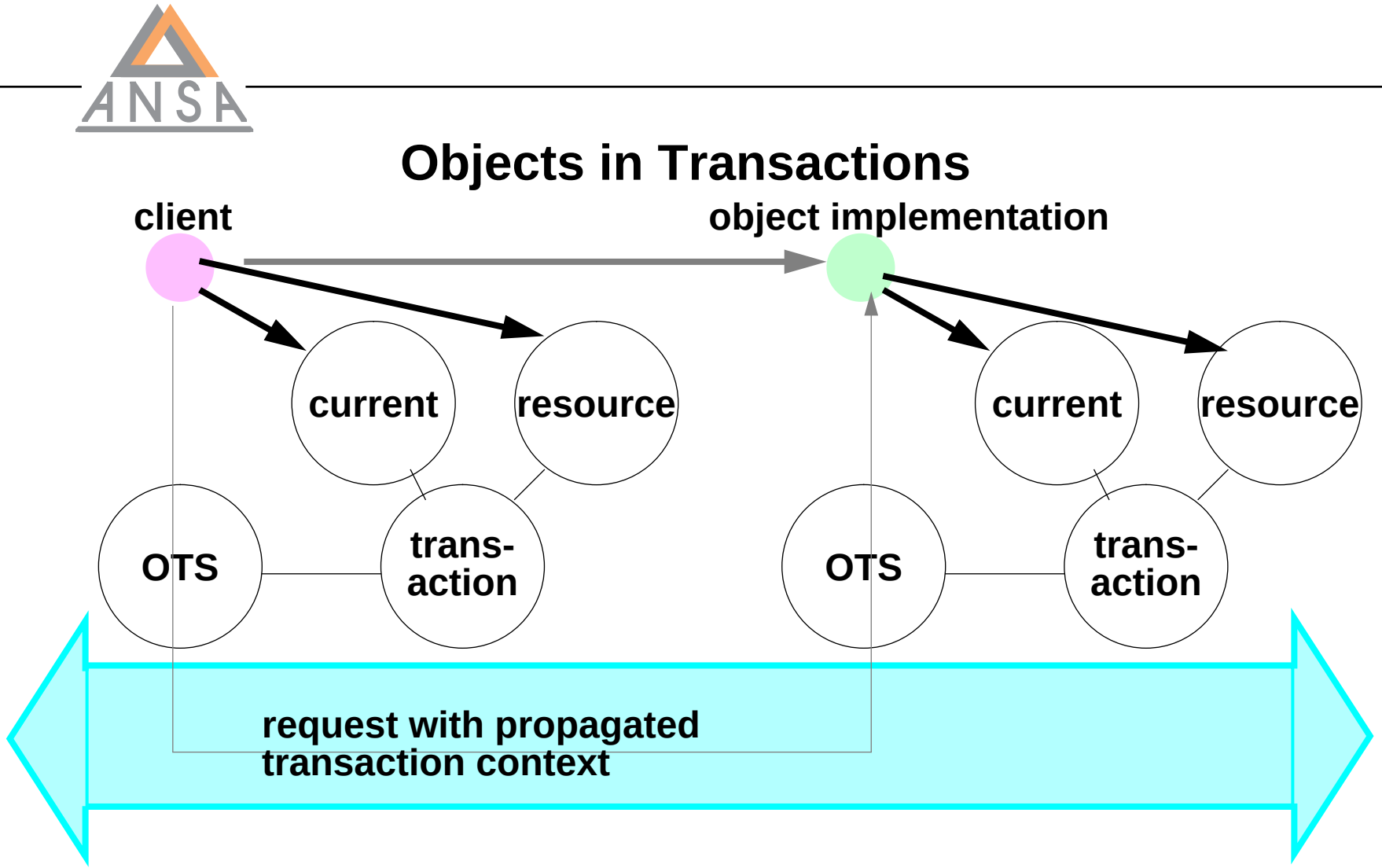
Get ready to discuss this

CORBA Transaction Service - Starting Points

- Based on the X/Open DTP model
- Supports flat and nested (optional) transactions
- Supports transparent and non-transparent use
- Assumes a two-phase commit
- Is a framework for transactions

CORBA Transaction Framework

- The CORBA Transaction service does not do all the work for you
  - it simply coordinates transactions
- Transactional Clients must demarcate their transactions
  - using `begin`, `commit`, `rollback`
- Transactional Servers may force rollback
- Recoverable Servers must also register their resources and handle commits and rollbacks



Starting a Transaction

- Transactions are (naturally) objects
- In the simplest case, transactional clients invoke `begin`, `commit`, `rollback`, and `set_timeout` on the `Current` pseudo-object
  - the `Current` object is automatically associated with the current thread of execution
  - subsequent requests implicitly propagate the transaction context of `Current`
- Clients can also manage contexts directly, and propagate contexts explicitly

Registering Resources

- **Remember that the CORBA Transaction service does not know of the existence or scope of resources**
  - **resources must register themselves explicitly**
- **Resources must take care to register themselves exactly once with each transaction**
  - **using the `is_same_transaction` operation**

Transaction Termination

- Any participant can force the transaction to roll back
  - all participants must then roll back their changes
- A Transaction Service implementation may
  - allow an object other than the originator to commit the transaction
  - monitor the participants for failure or inactivity

Two-Phase Commit

- The coordinator is asked to commit the transaction
- In phase 1, the coordinator asks resources to prepare to commit
  - each resource returns its Vote....
  - ... `VoteReadOnly`
  - ... `VoteCommit`
  - ... `VoteRollback`
- In phase 2, the coordinator tells each resource the result of the vote (commit or rollback)
  - and the resource must act accordingly

Optimizations

- If any resource returns `VoteRollback`, the coordinator does not ask the remaining resources for their votes

- If there is only one resource involved, the coordinator invokes `commit_one_phase` instead

Transaction Integrity

- The CORBA Transaction service supports checked and unchecked behavior
- Checked behavior
  - prevents premature commit while threads are active in a transaction, or if there are operation invocations outstanding
  - ... Transaction service enforces integrity
- Unchecked behavior
  - supports heuristic decisions
  - ...Application must enforce integrity

Heuristic Decisions

- In extreme situations one or more participants may unilaterally decide to commit or roll back after commit phase 1
  - each can make a heuristic decision
- Typically this is done to handle communications failure
- A participant making a heuristic decision may (unfortunately) decide differently from the consensus
  - the CORBA Transaction service will report a heuristic exception
- The application must then repair or compensate for the effects of an incorrect decision

Heuristic Exceptions

- `HeuristicRollback()`
  - a `commit` operation resulted in all updates being rolled back
- `HeuristicCommit()`
  - a `rollback` operation resulted in all relevant updates being committed
- `HeuristicMixed()`
  - some relevant updates were committed, others were rolled back
- `HeuristicHazard()`
  - some relevant updates were unknown; others were all committed or all rolled back

Points to watch

- **Beware**
  - **Expense of distributed transactions**
  - **Subtransaction support**
  - **Mixing transactional and non-transactional modes**
  - **Restrictions on the coordinator**
  - **Integrating other transaction systems**

Distributed Transactions Are Expensive

- The two-phase-commit protocol is inherently slow
- Conclusions
  - avoid distributed transactions if at all possible
  - exploit locality
  - if performance is still likely to be unacceptable, seek specialist advice

Subtransaction Support

- Remember that subtransactions are optional
- Few systems support nested transactions
- Subtransactions must not roll back unilaterally
- The CORBA Transaction service does not completely specify subtransactions
- Conclusions
  - avoid!

Mixing Modes

- **Do not mix explicit and implicit propagation within the same transaction**
  - **the transaction context may be propagated when you do not expect it**
- **Conclusions**
  - **keep it simple**

Restrictions On The Coordinator

- Two-phase commit only allows for a single coordinator
- The coordinator cannot move around the transaction tree
- Some environments cannot reasonably support the coordinator
- Conclusions
  - confirm with your supplier how desktops are supported

Integrating Other Transaction Systems

- Some systems still don’t have an open commit or recovery protocol
- Some systems don’t communicate lock conflicts externally
- Systems that cannot be changed must retain their own autonomy
- Still need to maintain global and local serializability
- Gateways may need to maintain transaction state
- Conclusions
  - get the vendor to do the work!

CORBA Transaction Service - Implementations

- **Possible implementations**
  - **on top of an existing TP monitor**
  - **integrated with XA-compliant Resource Managers**
  - **integrated with ORB binding mechanisms (and the CORBA Event service)**
  - **integrated with ORB-specific mechanisms**
  - **integrated with an ODBMS Object Adapter**

Future Trends

- **Asynchronous interactions**
  - **using the forthcoming CORBA Asynchronous Messaging service**
- **Long-lived transactions**
  - **Sagas, workflows,...**
- **Transactions in the World Wide Web**

Summary

- The CORBA Transaction Service is an Object Service
  - see *CORBAservices* by the Object Management Group (Wiley)
- Specifies transactional and recoverable objects
  - similar to Resource Managers in the X/Open DTP model
- Tracks object usage automatically across threads
- Supports flat and nested transactions
- Supports legacy TP standards and protocols
  - see Appendix A of the Transaction Service Specification