



Strategies for Integrating Messaging and Distributed Object Transactions

Stefan Tai and Isabelle Rouvellou



T.J. Watson Research Center, New York, U.S.A.

stai@us.ibm.com, isabelle@watson.ibm.com





Introduction

- **Distributed middleware** is an essential element in the development of enterprise-scale software systems
- The communication paradigms of object-orientation and messaging define the two main classes of distributed middleware that exist today:
object-oriented middleware (OOM) and **message-oriented middleware (MOM)**
- **Transactional OOM** (e.g. CORBA OTMs, EJB) support transaction processing among distributed objects
- **Messaging services for OOM** (e.g. CORBA Messaging, Java Message Service JMS) are currently being introduced
 - ▶ to allow for asynchronous messaging among distributed objects
 - ▶ to support integration with MOM and other MOM-based applications





Introduction

- Integrating messaging and distributed transactions is non-trivial
 - ▶ Building the middleware
 - ▶ Using the middleware
- Transactional asynchronous messaging is a well-recognized problem
 - ▶ OOM distributed transaction processing is understood, but *difficult*
 - ▶ MOM asynchronous messaging is understood, but OOM asynchronous messaging is still in its *experimental* stage
 - ▶ Messaging properties and qualities affect transactions; this has *not been fully explored*





Presentation Overview

1. Messaging Classification Framework

- ▶ Definition of a **taxonomy** and common language to better communicate about messaging

2. Study of Messaging Middleware

- ▶ Understanding different messaging middleware and the **messaging architectural styles** that they induce

3. Strategies for Integrating Messaging and Distributed Object Transactions

- ▶ Identification and proposal of **four integration strategies**

4. Summary and Discussion

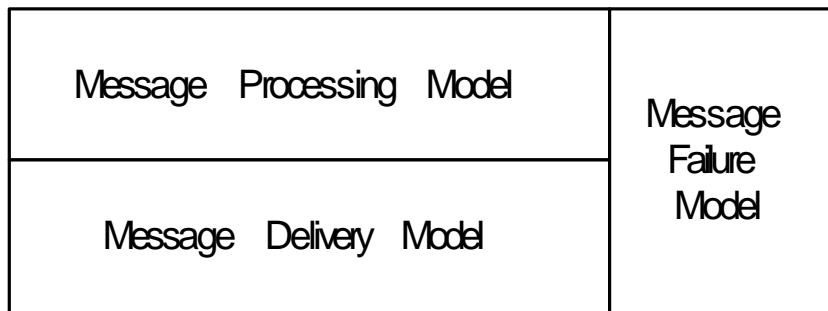




What is Messaging?

- ▶ *"Programs communicating by putting messages on queues"*
 - ▶ *"Multicast event notification"*
 - ▶ *"Processing requests asynchronously"*
- Different **message delivery** and **message processing models**; with various properties and qualities of messaging
 - Understanding the **desired** and/or **middleware-supported messaging model** is critical for the integration with distributed transactions

Messaging Classification Framework





Message Delivery Model (1/3)

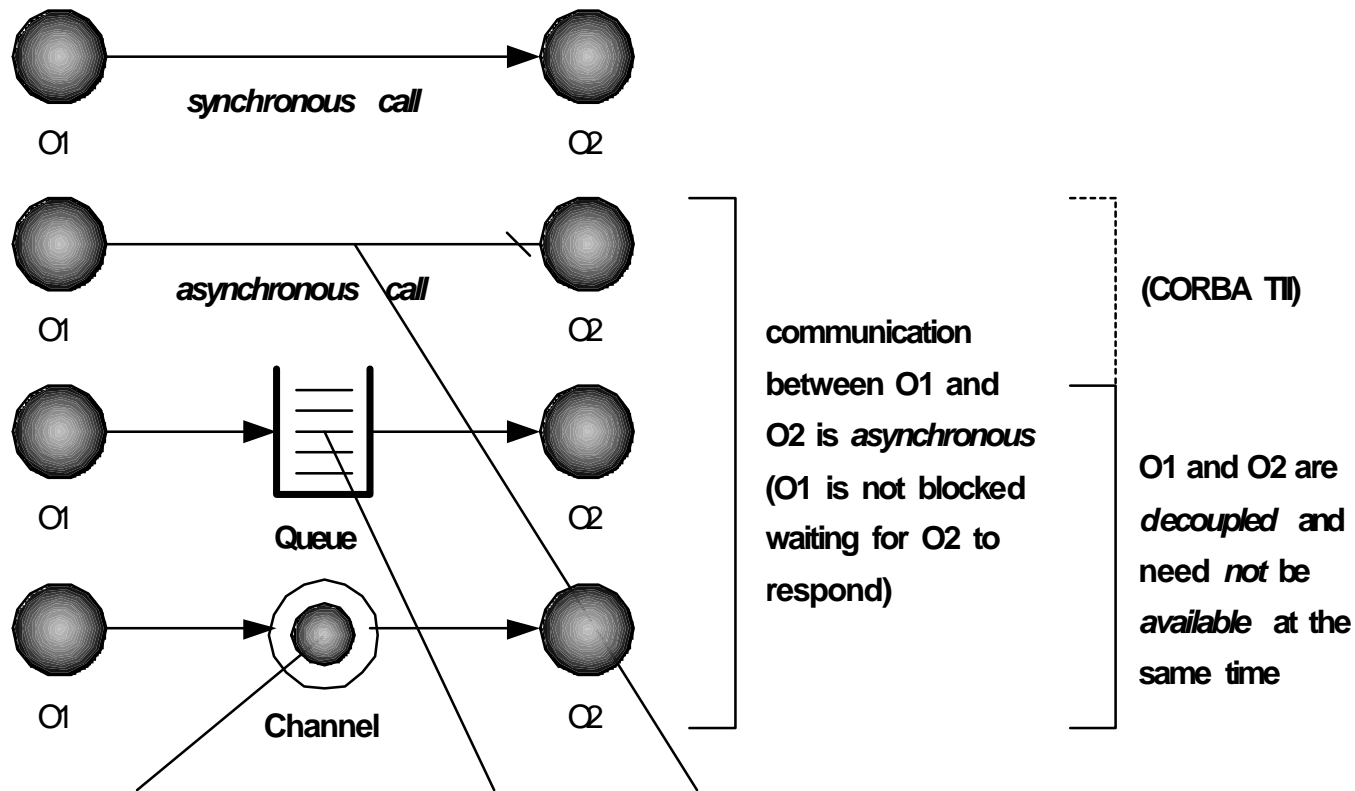
✚ Properties of exchanging messages for (one-way) *event notification*

1. Representation	2. Messaging API	3. Initiation	4. Intermediation
1. as object	1. application-independent	1. by producer	1. none
2. as data	2. application-specific	2. by consumer	2. exactly one
3. operation invocation	3. combination of 1. and 2.	3. push & pull	3. multiple

5. Multiplicity	6. Anonymity	7. Subscription
1. unicast	1. known set	1. yes
2. multicast	2. unknown set	2. no
3. broadcast	3. mixed	3. mixed



Synchronicity/Asynchronicity (2/3)



Message as Object Message as Data Message as Invocation





-
-
-
-
-
-
-
-
-
-

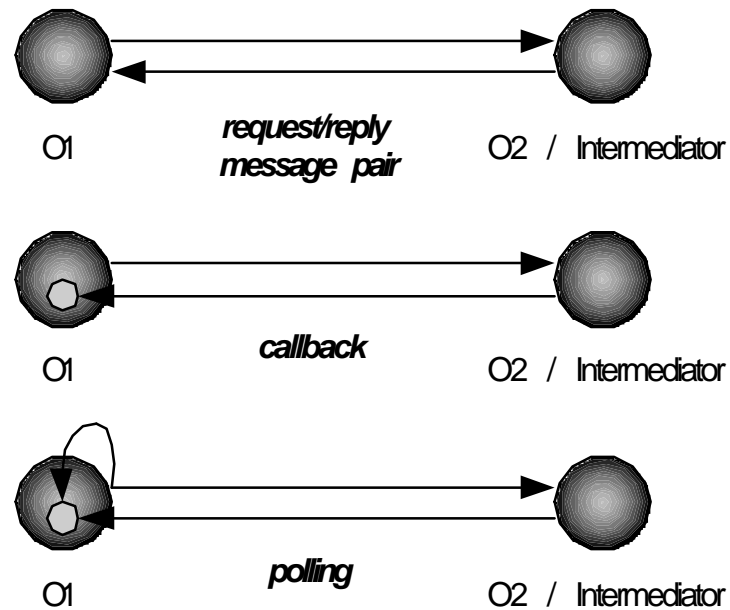
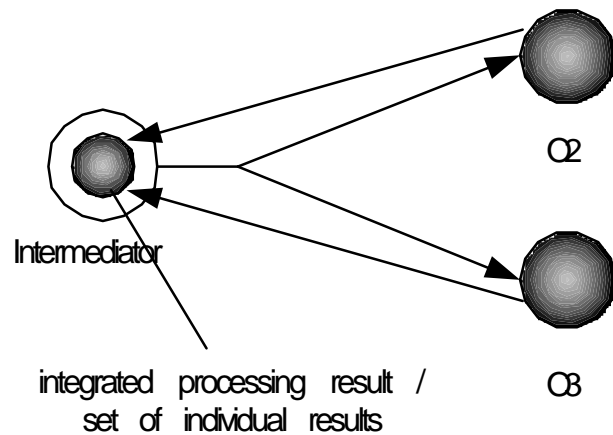
Message Delivery Model (2/3)

8. Synchronicity	9. Delivery Guarantee	10. Persistence	11. Ordering	12. Filtering
1. synchronous	1. best effort	1. persistent	1. FIFO	1. none
2. asynchronous	2. at-most-once	2. transient	2. LIFO	2. header/type
	3. exactly-once	3. either 1. or 2.	3. random	3. body/content
	4. either 2. or 3.		4. priority-based	4. 2. and 3.



Message Processing Model (1/2)

➤ Properties of **message processing**: messaging for (two-way) *request processing*





Message Processing Model (2/2)

13. Processing Result	14. Communication
1. single return value	1. separate reply message
2. single integrated return value	2. callback
3. set of individual return values	3. polling





Message Failure Model

➤ Properties of [message failure](#)

F1. Failure Level	F2. Failure Scope	F3. Failure Detection
1. delivery	1. particular recipients	1. exceptions
2. processing	2. number of any recipients	2. timeout of ack/reply
	3. all recipients	3. combination of 1. and 2.






Middleware Comparison (1/2)

Middleware	1. Representation	2. Messaging API	3. Initiation	4. Intermediation	5. Multiplicity	6. Anonymity	7. Subscription	8. Synchronicity
CORBA Messaging	3. invocation	2. specific	1. by producer	1. none	1. unicast	1. known set	2. no	2. asynchronous
CORBA Events untyped	2. data	1. independent	2. push & pull	3. multiple	3. both	2. unknown set	1. yes	1. synchronous
CORBA Events typed	3. invocation	3. combination	2. push & pull	3. multiple	3. both	2. unknown set	1. yes	1. synchronous
CORBA Notification	2. data	1. independent	2. push & pull	3. multiple	3. both	2. unknown set	1. yes	1. synchronous
JMS Messaging	1. object	1. independent	2. push & pull	3. multiple	3. both	3. mixed	3. mixed	1. synchronous
MQ Messaging	2. data	1. independent	2. push & pull	3. multiple	3. both	2. unknown set	2. no	1. synchronous





Middleware Comparison (2/2)

9. Delivery Guarantee	10. Persistence	11. Ordering	12. Filtering	13. Proc. Results	14. Communication	F1. Failure Level	F2. Failure Scope	F3. Failure Detection
4. either 2. or 3.	3. either 1. or 2.	1. FIFO / 4. priority	1. none	1. single value	2. callback/ 3. polling	2. processing	1. particular	3. combination
2. at-most-once	2. transient (*)	undefined	1. none	n/a	n/a	1. delivery	n/a	n/a
2. at-most-once	2. transient (*)	undefined	1. none	n/a	n/a	1. delivery	n/a	n/a
4. either 2. or 3.	3. either 1. or 2.	undefined	4. 2. and 3.	n/a	n/a	1. delivery	n/a	n/a
4. either 2. or 3.	3. either 1. or 2.	1. FIFO / 4. priority	2. header/type	3. set of values	1. separate message	2. processing	1. particular	3. combination
4. either 2. or 3.	3. either 1. or 2.	1. FIFO / 4. priority	2. header/type	3. set of values	1. separate message	2. processing	1. particular	3. combination





Integration Strategies

■ Distributed Object Transaction Processing

- ▶ ACID transactions
- ▶ Transaction service and monitor
 - Transaction demarcation and management API
 - Distributed transaction context propagation
 - Integration of diverse persistent stores (XA resources)
 - Two-phase-commit

■ Integrating four different messaging models

✚ MQ-Integrating Transactions

✚ Message Delivery Transactions

✚ Message Processing Transactions

✚ Full Messaging Transactions





MQ-Integrating Transactions

- Intent

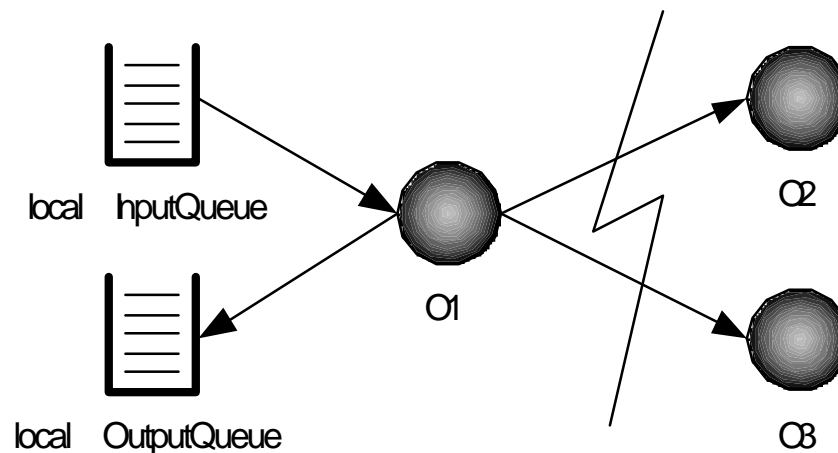
Integrate **message queues** of common MQ-systems as persistent resource managers into the transaction

- Concept

Local dequeuing and enqueueing of **messages as transactional data** in combination with distributed object invocations

- Implementation

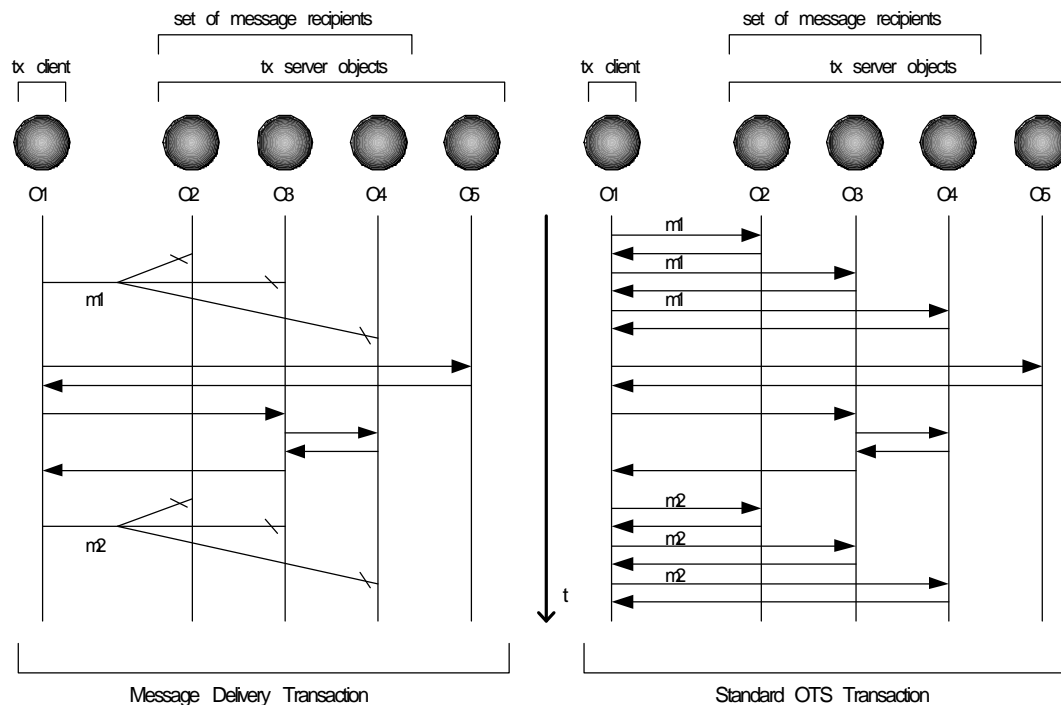
Through X/Open **XA Resource Manager** interface of MQ-systems



Message Delivery Transactions (1/2)

■ Intent

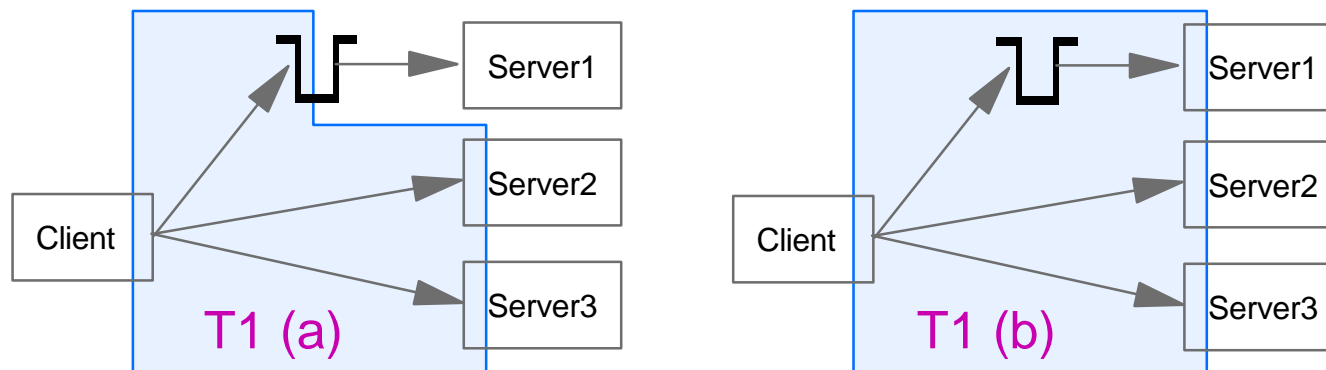
- ▶ Enable **event notification** (possibly multicast) between remote communication partners within the scope of a synchronous ACID-like transaction
- ▶ Messages are sent at **any point** during the transaction, and **delivery to ultimate recipients** is monitored



Message Delivery Transactions (2/2)

■ Concept

- ▶ Extending the transaction service to handle asynchronous messaging to (a) queues and (b) ultimate recipients behind queues
- ▶ Introducing a **message failure model** of message failure definition, message failure observation and evaluation
- ▶ **Run-time message compensation** in case of transaction failure



■ Implementation

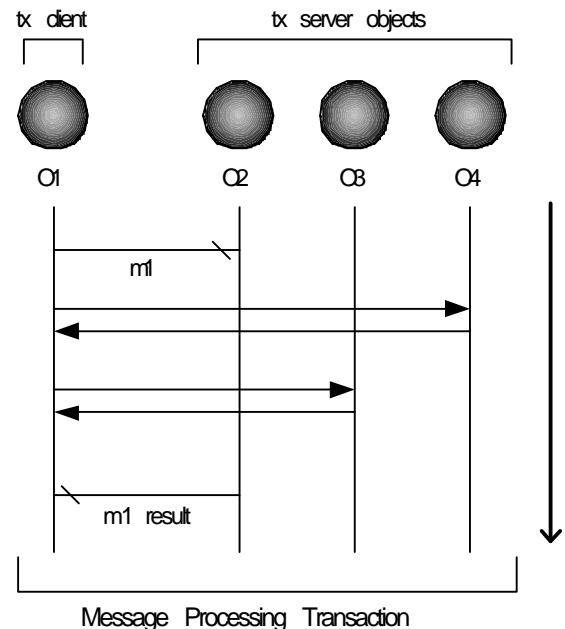
- ▶ Not supported with any OOM messaging service
- ▶ *Currently under development at IBM Watson*



Message Processing Transactions (1/2)

Intent

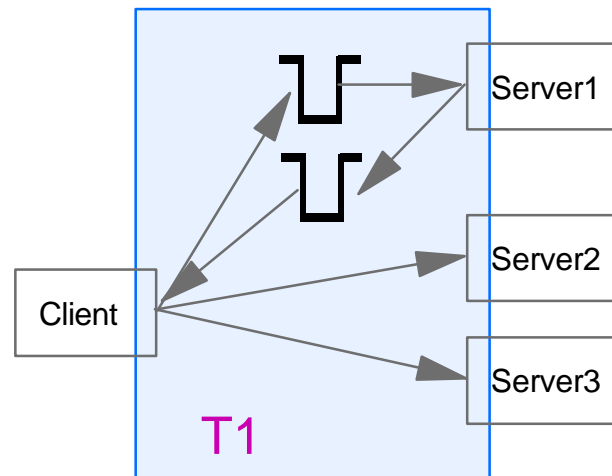
- ▶ Enable **asynchronous request processing** between remote communication partners within the scope of a synchronous ACID-like transaction
- ▶ Messages are sent at **any point** during the transaction, **delivery to ultimate recipients** is monitored, and **results of processing** must be returned ultimately at commit time based on bounded asynchronicity



Message Processing Transaction (2/2)

■ Concept

- ▶ *Message delivery transactions* +
- ▶ Time-out *evaluation function* for messages carrying results correlated to request messages
- ▶ *Transaction context propagation* with *time-independent* process lifetimes





Full Messaging Transactions

- Intent

The "right" combination of message delivery transactions and message processing transactions





Summary and Discussion

■ Messaging Classification Framework

- ▶ Definition of a **taxonomy** of messaging concepts

■ Messaging Middleware Comparison

- ▶ Identification of **messaging architectural styles**
- ▶ ... *affecting standard distributed object transactions*

■ New transaction models: "messaging transactions"

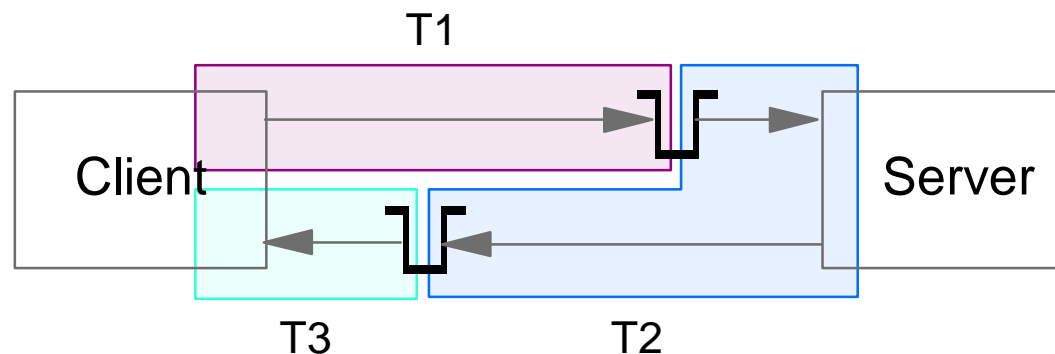
- ▶ Towards better integration of distributed transactions and asynchronous messaging
 - "Relaxing" ACIDity
- ▶ Advancing the MOM state-of-the-art
 - Ability to **send remote messages during a transaction** before commit-time
 - Making the delivery of messages **critical** to the success of a transaction (making a **transaction dependent on messaging and vice versa**)
- ▶ Advancing the OOM state-of-the-art
 - **Combining the two communication models** of synchronous object invocations and asynchronous messaging within a single transaction scope



Additional Slides

MOM Transactional Asynchronous Messaging

- MOM allows to group a set of produced and consumed messages into one atomic unit of work using explicit transaction demarcation
 - ▶ Messages are put on local queues, and messages can be received from local queues
 - ▶ Messages cannot be received from remote queues
 - ▶ Outgoing messages to remote queues are not delivered before transaction commit
- A distributed transaction typically requires three transactions of the above kind





State-of-the-Art OOM Transactional Asynchronous Messaging

- The **JMS** supports a similar transaction model to that of MOM
- **CORBA Messaging** distinguishes **shared** and **unshared transactions**
 - ▶ Shared transactions have an end-to-end transaction semantics (can be mapped to standard OTS transactions)
 - ▶ Unshared transactions include time-independent invocations (toleration of server non-availability), are not supported
- The integration of OOM message services with OOM transaction services is mostly **undefined or unsupported**
 - How does messaging relate to
 - ▶ Transaction demarcation?
 - ▶ Distributed transaction context propagation?
 - ▶ Integration of diverse persistent data stores?
 - ▶ Commit protocol?

