Lower Level Techniques

- What do you do if you don’t have a fully functional container?
  - No Distributed Transactions
  - No Replicated App Server State
  - Fake it!
Sessions and Session State

• Recall the elegant J2EE HttpSession class

```java
public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
throws ServletException, IOException {
    ... 
    HttpSession session = request.getSession(true);  
        // true => create new session if none exists
    ...
    session.setAttribute(attrName, attrValueObject);
    ...
    SomeClass theAttr
        = (SomeClass)session.getAttribute(attrName);
    ...
}
```
HttpSession Class

- Easy to use

- Container manages session data. So container must support replicating session data among multiple app servers. Many containers don’t do this (yet)
Reliable Multicast Solution

- Use SPREAD to replicate among app servers
- Simple way does not give locking / serializable semantics
- Scalability issues -- every app server must track every session state change
Two Other Places

- Two other places to keep session state:
  - In cookies exchanged between client and server
    - limited space -- 4k per cookie
  - In the database
    - performance
Hybrid Approach

• Session state is queried more frequently than it is updated
• Keep session state in database
• Use cookie as a readonly cache of frequently accessed part of session state
• App server is completely stateless!
```
private static final String sessionCookieName = "MyApplicationSession";

public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    ... 
    String rawSessionData = null;
    Cookie[] cookies = request.getCookies();
    if( cookies != null ) {
        for( int i = 0; i < cookies.length; i++ ){
            Cookie c = cookies[i];
            if( c.getName().equals(sessionCookieName) ) {
                rawSessionData = c.getValue();
                break;
            }
        }
    }
    if( rawSessionData == null ) {
        // create a new session
    } else {
        // decrypt and parse data for existing session
    }
    ...
```
private static final String sessionCookieName
    = "MyApplicationSession";

public void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    ...
    String rawSessionData
        = ... // convert session data to encrypted String
    Cookie c = new Cookie(sessionCookieName, rawSessionData);
    c.setMaxAge(sessionTimeout);
    response.addCookie(c);
    ...
}
In Data Base

• Keep Session State as Entity Bean (!)
• Set the isolation level to SERIALIZABLE
• Instantiate the SessionState EJB only for operations that will modify data or read data not in the cookie
• Result is serializable!
In Data Base

```java
InitialContext ctx = new InitialContext();
UserTransaction ut = ctx.getUserTransaction();
ut.begin();
SesStateEntityBeanHome ssh = (SesStateEntityBeanHome)
        .ctx.lookup(sessionStateHomeName);
SesStateEntityBean ssb = ssf.findByPrimaryKey(sessionId);
...
ut.commit();
```
To Think About ...

- How do you garbage collect the abandoned session states in the database?
Other Amenities

- Message Queues
- Distributed Transactions
An Order Processing Protocol

- Web Site takes orders
  - coordinator
- Billing department
  - Visa transactions in real time
- Shipping department
  - schedule
  - ship
  - delivered
- Goal: do not lose transactions or stall
Easy Solution

- Use a big hammer ...
  - transactional message queues
  - distributed transactions
Low Level Solution

• Assumptions
  • individual apps are transactional
  • apps can detect/eliminate duplicates (until explicitly told a txn may be forgotten)
Low Level Solution

An Order object at the Web Site with a state field (intention)

A globally unique Order ID value

Forward Recovery
Low Level Solution

Billing

Web Site

Shipping

state = do_schedule
commit

state = wait_ship
commit

state = shipped
commit

state = delivered
commit

Forget

Forget

Schedule

Ready to Ship

Delivered

Visa Charge
Forward Recovery

Billing | Web Site | Shipping

- state == do_schedule
  - resend Schedule() call

- state == wait_ship
  - no action

- state == shipped
  - (for dup elimination)

- state == delivered
  - (for dup elimination)
  - proceed to Forget()

- Visa Charge
- Ready to Ship
- Delivered
- Forget
the Delivered() call has returned -- so what thread of control calls Forget()?

may be able to create a thread

an unreliable message queue implementation works here!
Compensation (Using Order State)

Billing  |  Web Site  |  Shipping

- Visa Charge  → Ready to Ship
  
- Ready to Ship  → Schedule
  
- Schedule  → Visa Charge

- Undeliverable  → Forget
  
- Shipped  → Undeliverable
  
- Undelivered  → Shipped

- Refunded  → Undelivered

- Refunded  → Shipped

state = do_schedule
commit

state = wait_ship
commit

state = shipped
commit

state = undelivered
commit

state = refunded
commit