1 Collaborative Work

1.1 Transactions

1.1.1 Problem 11.10

Odds = \( c!/(c!)^s = (c!)^{1-s} \)

The probability that \( n \) of these \( c \) services will commit is \( (c!)^{n(1-s)} \)

Substituting \( c = 3 \), \( n = 3 \) and \( s = 3 \), we get \( P(3) = (3!)^3(1-3) = 0.002\% \)

1.1.2 Problem 11.11

Transactions at CTM:
T1: r1(a), r1(c)
T2: r2(b), r2(d)

Transaction at local site 1:
T3: w3(a), w3(b)

Transaction at local site 2:
T4: w4(c), w4(d)

Transaction at each site:
LDB1: r1(a), c1, w3(a), w3(b), c3, r2(b), c2
LDB2: w4(c), r1(c), c1, r2(d), c2, w4(d), c4
CTM: r1(a), r1(b), r2(c), r2(d)

Conflict graph:

T1 \rightarrow T3 \rightarrow T2
T1 \leftarrow T4 \leftarrow T2
1.1.3 Problem 11.17

A key assumption is that "the compensate transactions will eventually succeed inspite of a failure"

Compensates are needed for the first (n-1) steps. The following compensate transactions are needed:

- Cancel coupling sleeves order \((C_c)\): If coupling sleeves are not yet received and \(V_a\) was executed
- Abort saga \((S_a)\): If hose procuring transaction is cancelled or aborted

Refer to figure 1 for Saga subtransactions ordering.

![Figure 1: Saga subtransactions ordering](image)

\(S_o, H_o, C_o, V_o - \text{Order events for saga, hose and valves}\)
\(H_r, V_r, C_r - \text{Order received events for hose, valve and coupling sleeves}\)
\(S_a, V_a, C_a - \text{Abort events for saga, valves and coupling sleeves}\)

1.2 Coordination

1.2.1 Problem 12.3

Refer to figure 2

1.3 Process

1.3.1 Problem 13.5

Refer to figure 3
1.3.2 Problem 13.7

On failure of Task #2 fails, the following operations are required to restore consistency

- Cancel the fee payment and refund to student. Abort task #5 if it was not committed.
- Cancel the issued bill for tuition and fees. Abort task #4 if it was not committed.
- Ignore task #3 because it is read only.

2 Independent Work

2.1 Transactions

2.1.1 Problem 11.12

CTM does T1: r1(a); w1(c)
CTM does T2: r2(b); w2(d)
LDB1 does T3: w3(a); r3(c) and LDB2 does T3: w3(b); r3(d)

CTM: r1(a); w1(c); c1; r2(b); w2(d); c2
LDB1: w3(a); r1(a); c1; w3(b); c3; r2(b); c2 (does not follow 2PL)
LDB2: w1(c); c1; r3(c); w2(d); c2; r3(d); c3 (2PL enforced)
2.1.2 Problem 11.13

A local database uses strict 2PL for locking.

a) All composed transactions that run on the database take a ticket just before they begin their lock-acquisition phase.
Correct but has a negative impact on concurrency, thus slowing down the system. May deadlock if two composed transactions take tickets in reverse order on the two LDBs

b) All composed transactions that run on the database take a ticket just before they begin their lock-relinquishing phase.
Correct with maximum concurrency possible, hence works faster. May cause a lot of aborts.

c) All composed transactions that run on the database take a ticket just after they release their locks.
Does not produce correct schedules.
2.1.3 Problem 11.18
Let C, V and H denote subtransactions to procure coupling sleeves, valves and hoses respectively

Start dependencies: C, V and H are started simultaneously

Commit dependencies:
C, V and H should commit for the multitransaction to commit. V can commit only after H and C can commit only after H and V have committed.

Failure dependencies:
If H fails undo V and C
If V fails undo C

2.2 Coordination
2.2.1 Problem 12.4

Mark’s solution

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:a="http://student/agent"
xmlns:c="http://nscu.edu/course/enroll"

From the Student Agent to CoordNSCU:

<wsc:CreateCoordinationContext>
  <wsc:ActivationService>
    <wsa:Address>http://nscu.edu/coordnsedu/activation</wsa:Address>
    <a:state>9876</a:state>
  </wsc:ActivationService>
  <wsc:RequestorReference>
    <wsa:Address>http://student/agent</wsa:Address>
  </wsc:RequestorReference>
  <wsc:CoordinationType>
    http://schemas.xmlsoap.org/ws/2002/08/wstx
  </wsc:CoordinationType>
</wsc:CreateCoordinationContext>

From the CoordNSCU to Student Agent:

<wsc:CreateCoordinationContextResponse>
  <wsc:RequestorReference>
    <wsa:Address>http://student/agent</wsa:Address>
  </wsc:RequestorReference>
</wsc:CreateCoordinationContextResponse>
```
From the NSCU Course Enrollment to CoordNSCU:

<wsa:Address>http://nscu.edu/coordncsu/activation</wsa:Address>
<c:state>1234</c:state>

From the CoordNSCU to NSCU Course Enrollment:

<wsa:Address>http://nscu.edu/coordncsu/registration</wsa:Address>
<c:private>1234</c:private>

From the NSCU Course Enrollment to CoordNSCU:

<wsa:Address>http://nscu.edu/coordncsu/activation</wsa:Address>
<c:state>1234</c:state>

From the CoordNSCU to NSCU Course Enrollment:

<wsa:Address>http://nscu.edu/coordncsu/registration</wsa:Address>
<c:private>1234</c:private>
From the CoordNSCU to NSCU Course Enrollment:

2.3 Process

2.3.1 Problem 13.6

Angana’s solution

<process name="custAcct"
    targetNamespace="urn:namespace"
    xmlns: cus="http://customer.com">
    <partners>
        <partner name="manager" serviceLinkType="cus:customerLinkType" myRole="manager"/>
        <partner name="FBIlistProvider" serviceLinkType="cus:FBIlistProviderLinkType"
                partnerRole="FBIlistProvider"/>
    </partners>
    <containers>
<container name="request" messageType="cus:CustInfoMessage"/>
<container name="approvalInfo" messageType="cus:approvalMessage"/>
<container name="FBIresponse" messageType="cus:FBIresponse"/>
</containers>

<sequence name=sequence>
  <receive name="receive" partner="manager" portType="cus:requestMonitorPT"
          operation="requestMonitor" container="request" createInstance="yes">
  </receive>
  <invoke name="invokeFBI" partner="FBIlistProvider" portType="apns:FBIlistPT"
          operation="provideTerroristList" outputContainer="FBIresponse">
  </invoke>
  <reply name="reply" partner="manager" portType="cus:requestMonitorPT"
         operation="requestMonitor" container="response">
  </reply>
</sequence>
</process>