PI10 reporting: Settlement

July 2020

PI9 objective:

- A flexible and configurable settlement service

PI9 objective:

- A flexible and configurable settlement service

SMART?
Er, I’ll get back to you…

Credits where they're due:

- Design
  - ➢Neal Donnen
  - ➢Jo Sowden
- Execution
  - ➢Deon Botha
  - ➢Lazola Lucas
- Testing
  - ➢Eugene Polshchykau
- Documentation
  - ➢Lisa Durbin
- Indispensible
  - ➢Thea de Villiers

Progress so far

- ✓ Different settlement models can be defined for different ledger account types.
- ✓ Required database changes have been made and verified as non-breaking
- ✓ Extend API specification to allow account types to be defined by parties
- ✓ Settlement API has been extended to support settlement model definition
- ✓ A rule for calculating interchange fees can be defined and attached to a processing event.
- ✓ Interchange fee entries are correctly recorded in the switch ledgers.
- Settlement API is extended to support settlement model definition (*in testing*)
- Transfer/account entries which are settled gross pass through a per-transfer settlement process. (*In testing*)
- Revise settlement window aggregation procedures to aggregate only transfers which are settled net (*In development*)
- Revise liquidity check to take account of settled funds (*in development*)

To do

- Exclude liquidity check where not required (*not needed for existing implementations*)
- Check automated position reset (for backwards compatibility)

Settling an individual transfer (gross settlement)

The proposal

1. Add a status of "SETTLED" to the transfer status enumeration.
2. Generate ledger entries to transfer the principal amount of the transfer from the participants' position account to their settlement account.
3. Add a record to the transfer state change table to mark that the transfer has been settled.
4. Add records to the participant position change table for both parties to mark that their position has changed.
5. Modify the positions of the parties to the transfer to back out the effect of the transfer.
6. Modify the participant limits of both parties to increase the creditor's limit, and decrease the debtor's limit, by the amount of the transfer.

Small diversion

- We’ve been assuming that funds are always settled into a single Settlement account.
- This may not be a reliable assumption going forward – we may need to have more than one settlement account for different purposes.
- …so I’ve taken the liberty of adding a settlement account to the settlement model.
- This means that we can fully parameterise the settlement process.

Current state of the transfer

transferId	Participant	Ledger Account	Role Type	Ledger Entry	amount	createdDate
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	POSITION	PAYEE_DFSP	PRINCIPLE_VALUE	-100	2020-04-22 08:49:05
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	POSITION	PAYER_DFSP	PRINCIPLE_VALUE	100	2020-04-22 08:49:05

Transfer the gross settled components to the settlement ledger

```
INSERT INTO transferparticipant
(transferID, participantCurrencyId, transferParticipantRoleTypeId, ledgerEntryTypeId, amount)
(SELECT
           TP.transferId, TP.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount*-1
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
WHERE
           TP.transferId = @ThisTransfer
    AND G.name = 'GROSS’
UNION SELECT
           TP.transferId, PC1.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
    INNER JOIN participantCurrency PC1 ON
PC1.currencyId = PC.currencyId AND PC1.participantId = PC.participantId AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
           TP.transferId = @ThisTransfer AND G.name = 'GROSS');
```

Transfer the gross settled components to the settlement ledger

```
INSERT INTO transferparticipant
(transferID, participantCurrencyId, transferParticipantR                                    )
(SELECT
           TP.transferId, TP.participantCurrencyId, TP.t                                    gerEntryTypeId, TP.amount*-1
FROM
    transferparticipant TP INNER JOIN participa                                              C.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.led          ountTypeId = M.ledgerAccountTypeId
    INNER JOIN settlementgranularit      N M.settlementGranularityId = G.settlementGranularityId
WHERE
           TP.transferI    @ThisTransfer
    AND G.name =  GROSS'
UNION SELECT
           TP.transferId, PC1.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
    INNER JOIN participantCurrency PC1 ON
PC1.currencyId = PC.currencyId AND PC1.participantId = PC.participantId AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
           TP.transferId = @ThisTransfer AND G.name = 'GROSS');
```

We join two queries to make the update

Transfer the gross settled components to the settlement ledger

```
INSERT INTO transferparticipant
(transferID, participantCurrencyId, transferParticipantRoleTypeId,
(SELECT
          TP.transferId, TP.participantCurrencyId, TP.transferPa                              Id, TP.amount*-1
FROM
    transferparticipant TP INNER JOIN participantcurr          ON TP.pa                    = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAcc          ypeId = M.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON    settlementGranularityId = G.settlementGranularityId
WHERE pacs.002.001.11
          TP.transferId = @ThisTransfer
    AND G.name = 'GROSS’
UNION SELECT
          TP.transferId, PC1.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
    INNER JOIN participantCurrency PC1 ON
PC1.currencyId = PC.currencyId AND PC1.participantId = PC.participantId AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
          TP.transferId = @ThisTransfer AND G.name = 'GROSS');
```

The first collects any existing ledger movements which belong to a settlement model which is settled GROSS

Transfer the gross settled components to the settlement ledger

```
INSERT INTO transferparticipant
(transferID, participantCurrencyId, transferParticipantRoleTypeId, ledgerEntryTypeId, amount)
(SELECT
           TP.transferId, TP.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount*-1
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participa      rrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.s
WHERE pacs.002.001.11
           TP.transferId = @ThisTransfer
    AND G.name = 'GROSS'
UNION SELECT
           TP.transferId, PC1.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
    INNER JOIN participantCurrency PC1 ON
PC1.currencyId = PC.currencyId AND PC1.participantId = PC.participantId AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
           TP.transferId = @ThisTransfer AND G.name = 'GROSS');
```

The first reverses the effect of the original ledger posting

Transfer the gross settled components to the settlement ledger

```
INSERT INTO transferparticipant
(transferID, participantCurrencyId, transferParticipantRoleTypeId, ledgerEntryTypeId, amount)
(SELECT
           TP.transferId, TP.participantCurrencyId, TP.transferParticipantRoleTypeId, TP.ledgerEntryTypeId, TP.amount*-1
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.s
WHERE pacs.002.001.11
           TP.transferId = @ThisTransfer
    AND G.name = 'GROSS'
UNION SELECT
           TP.transferId, PC1.participantCurrencyId, TP.transferParticipantRol                TP.ledgerEntryTypeId, TP.amount
FROM
    transferparticipant TP INNER JOIN participantcurrency PC ON T   articipantCurrencyId = PC.participantCurrencyId
    INNER JOIN settlementmodel M ON PC.ledgerAccountTypeI   = PC.ledgerAccountTypeId
    INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
    INNER JOIN participantCurrency PC1 ON
PC1.currencyId = PC.currencyId AND PC1.participantId = PC.participantId AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
           TP.transferId = @ThisTransfer AND G.name = 'GROSS');
```

The second generates a balancing entry for each component in the appropriate settlement account(s)

Results

Original state

transferId	Participant	Ledger Account	Role Type	Ledger Entry	amount	createdDate
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	POSITION	PAYEE_DFSP	PRINCIPLE_VALUE	-100	2020-04-22 08:49:05
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	POSITION	PAYER_DFSP	PRINCIPLE_VALUE	100	2020-04-22 08:49:05

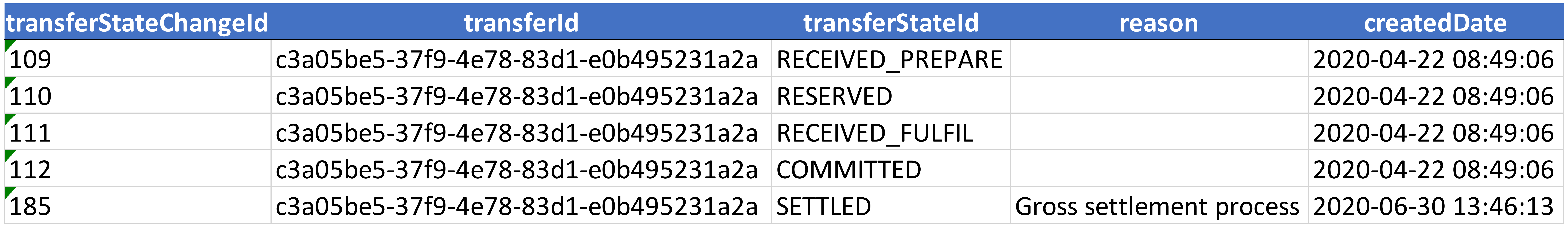
Modified state

transferId	Participant	Ledger Account	Role Type	Ledger Entry	amount	createdDate
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	POSITION	PAYEE_DFSP	PRINCIPLE_VALUE	-100	2020-04-22 08:49:05
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	POSITION	PAYEE_DFSP	PRINCIPLE_VALUE	100	2020-06-30 13:46:13
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	SETTLEMENT	PAYEE_DFSP	PRINCIPLE_VALUE	-100	2020-06-30 13:46:13
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	POSITION	PAYER_DFSP	PRINCIPLE_VALUE	100	2020-04-22 08:49:05
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	POSITION	PAYER_DFSP	PRINCIPLE_VALUE	-100	2020-06-30 13:46:13
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	SETTLEMENT	PAYER_DFSP	PRINCIPLE_VALUE	100	2020-06-30 13:46:13

Mark settlement state

- Add a record to the transfer state change table to mark that the transfer has been settled. (*Note that at present, this table only contains records at the transfer level; but we assume for the moment that this will not be a blocker*.)

```
INSERT INTO transferstatechange(transferId, transferStateId, reason)
  VALUES(@ThisTransfer, 'SETTLED', 'Gross settlement process');
```



transferStateChangeId	transferId	transferStateId	reason	createdDate
109	c3a05be5-37f9-4e78-83d1-e0b495231a2a	RECEIVED_PREPARE		2020-04-22 08:49:06
110	c3a05be5-37f9-4e78-83d1-e0b495231a2a	RESERVED		2020-04-22 08:49:06
111	c3a05be5-37f9-4e78-83d1-e0b495231a2a	RECEIVED_FULFIL		2020-04-22 08:49:06
112	c3a05be5-37f9-4e78-83d1-e0b495231a2a	COMMITTED		2020-04-22 08:49:06
185	c3a05be5-37f9-4e78-83d1-e0b495231a2a	SETTLED	Gross settlement process	2020-06-30 13:46:13

Modify the positions

```
UPDATE participantPosition PP INNER JOIN
(SELECT
  PC.participantCurrencyId, TP.Amount
FROM
  transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
  INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccountTypeId
  INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
  INNER JOIN transferParticipantRoleType R ON TP.transferParticipantRoleTypeId = R.transferParticipantRoleTypeId
WHERE
  TP.transferId = @ThisTransfer AND G.name = 'GROSS' AND R.name = 'PAYER_DFSP'
UNION SELECT
  PC1.participantCurrencyId, TP.amount
FROM
  transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
  INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
  INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
  INNER JOIN participantCurrency PC1 ON
    PC1.currencyId = PC.currencyId
    AND PC1.participantId = PC.participantId
    AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
  TP.transferId = @ThisTransfer AND G.name = 'GROSS')
AS TR ON PP.participantCurrencyId = TR.ParticipantCurrencyId
SET PP.value = PP.value+TR.amount;
```

Modify the positions

We use the same joined query as we did to update the ledgers

```
UPDATE participantPosition PP INNER JOIN
(SELECT
   PC.participantCurrencyId, TP.Amount
FROM
   transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
   INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccountTypeId
   INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
   INNER JOIN transferParticipantRoleType R ON TP.transferParticipantRoleTypeId = R.transferParticipantRoleTypeId
WHERE
   TP.transferId = @ThisTransfer AND G.name = 'GROSS'
UNION SELECT
   PC1.participantCurrencyId, TP.amount
FROM
   transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantCurrencyId = PC.participantCurrencyId
   INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = PC.ledgerAccountTypeId
   INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.settlementGranularityId
   INNER JOIN participantCurrency PC1 ON
      PC1.currencyId = PC.currencyId
      AND PC1.participantId = PC.participantId
      AND PC1.ledgerAccountTypeId = M.settlementAccountId
WHERE
   TP.transferId = @ThisTransfer AND G.name = 'GROSS')
AS TR ON PP.participantCurrencyId = TR.ParticipantCurrencyId
SET PP.value = PP.value+TR.amount;
```

Modify the positions

And we don’t need to worry about reservations

```
UPDATE participantPosition PP INNER JOIN
(SELECT
   PC.participantCurrencyId, TP.Amount
FROM
   transferparticipant TP INNER JOIN participantcurrency PC ON TP.participantC[illegible]yId = PC.participantCurrencyId
   INNER JOIN settlementmodel M ON PC.ledgerAccountTypeId = M.ledgerAccoun[illegible]
   INNER JOIN settlementgranularity G ON M.settlementGranularityId = G.[illegible]mentGranularityId
   INNER JOIN transferParticipantRoleType R ON TP.transferParticipa[illegible]TypeId = R.transferParticipantRoleTypeId
WHERE
   TP.transferId = @ThisTransfer AND G.name = 'GROSS’ AND R[illegible] = 'PAYER_DFSP’
UNION SELECT
   PC1.participantCurrencyId, TP.amount
FROM
   transferparticipant TP INNER JOIN partici[illegible]currency PC ON TP.participantCurrencyId = PC.participantCurrencyId
   INNER JOIN settlementmodel M ON PC.led[illegible]ccountTypeId = PC.ledgerAccountTypeId
   INNER JOIN settlementgranularity G [illegible] M.settlementGranularityId = G.settlementGranularityId
   INNER JOIN participantCurrency [illegible] ON
     PC1.currencyId = PC.curre[illegible]yId
     AND PC1.participantId [illegible] PC.participantId
     AND PC1.ledgerAcc[illegible]ntTypeId = M.settlementAccountId
WHERE
   TP.transferId = @ThisTransfer AND G.name = 'GROSS’)
AS TR ON PP.participantCurrencyId = TR.ParticipantCurrencyId
SET PP.value = PP.value+TR.amount;
```

Results

- Before

participantCurrencyId	Participant	Account Type	Role Type	Value	reservedValue
3	payerfsp	POSITION	PAYER_DFSP	-581.7801	0
15	noresponsepayeefsp	POSITION	PAYEE_DFSP	600	0
4	payerfsp	SETTLEMENT	PAYER_DFSP	-8400	0
16	noresponsepayeefsp	SETTLEMENT	PAYEE_DFSP	-600	0

- After

participantCurrencyId	Participant	Account Type	Role Type	Value	reservedValue
3	payerfsp	POSITION	PAYER_DFSP	-481.78	0
15	noresponsepayeefsp	POSITION	PAYEE_DFSP	500	0
4	payerfsp	SETTLEMENT	PAYER_DFSP	-8300	0
16	noresponsepayeefsp	SETTLEMENT	PAYEE_DFSP	-700	0

Results

transferId	Participant	Ledger Type	Value
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	POSITION	-481.78
c3a05be5-37f9-4e78-83d1-e0b495231a2a	payerfsp	SETTLEMENT	-8300
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	POSITION	500
c3a05be5-37f9-4e78-83d1-e0b495231a2a	noresponsepayeefsp	SETTLEMENT	-700

Liquidity cover

The consequence of this approach for liquidity cover is that the switch will accept a request to transfer funds if:

- The proposed position, which is the total of:
  - Funds reserved but not yet cleared, and
  - Funds cleared but not yet settled, and
  - The amount of the proposed transfer request
- Is less than the liquidity cover, which is the total of:
  - Settled funds
  - Plus or minus any amounts credited or debited from settled funds by the scheme,
  - Less any reservations against settled funds made by the scheme or the participant.

Next steps

The way forward

- Custom aggregation
- Settlement on demand
- Full implementation of settlement model definition via portal
- Align settlement recording

Custom aggregation: the problem

- At present, we aggregate transfer entries for a net settlement in a standard way:
    - Participant DFSP
    - Currency
    - Account type
    - Role type (debitor, creditor)
    - Ledger entry type

Custom aggregation: the problem

- This is adequate for some purposes, but not for others
- For instance:
  - Mowali want to be able to categorise FX transfers by currency pair (e.g. Euro->Dinar)
  - A scheme which includes cross-network providers might want to aggregate cross-network transfers separately from intra-network transfers

Custom aggregation: the objective

- A scheme should be able to:
  - Decide how to aggregate its settlement values.
  - Assign individual transfers to categories based on their characteristics
  - Obtain settlement statements segmented by category

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
2. A generic mechanism for persisting the category with the transfer
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
   - The rules processor we have already developed could support this
2. A generic mechanism for persisting the category with the transfer
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
   - ➢The rules processor we have already developed could support this:
     - ➢We interrogate the content of the transfer request
     - ➢And add a SetCategory method to categorise the transfer
2. A generic mechanism for persisting the category with the transfer
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
   - The rules processor we have already developed could support this:
     - We interrogate the content of the transfer request
     - And add a SetCategory method to categorise the transfer
   - This would allow any per-transfer categorisation scheme to be implemented
2. A generic mechanism for persisting the category with the transfer
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
2. A generic mechanism for persisting the category with the transfer
   - We add a simple table to the central ledger database. It has two columns
     - A reference to the transfer to which the category belongs
     - The value of the category
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
2. A generic mechanism for persisting the category with the transfer
   - We add a simple table to the central ledger database. It has two columns
     - A foreign key reference to the transfer to which the category belongs
     - The value of the category
   - This would be a non-breaking change
3. An aggregation process for including categories in the aggregation

Custom aggregation: a possible solution

Components:

1. A way of assigning an arbitrary category value to a transfer
2. A generic mechanism for persisting the category with the transfer
3. An aggregation process for including categories in the aggregation
   - We join across from the **transfers** table to the **transferCategories** table.
   - We include the content of each category in the GROUP BY clause
   - And add a default value for NULL entries (which we allow)

Settlement on demand: the problem

- At present, all transfers are assigned to a settlement window at the time they are committed
- There is a single active settlement window at any time.
- When a new settlement window is created, there is no way of controlling which transfers are assigned to the new window and which transfers are assigned to the old one.
- Settlements can only be composed of whole settlement windows for a given account type.
  - They *can* be broken down by account types, which can be settled in different ways;
  - But it’s not possible to settle *some* of the transfers of a given account type in a given window

Settlement on demand: the problem

- Mowali need to be able to settle a group of transfers which share a characteristic (in this case, an exchange rate quotation) which is assigned *at the time of quotation*.

Settlement on demand: objectives

- I should be able to settle a group of transfers by defining their characteristics…
- While ensuring that I leave no transfer unsettled

Settlement on demand: a possible solution

1. Modify the settlement request API call to settle on demand
2. Continue to support old-school settlements

Settlement on demand: a possible solution

1. Modify the settlement request API call to settle on demand
    - Select by category and/or time period
    - Create a settlement window
    - Move all transfers that meet the criteria into the new settlement window
        - (whichever settlement window they are currently in…)
    - Create a settlement for the newly created window
    - …and away we go.
2. Continue to support old-school settlements

Settlement on demand: a possible solution

1. Modify the settlement request API call to settle on demand
2. Continue to support old-school settlements
   - ➢Any transfers which have been settled on demand will be removed from the standard settlement windows
   - ➢…so we can continue to support old-school net settlement without modification
   - ➢… and this will provide a "settle all not previously settled" option

Full implementation of settlement API

- Provide a structure to add APIs for the new settlement functionality
- Develop a reference portal to support the APIs

Align settlement recording

- Gross settlements now settle in the way we want:
  - Transfer amounts are moved to the settlement account
  - Positions are adjusted to record this movement
  - The liquidity check compares the position with the balance of settled funds
- We should extend this accounting to transfers which are settled net.

Any questions?