mojaloop

Supporting aggregators in Mojaloop

mojaloop

What criteria should we apply to decide whether a benefit belongs in Mojaloop or not?

- Thinking back to Monday's introduction, the questions we wanted to ask are:
  - Is this a feature which needs new kinds of access to the payment system, or can it make use of existing entry points?
  - Is this a feature where participants can make parallel offerings and customers can choose between them on the basis of cost, convenience or functionality?
  - Can we reduce the cost *for everybody* without stifling innovation *for anybody*?

What criteria should we apply to decide whether a benefit belongs in Mojaloop or not?

- Thinking back to Monday's introduction:
  - Is this a feature which needs new kinds of access to the payment system, or can it make use of existing entry points?
  - Is this a feature where participants can make parallel offerings and customers can choose between them on the basis of cost, convenience or functionality?
  - Can we reduce the cost *for anybody* without stifling innovation *for anybody*?

For aggregators, we'll need new kinds of access, and we talk about them here

What criteria should we apply to decide whether a benefit belongs in Mojaloop or not?

- Thinking back to Monday's introduction:
  - Is this a feature which needs new kinds of access to the payment system, or can it make use of existing entry points?
  - Is this a feature where participants can make parallel offerings and customers can choose between them on the basis of cost, convenience or functionality?
  - Can we reduce the co*for everybody* without stifling innovation *for anybody*?

Given the right kinds of access, most definitely

What criteria should we apply to decide whether a benefit belongs in Mojaloop or not?

- Thinking back to Monday's introduction:
  - Is this a feature which needs new kinds of access to the payment system, or can it make use of existing entry points?
  - Is this a feature where participants can make parallel offerings and customers can choose between them on the basis of cost, convenience or functionality?
  - Can we reduce the cost *for everybody* without stifling innovation *for anybody*?

Of course we can, and today we show you how

Our conclusion is:

Extending the Mojaloop system to support aggregators will:

- Provide new services with significant benefits to customers
- Support competition between operators to provide those services
- Increase throughput in Mojaloop systems

What is an aggregator?

How do we define an aggregator?

A participant in a Mojaloop scheme who is permitted to:

- Request payments from other participants (the payer DFSP)
- Obtain from its customers forms of authorisation which can be trusted by the participants holding the customers' accounts
- If required, split the payment received from the payer DFSP into multiple payment requests to other participants (the payee DFSPs)
- Issue payment requests directly to the payee DFSPs
- Support all-or-nothing transfers and any-of-many transfers
- Pay bills on behalf of customers and receive confirmation that customer accounts have been updated

How do we define an aggregator?

A participant in a Mojaloop scheme who is not required to:

- Be a direct participant in settlement
  - Settlement will take place between payer and payee DFSPs
- Rely on pre-existing relationships between customers and the DFSPs who own the customers' transaction accounts.

What sorts of entities might be aggregators?

- Entities who perform a traditional bill aggregation service
  - Their customers make a single payment to the aggregator
  - The aggregator distributes these payments to utilities at the customer's direction
  - And confirms that the customer's payment has been registered against their account with the utility
- Entities who accept payments which need to be disbursed across multiple beneficiaries
  - For instance, the SSNAPP ticket sales example
  - This is an *all-or-nothing* example
- Entities which act as marketplace managers
  - For instance, the SSNAPP marketplace example
  - This is an *any-of-many* example
- Entities which provide bulk services like running payrolls
  - The customer submits a single payroll and the service splits it into separate batches by payee DFSP, submits the individual payrolls and returns a consolidated report to the customer.
  - This has been canvassed as a potential switch upgrade consumed by DFSPs, but might make more sense as an adjacency

What would the aggregator use case look like?

1) The aggregator proposes the terms of the transfer to the switch

1. The customer asks the aggregator's application to purchase goods or services, or to make a bulk disbursement.
2. The aggregator sends a request to the switch which describes the proposed transfer, for instance:
   - This is an all-or-nothing payment
   - Pay 100 TZS to the aggregator
   - Pay 100 TZS to the government's tax account
   - Pay 800 TZS to the football team the customer wants a ticket for
3. The switch acknowledges the request in the normal way.
4. The request remains open…

This request is managed through an extended version of the proposed PISP functionality

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor (a VDFSP to manage this stuff)

{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType: "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
            (there's no FSPID, because the PISP doesn't know it yet)
        }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAPP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType: "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
         }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAPP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction
(= the switch…)

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType: "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
         }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAPP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction (= the switch…)

We specify that all parts of the transfer must succeed for the transfer to succeed

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType: "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
         }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAPP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction (= the switch…)

We specify that all parts of the transfer must succeed for the transfer to succeed

…and we specify that we don’t want to return per-payee information to the payer

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType": "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
         }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAPP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction (= the switch…)

We specify that all parts of the transfer must succeed for the transfer to succeed

…and we specify that we don’t want to return per-payee information to the payer

We don’t need to specify which DFSP owns the customer: the switch will sort that out. And we don’t specify the amount: the switch will calculate that from the payee amounts…

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType": "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
        }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAME"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction (= the switch…)

We specify that all parts of the transfer must succeed for the transfer to succeed

…and we specify that we don’t want to return per-payee information to the payer

We don’t need to specify which DFSP owns the customer: the switch will sort that out. And we don’t specify the amount: the switch will calculate that from the payee amounts…

We support any number of payees

mojaloop

An (abbreviated) example of a request

```
POST /transactionRequests

FSPIOP-Source: TheAggregator
FSPIOP-Destination: BulkProcessor
{
    "transactionRequestId": "7c23e80c-d078-4077-8263-2c047876fcf6",
    "commitmentRule": "AllOrNothing",
    “payerReport”:”Abbreviated”,
    "payer": {
        "partyIdInfo": {
            "partyIdType": "ALIAS",
            "partyIdentifier": "Mike@Somewhere"
        }
    },
    "payee": [
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "SNNAP"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
        {
            "partyIdInfo": {
                "partyIdType": "ALIAS",
                "partyIdentifier": "TheGovernment"
            },
            "amount": {
                "currency": "TZS",
                "amount": 100
            }
        },
```

A Virtual DFSP to manage the transaction (= the switch…)

We specify that all parts of the transfer must succeed for the transfer to succeed

…and we specify that we don’t want to return per-payee information to the payer

We don’t need to specify which DFSP owns the customer: the switch will sort that out. And we don’t specify the amount: the switch will calculate that from the payee amounts…

We support any number of payees

Again, the switch will identify the participants who own the payee accounts

2) The switch identifies the parties

1. The switch checks which participants own the payer identifier and each of the payee identifiers.
2. If any of the participants cannot be identified, the switch responds to the aggregator with an error and the transfer request is cancelled.

These functions are managed through existing switch functionality

3) The switch requests payment from the payer DFSP

1. The switch calculates the total payment which the payer will need to make (in the example case, 1000 TZS.)
2. The switch generates a transaction request for the payer DFSP to make a payment of the total amount *to the PISP*.
   - This should ensure that the transfer shows up correctly in the customer's statements: "I paid 1000 TZS to SSNAPP."
   - …and is a consequence of selecting abbreviated payer information in the request
3. The payer DFSP requests a quotation from the PISP.

These functions are managed through existing DFSP functionality

4) The switch requests quotations from the payee DFSPs

1. For each of the payees, the switch:
    1. Generates a quotation request for a transfer from the PISP to the recipient account.
    2. Stores the response, together with the condition that was generated by the payee.
2. If any quotation requests are rejected, the switch:
    1. Rejects the original quotation request from the payer DFSP
    2. Rejects the original transaction request from the PISP
3. If all the quotation results are accepted, the switch accepts the payer DFSP’s quotation request.

These functions are managed through existing DFSP functionality

4) The switch requests quotations from the payee DFSPs

1. For each of the payees, the switch:
    1. Generates a quotation request for a transfer from the PISP to the recipient account.
    2. Stores the respon
    by the payee.
2. If any quotation re
    1. Rejects the origin
    2. Rejects the original transact est from the PISP
3. If all the quotation results are accepted, the switch accepts the payer DFSP's quotation request

Open questions:

What condition should be attached to this quotation response?
- Should the switch generate it?
- Should the PISP generate it?
- Should it be a composition of the conditions returned by the payee DFSPs?

These functions are managed through existing DFSP functionality

5) The transfer request

1. The payer DFSP requests a transfer to the aggregator.
2. The switch reserves the DFSP's funds.
3. The switch constructs transfer requests to each of the payee DFSPs, based on the quotation responses they gave in step 4
4. The switch requests each payee not to clear the payment until it receives confirmation from the switch

This is new functionality in the switch: the DFSP functionality works as before, except that the payee DFSPs must take notice of the switch's instruction not to clear the funds until confirmation is given

5) The transfer request

1. The payer DFSP requests a transfer to the aggregator.
2. The switch reserves the DFSP's funds.
3. The switch constructs transfer requests to each of the payee DFSPs, based on the quotation responses they gave in step 4
4. The switch requests each payee not to clear the payment until it receives confirmation from the switch

Open question:

Who should the payer DFSP be in this request (and in the original requests for quotation)?

6) Transfer responses

- Each DFSP responds to the switch with success or failure
- After all DFSPs have responded, the switch:
  - Registers the transfers in its ledgers
    - The payer DFSP is debited with the total amount of the transfer
    - Each payee DFSP is credited with its portion of the transfer
  - Sends confirmation to the payee DFSPs that they can clear the funds to their account holders
  - Confirms to the payer DFSP that its transfer has succeeded.
  - Confirms to the PISP that its transfer request (which is still open) has succeeded.

This is new functionality in the switch: the DFSP functionality works as before

Confirming bill payments

What’s our definition of a bill payment?

- The customer wants to pay an organisation with whom they have (or the person they represent has) an account.
- They will specify which account at the organisation they want to pay.
- They’re not (very) interested in whether the payee organisation’s account has been credited with the funds or not.
- But they do want to know that their account with the organisation has been credited with the payment.

So the question is:

- How should we get confirmation from the payee institution that the customer's account with them has been credited?

Our proposal is:

- We allow entities who want to support bill payment to join Mojaloop schemes directly.

Our proposal is:

- We allow entities who want to support bill payment to join Mojaloop schemes directly.
- They only need to support a very lightweight interface with two resources:

Our proposal is:

- We allow entities who want to support bill payment to join Mojaloop schemes directly.
- They only need to support a very lightweight interface with two resources:
  1. Request to pay:
     1. Receive a request to pay to a customer account that they own (authenticated by a nomination of the account that will be credited)
     2. Respond to that request with confirmation that the customer account exists
  2. Payment confirmation:
     1. Receive a request to credit a nominated customer account with the amount of funds that have been transferred to a nominated utility account.
     2. Respond to that request with confirmation that the customer's account has been credited with the amount paid.

Our proposal is:

- We allow entities who want to support bill payment to join Mojaloop schemes directly.
- They only need to support a very lightweight interface with two resources:
  1. Request to pay:
     1. Receive a request to pay to a customer account that they own (authenticated by a nomination of the account that will be credited)
     2. Respond to that request with confirmation that the customer account exists
  2. Payment confirmation:
     1. Receive a request to credit a nominated customer account with the amount of funds that have been transferred to a nominated utility account.
     2. Respond to that request with confirmation that the customer's account has been credited with the amount paid.
- We allow the switch's aggregator service to generate these requests and confirmations as part of their management of the aggregation request.

Our proposal is:

- We allow entities who want to support bill payment to join Mojaloop schemes directly.
- They only need to support a very lightweight interface with two resources:
  1. Request to pay:
     1. Receive a request to pay to a customer account that they own (authenticated by a nomination of the account that will be credited)
     2. Respond to that request with confirmation that the customer account exists
  2. Payment confirmation:
     1. Receive a request to credit a nominated customer account with the amount of funds that have been transferred to a nominated utility account.
     2. Respond to that request with confirmation that the customer's account has been credited with the amount paid.
- We allow the switch's aggregator service to generate these requests and confirmations as part of their management of the aggregation request.
- And we allow aggregators to specify this functionality as part of their request to the aggregator service.

Er, that’s it…