mojaloop

The Mojaloop API Family

Can you see the resemblance?

Best practice in API design, by Ovid:

"facies non omnibus una,
non diversa tamen, qualem decet esse sororum."

Metamorphoses, II, 13-14

“Not one in all poyntes fully lyke an other coulde ye see,
Nor verie farre unlike, but such as sisters ought to bee.”

Tr. Arthur Golding

The Mojaloop API

- Originally developed and defined to support interoperabilityfor funds transfers between deposit-holding DFSPs
- Initially used in a peer-to-peer implementation between Mobile Money technology providers
- Subsequently used for the switch-based reference implementation of Mojaloop

API characteristics

1. “Service Oriented REST”-architecture (“RESTish”)
   - Services are distributed: there is no central authority which executes a transfer
   - Services imply actions as well as defining content
   - Services are not fully stateless: some parts of the orchestration of a transfer rely on comparing states
   - Clients decide common IDs

API characteristics

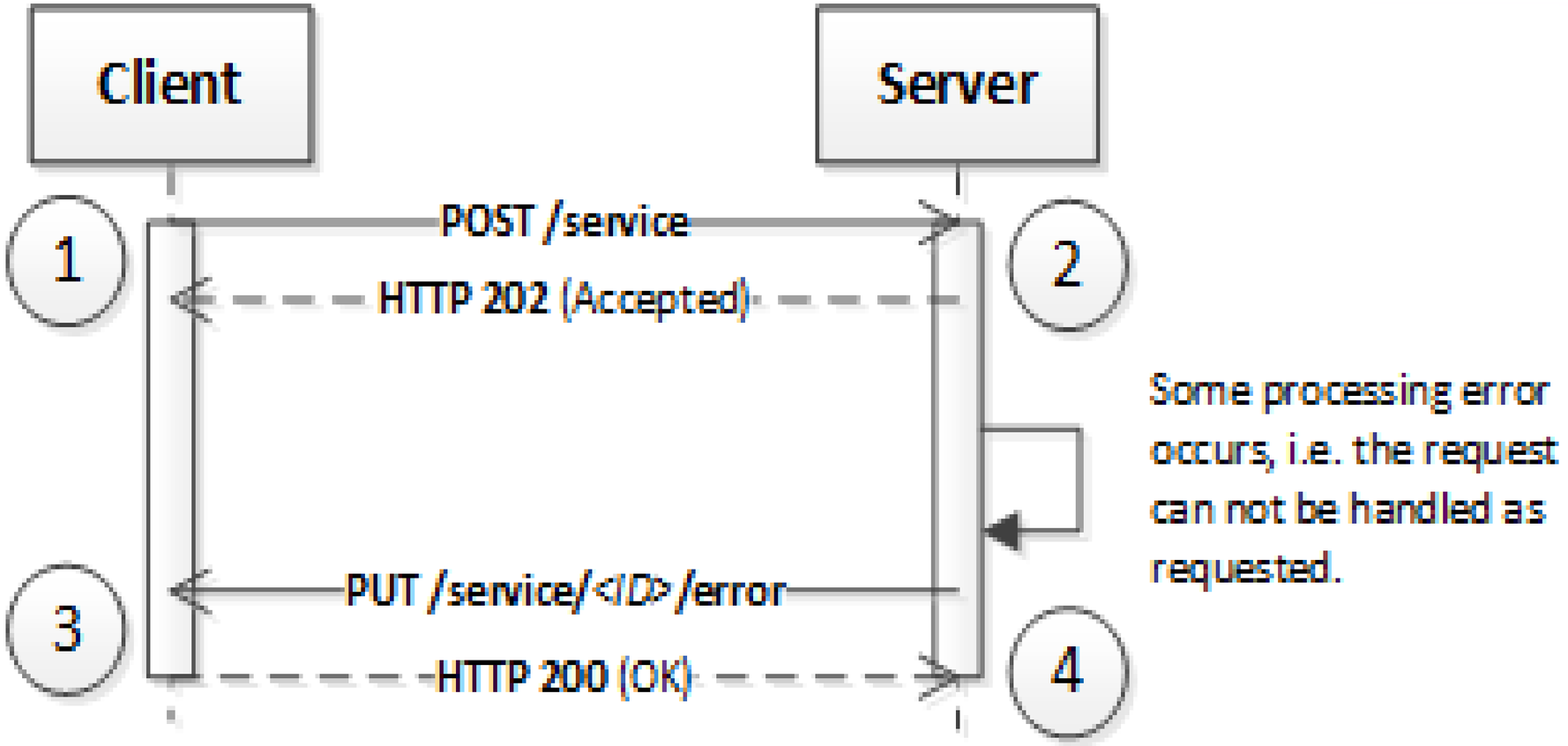
1. “Service Oriented REST”-architecture (“RESTish”)
2. Uses HTTP over TLS and standard HTTP calls
3. JSON is used as data exchange format
4. All services are asynchronous
   - A recipient will respond “immediately” to an HTTP call
   - But only with 2xx or 4xx status codes
   - All other errors are sent asynchronously in a PUT callback

General structure of an error response

If an error occurs as a consequence of its processing a message, a participant should:

- Respond to the resource as it normally would, but to the ***/error*** endpoint.
  - Normal response: `PUT /quotes/7c23e80c-d078-4077-8263-2c047876fcf6`
  - Error response: `PUT /quotes/7c23e80c-d078-4077-8263-2c047876fcf6/error`
- Send a standard message body, containing:
  - The error code
  - A description of the error
  - Any additional information, specific to the implementing scheme and the error type, in the form of key/value pairs

Schematic error response



Unexamined assumptions

- The level playing field is about enabling very different kinds of actors to behave in a standard way.
- So, every participant in a scheme will support the standard API…
- …and they'll all support all of it.
- So changes to the API can be managed by a single Change Control Board

Matthew Arnold considers functional extension…

"the world, which seems
To lie before us like a land of dreams,
So various, so beautiful, so new…"

Dover Beach

A world of new roles

- Payment Initiation Service Providers
- Cross Network Providers
- Aggregators
- Foreign Exchange Providers
- Account payees…

Managing roles

- We don’t want to constrain a participant to only one type of role.
- We want to allow participants to discover which other participants in a scheme can fulfil a particular role.
- Standard actions may be different in different role situations, and we need to have a good way of expressing that.
- We don’t want participants to be able to access a particular endpoint if their role doesn’t support access to it.

Our solution: multiple APIs

- There will be an API definition specific to a particular set of role interactions.
  - A role interaction may include two (or perhaps more) role types. Example: a PISP, and a DFSP which supports PISP linking…
- Participants will be registered for certain roles
  - And a participant may have more than one role type
- The API will specify only the messages which constitute a role interaction.

But what about -

- Situations where more than one API uses (or wants to use) the same message, or the same data object?
- Situations where it’s not clear whether a new API is required or not?
- Keeping the various APIs aligned with each other?
- Defining and ensuring best practice?

But what about?

- Situations where more than one API uses (or wants to use) the same message, or the same data object?
  - We'll store the object centrally, and APIs can simply include it in their definitions
- Situations where it's not clear whether a new API is required or not?
- Keeping the various APIs aligned with each other?
- Defining and ensuring best practice?

But what about?

- Situations where more than one API uses (or wants to use) the same message, or the same data object?
- Situations where it’s not clear whether a new API is required or not?
  - We’ll have a standard way of deciding on that
- Keeping the various APIs aligned with each other?
- Defining and ensuring best practice?

The Change Control Board is changing

- Individual APIs will be managed by Special Interest Groups (SIGs)
- Each SIG will have a representative on a new Change Control Board.

The new Change Control Board

- Has a representative from each SIG responsible for a particular API.
- Defines best practice in API definition and documentation, and ensures that SIG definitions comply with it.
- Is the guardian of common data objects, and is responsible for changing them as required.
- Is responsible for reviewing proposed APIs and for publishing them when approved.

The new Change Control Board

- Has a representative from each SIG responsible for a particular API.
- Defines best practice in API definition and documentation, and ensures that SIG definitions comply with it.
- Is the guardian of common data objects, and is responsible for changing them as required.
- Is responsible for reviewing proposed APIs and for publishing them when approved.
- *…and has an overall responsibility for promoting world harmony and peace.*

What do we want the new world to look like?

It’s clear to a new observer:

- What sorts of roles Mojaloop supports
- How those roles relate to each other
- How individual APIs support participants in those roles
- What the standard operational patterns for Mojaloop APIs are

It’s clear to teams who are extending Mojaloop functionality:

- Whether their work requires a new API or not
- How to define the roles associated with their work
- How to go about defining an API to support those roles
- How their work relates to the overall API landscape of Mojaloop

Any questions?