## Endeavour architecture proposal

Jonny Rylands | 16 December 2015

## Introduction

We are building a data service at national scale, with:

* Many inputs in different clinical and messaging formats.
* And many outputs in different clinical and messaging formats.

Ambition to connect every care management system in the country.

=> hospitals themselves have > 100 systems each.

On top of this we have new projects underway:

* Leeds – Vitrucare, Pharmacy, Analytics
* GP-CIM
* Search and reports
* Family history

And many more in the pipeline:

* Spine connectivity/SMSP
* GPES clone
* CDA/discharges/transfer of care
* Medium/long term ambitions
* Complex, and growing at an accelerating rate.
* We need to ways of taming the complexity!
* And we should do it before it gets hard and expensive to do. (I.e. before we have live customers)
* Separation of concerns (aka divide and conquer, single responsibility principle) & loose coupling

## Separation of concerns & loose coupling – specifics

Don’t want huge monolithic application => harder to understand, develop, test, release, manage as time and requirements progress.

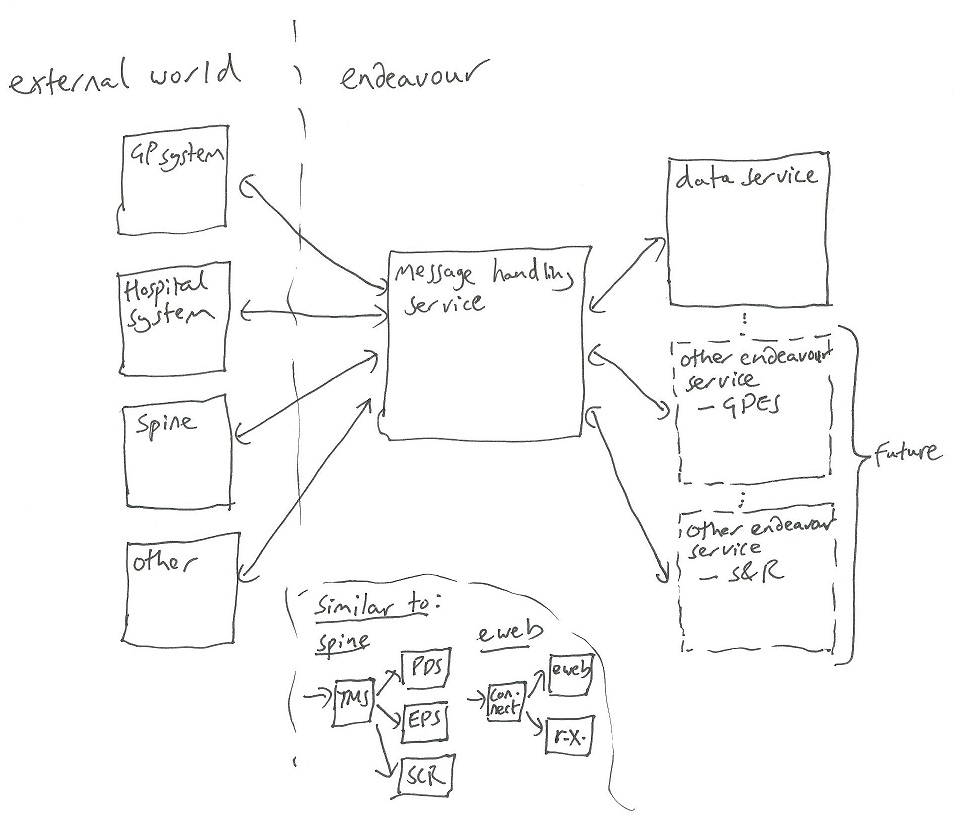
Separate application into more manageable, modular, loosely coupled pieces:

1. Separate messaging from data service.

Keep:

* Transports (http, tcp)
* External messaging formats (SOAP, REST, pipe and hat)
* Identification and authentication (Certificates, API keys)
* Authorisation (message X can be sent between Org A and Org B)
* Retries
* Async queues and acks
* Message logging

Separate from clinical/business domains to allow data service to focus on its job and not be involved in the intricacies of message delivery.

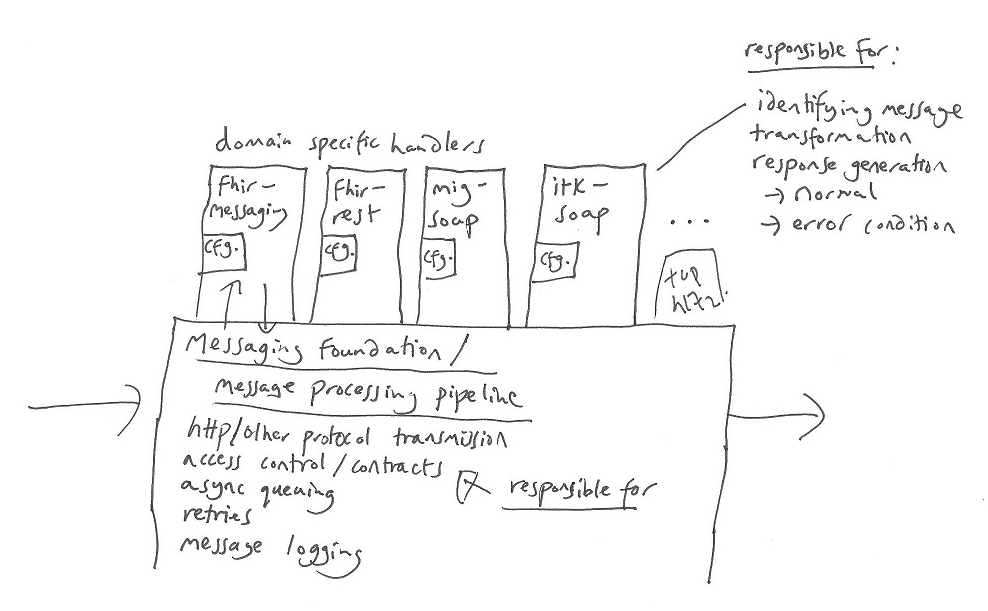


Note: V high level – see examples for greater detail

* Keeps boring transport code from clinical/domain specific code.
* Do common transport things like retries, async queues, transmission protocols once for all endeavour projects.
* Provides additional layer of abstraction/indirection to:
  + Keep single canonical model with data service and manage transformations outside it
  + Allow changes to data models and interfaces to be made easier – and keep core data service clean and able to change.
* Allows messaging only projects (e.g. GP-CIM, CDA) to be kept fully away from data service.
* Data service can be taken offline and messaging service can continue to accept async messages.

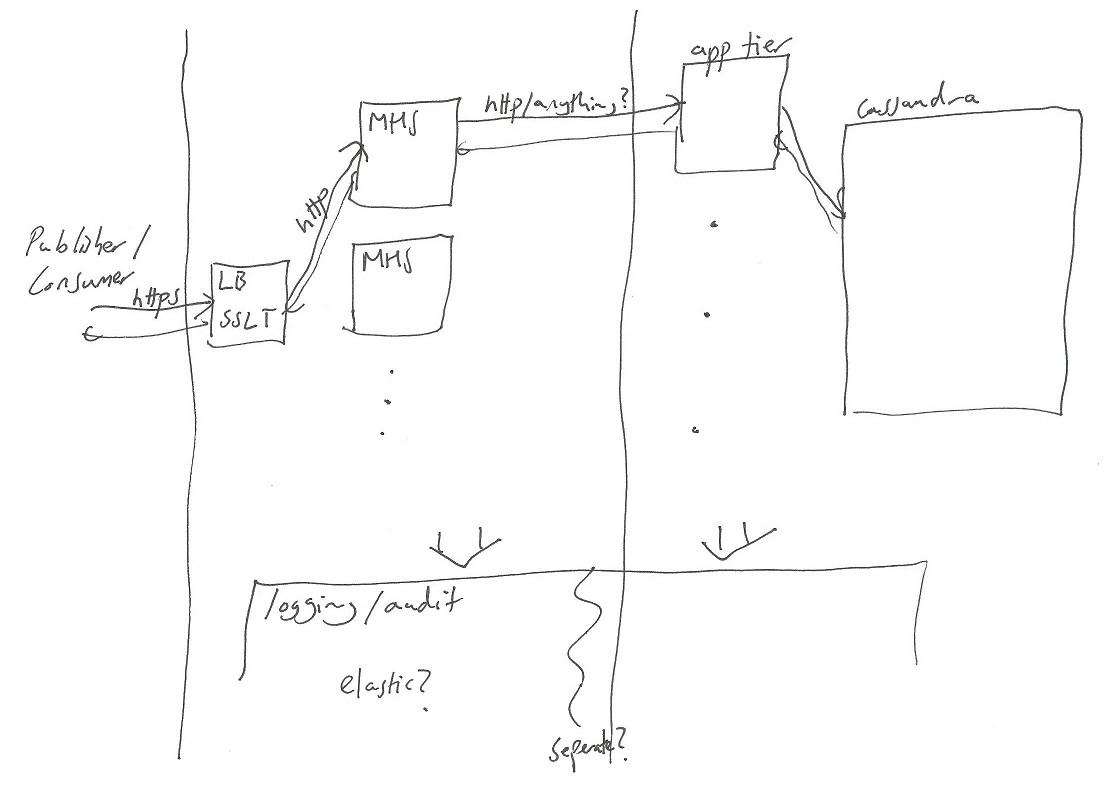
2. Separate messaging core from format specific message processing.

Single messaging pipeline with pluggable format handlers and associated configuration:



## 1. Separate messaging from data service – Examples

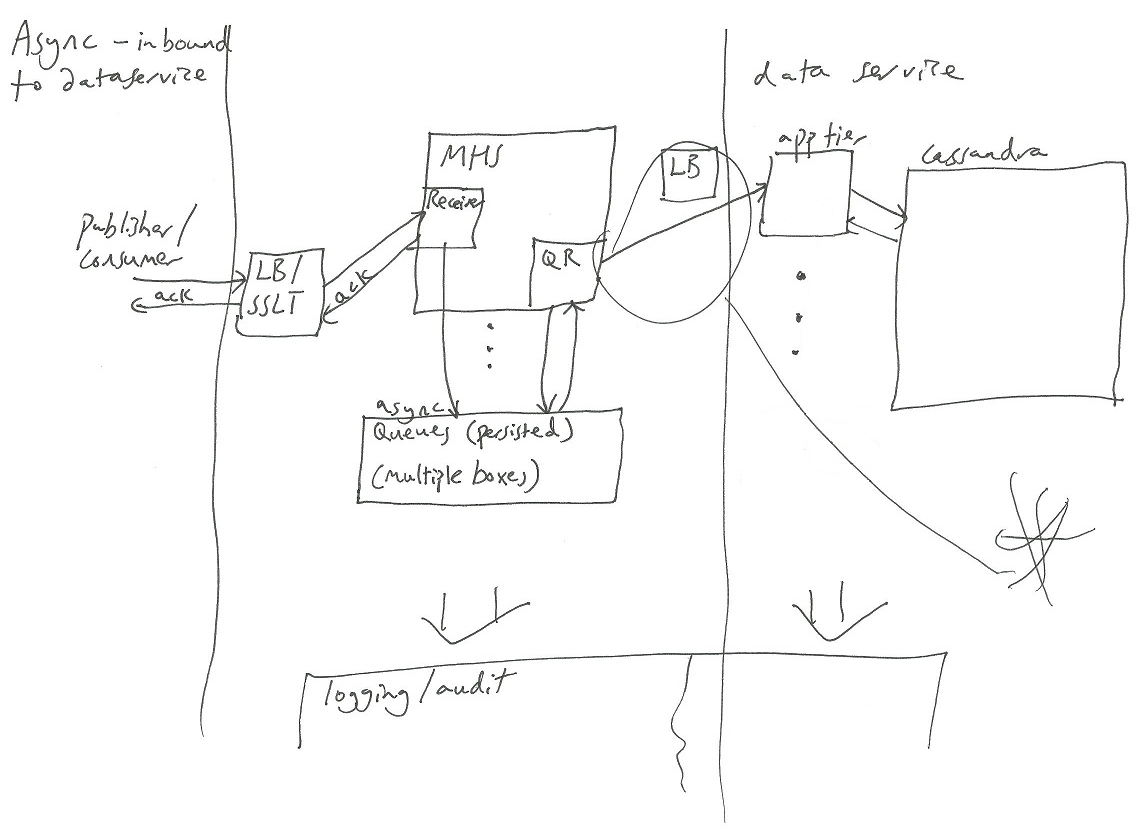
## 1.1 Synchronous call to data service



## 1.2 Synchronous GP-CIM call

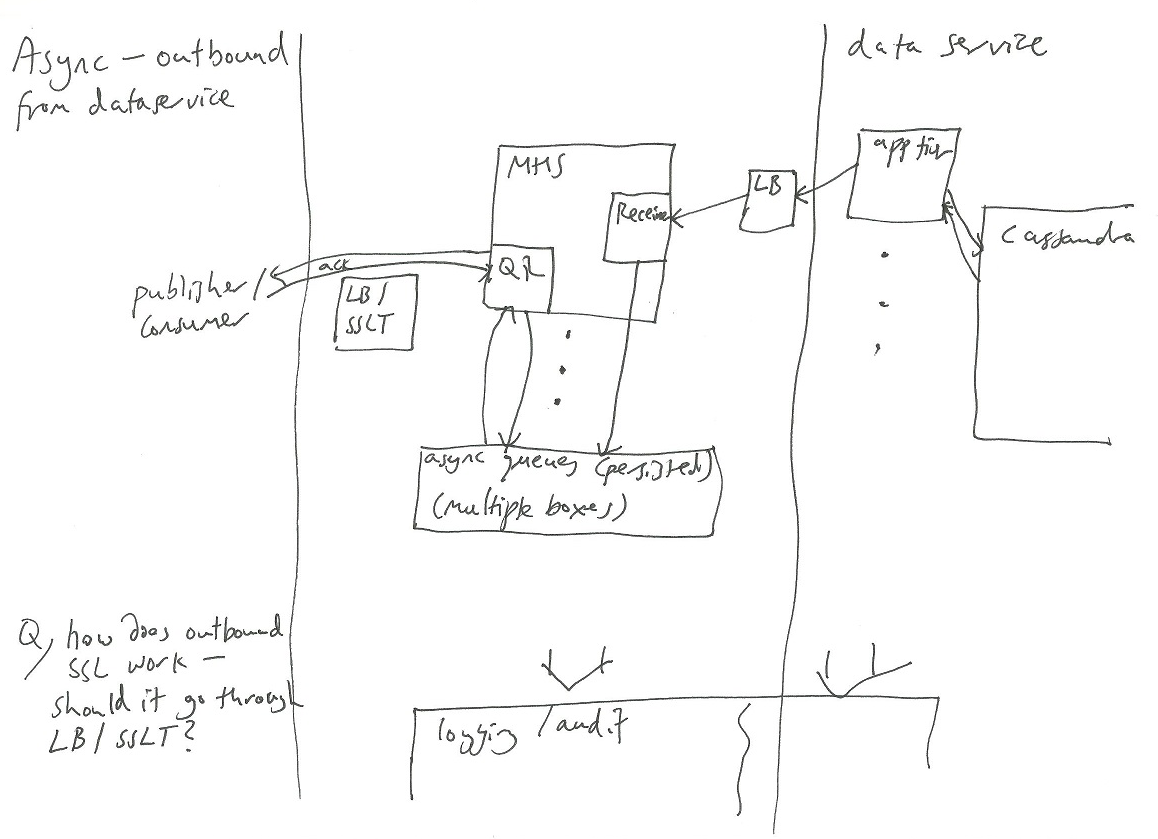
## 

## 1.3 Asynchronous call to data service – inbound leg

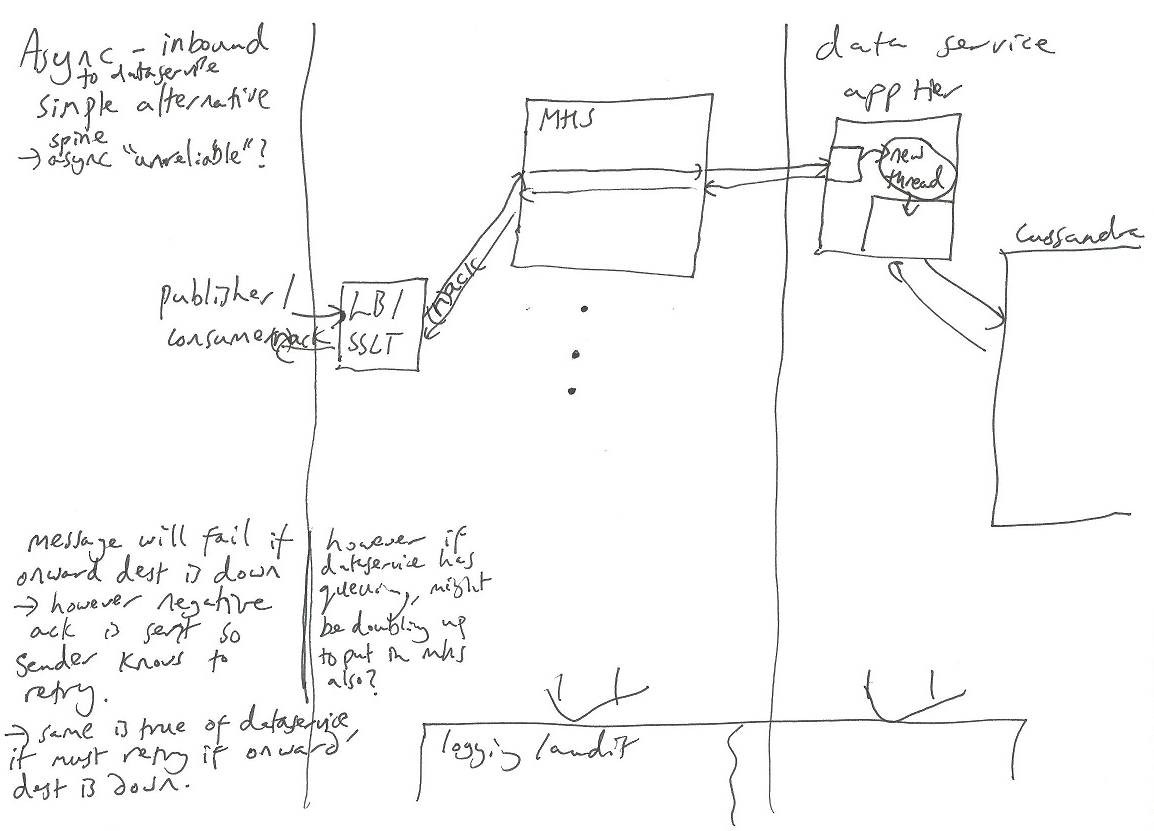


\* should the data service have queues of its own, or should the queue reader in the MHS synchronously wait for the data service to process its message (and queue up outbound messages)?

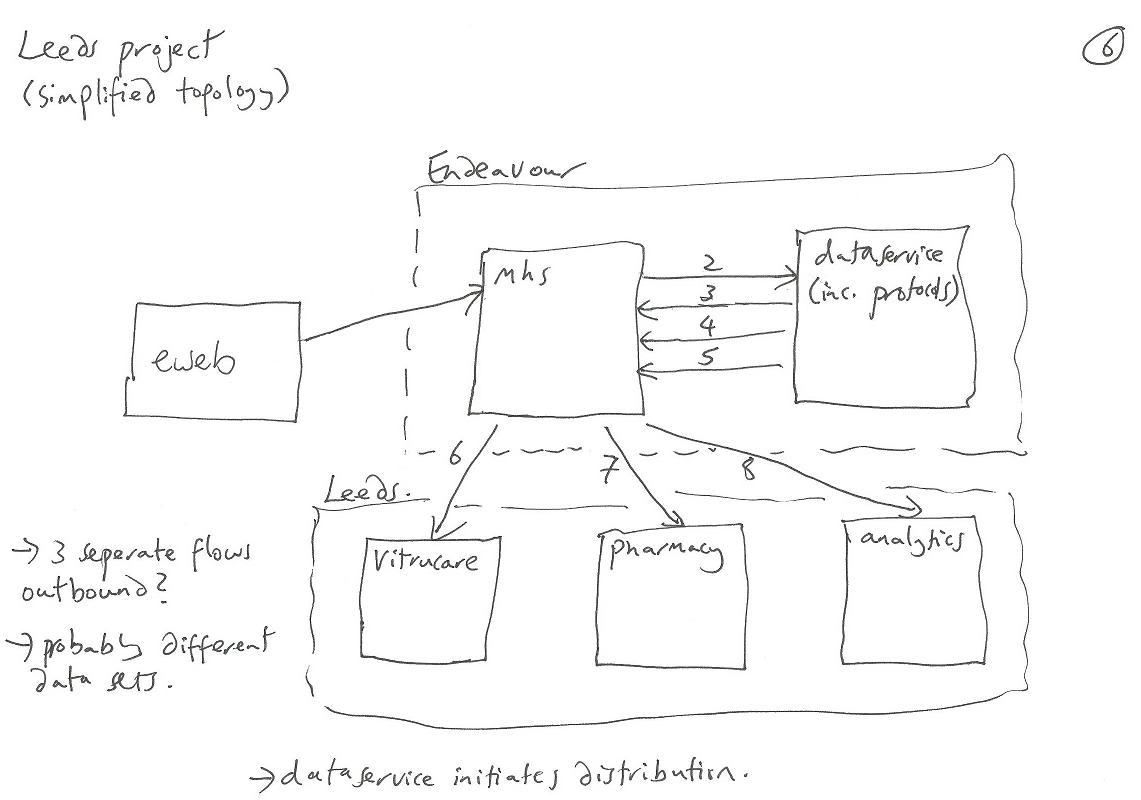
## 1.4 Asynchronous call to data service – outbound leg



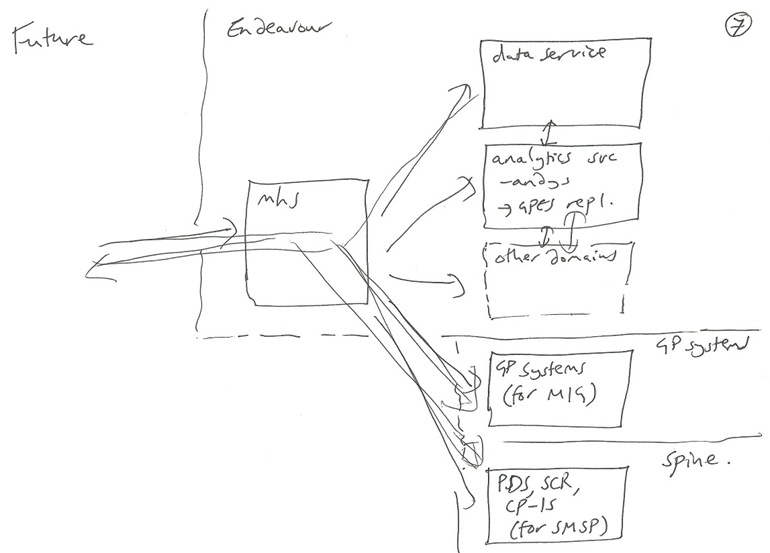
## 1.5 Simpler “asynchronous” alternative call to data service – inbound leg



## 1.6 Leeds project (simplified topology)



## (Near) Future – with GP systems, Spine and additional Endeavour domains



## 2. Separate messaging core from format specific message processing

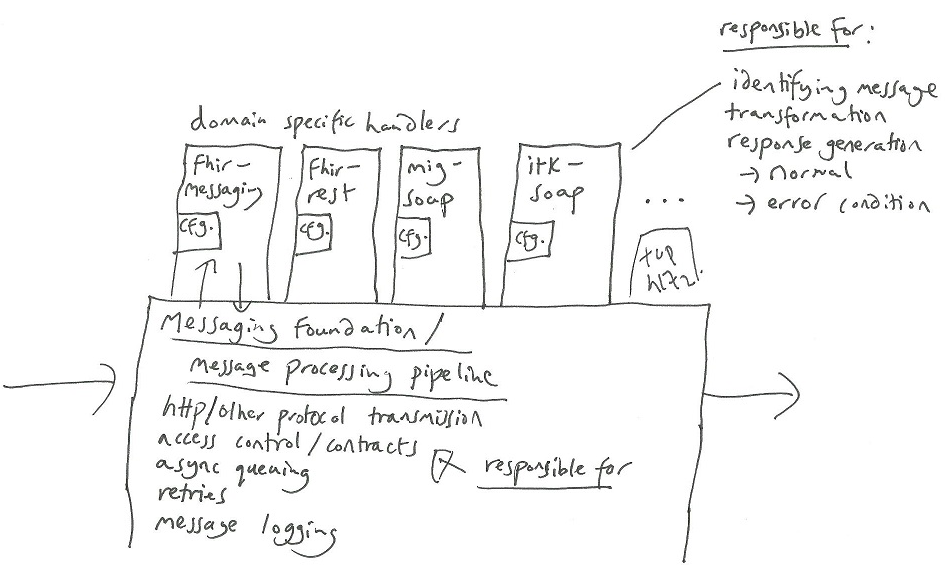
Create simple core service which handles common messaging themes:

* Transports (http, tcp)
* Identification and authentication (Certificates, API keys)
* Authorisation (message X can be sent between Org A and Org B)
* Retries
* Async queues and acks
* Message logging

With a single messaging pipeline for each pattern:

* Sync
* Sync proxy (e.g. GP-CIM)
* Async
* Async unreliable?

Messaging formats, message content, transformations and associated configuration are handled by pluggable modules, e.g.

* fhir-rest
* fhir-messaging
* mig-soap
* itk-soap

Each pluggable module comprises of:

1. **An Endpoint catalog**

E.g. (for the example have grouped endpoints of multiple format into one, but better to split across modules)

|  |  |
| --- | --- |
| **Endpoint** | **Endpoint handler class** |
| POST /mig-soap | MigSoapEndpoint |
| POST /fhir/$process-message | FhirMessagingEndpoint |
| POST /itk-soap | ItkSoapEndpoint |
| GET, POST, PUT, DELETE /fhir/\* | FhirRestEndpoint |

(404, 405 is returned if endpoint cannot be matched in any endpoint catalog)

1. **A set of endpoint handler classes**

It is the responsibility of the endpoint handler to identify the message once received off the wire, the handler should return:

* Message name
* Version (if separate from name)
* Sender
* Recipient

It is also the endpoint’s responsibility to construct

1. **A message catalog**

E.g.

|  |  |
| --- | --- |
| **Message name (version)** | **Message processor class** |
| MigEhrExtractRequestv1 | MigEhrExtractRequestv1Processor |
| MigTraceRequestv1 | MigTraceRequestv1Processor |
| FhirMessagingGetFullRecord | FhirMessagingGetFullRecordProcessor |
| FhirRestAny | FhirRestAnyProcessor |

1. **A set of message processors**

The message processors job is to take the message and transform as desired before reaching destination.

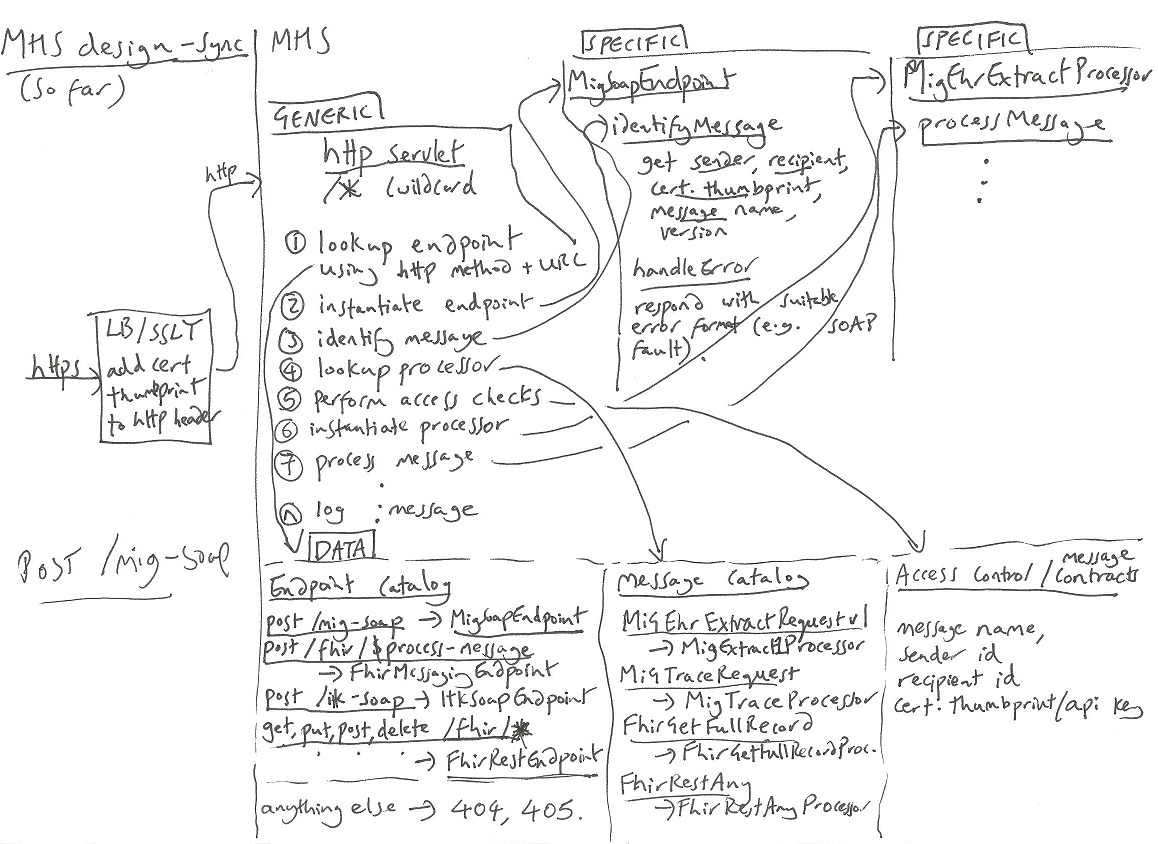
1. **A message contract catalogue**

This allows the core engine to check that messages can only flow between authorised parties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Message name (version)** | **Sender id** | **Recipient id** | **Sender cert thumbprint / api key** |
| MigEhrExtractRequestv1 | ods:A93847 | ods:B93847 | c3:a3:43:32:45:a9:bc |
| UploadEmisOpenEhrToDsAsync | ods:C29948 | endeavour:ds | a9:b2:10:93:38:32:a2 |
|  |  |  |  |

Section to finish – work out async message pipelines.

Diagram needs some re-work!



## Practical considerations

## Data service

The data service doesn’t change significantly – it is still responsible for persisting and initiating the distribution of records using data distribution protocols, but doesn’t have to cope with a multitude of formats and can focus on these jobs rather than transmission, retries, access control and other delivery related aspects.

## Source control

Separate codebases and repositories:

* DataService
* MessageHandlingService

If we have sufficient common and standalone code (libraries, helpers), create a third repository:

* Common

This should be built independently and referenced as a binary package import (in a similar fashion to Maven, Nuget packages).