Architecture discussion: ESB vs No ESB

Introduction

On the interoperability layer call yesterday we discussed some of the issues that have been brought up by the interoperability layer community members. Specifically, our conversation revolved around the use of a ESB for the interoperability layer. We identified some key problems that present themselves in the current ESB tools that we would NOT like to see within the interoperability layer:

1. We don't want the interoperability layer to be one large monolithic application. It should rather be modular such that it can be easily extended and it should not be a single point of failure.
2. For the heavy processing of messages within the interoperability layer we do not want to be using graphical programming tools to do so as some ESB tools require. We see the use of pure code to be more reusable, maintainable and easier to understand in the long run.

We also talked through the features that an ESB provides (message transformation, routing and orchestration) and found that these largely is what we want to achieve with the interoperability layer, however, we would like to be wary of the issues listed above.

In light of these point we discussed how a interoperability layer could be architected. We came up with two main independent components that could make up an interoperability layer:

1. Security and routing component: this component receives requests and can handle authorization and authentication, it can log and store the message it receives and it can route a message to the correct service provider or to component 2 if required.
2. This is a set of components that can handle more complex transactions that require orchestration. We can call these components processors or mediators. Each one of these processors/mediators will be able to transforms message and orchestrate specific message for a particular purpose (eg. for validation a messages content against the PR, FR and CR before sending to the the SHR). Only messages that require orchestration get routed to the correct processor/mediator from component 1.

I have included a very rough diagram of these components.

Two Diagrams

What follows are two diagrams of the central node. After some discussions, we realize that they are describing the same problem, but at two levels of generality. Ryan's shows the situation "in general". Logging and authentication for the "Main Line HL7 Data Flows" is handled by exactly the same machinery as the "pass through registry exposure" calls. Mark's diagram shows a concrete architecture conforms to Ryan's general picture. By saying that "pass through registry exposure" goes through a pass-thru apache, we are in essense limiting the amount of authentication and transformations that happen between what is exposed (message m1 sent to FR.exposed), vs what is presented to the FR (message m1')

Mark's diagram

The diagram below shows the internals of the Core Node.

At the top is a vanilla Apache. It exposes the restful interfaces of the various registries, and performs authentication, logging, on their behalf. For example, clients connect to the FacilityRegistry by presenting proper URL to the PassThruApache, which then forwards it off the the actual Facility Registry.

At the bottom is the full-on HL7 pipeline V2 messages arrive from edge nodes via LLP. Are remembered forever in the Raw message store. A pre-processor attacks them to normalize them. (Note: In the common case, the arriving messages are already "perfect"! But, when the edge node had operated disc-connected, there may be issues. In any case, we are ready to do hard work for mal-conformant edgenodes.) (NOTE: I assume that the LLP reader implements some simple certificate-based security.)

Finally, the normalized messages remembered, and presented to the SHR.

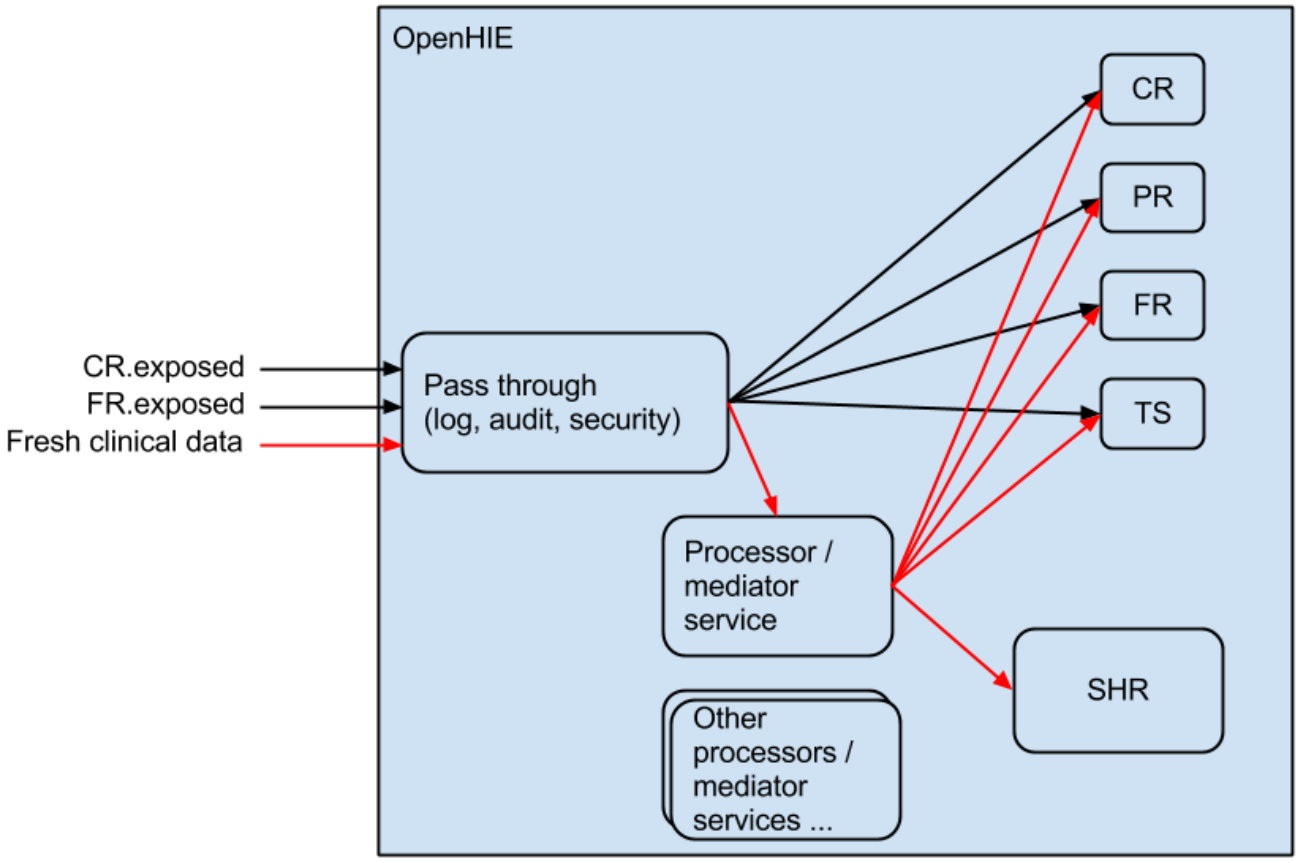
The bottom layer need not use an ESB, and may involve multiple JVM's.

In the middle, is a place holder for a ESB. The top and bottom layers handle very common use cases. If the need arises, an ESB could slot in at this point.

Unknown macro: 'mockup'

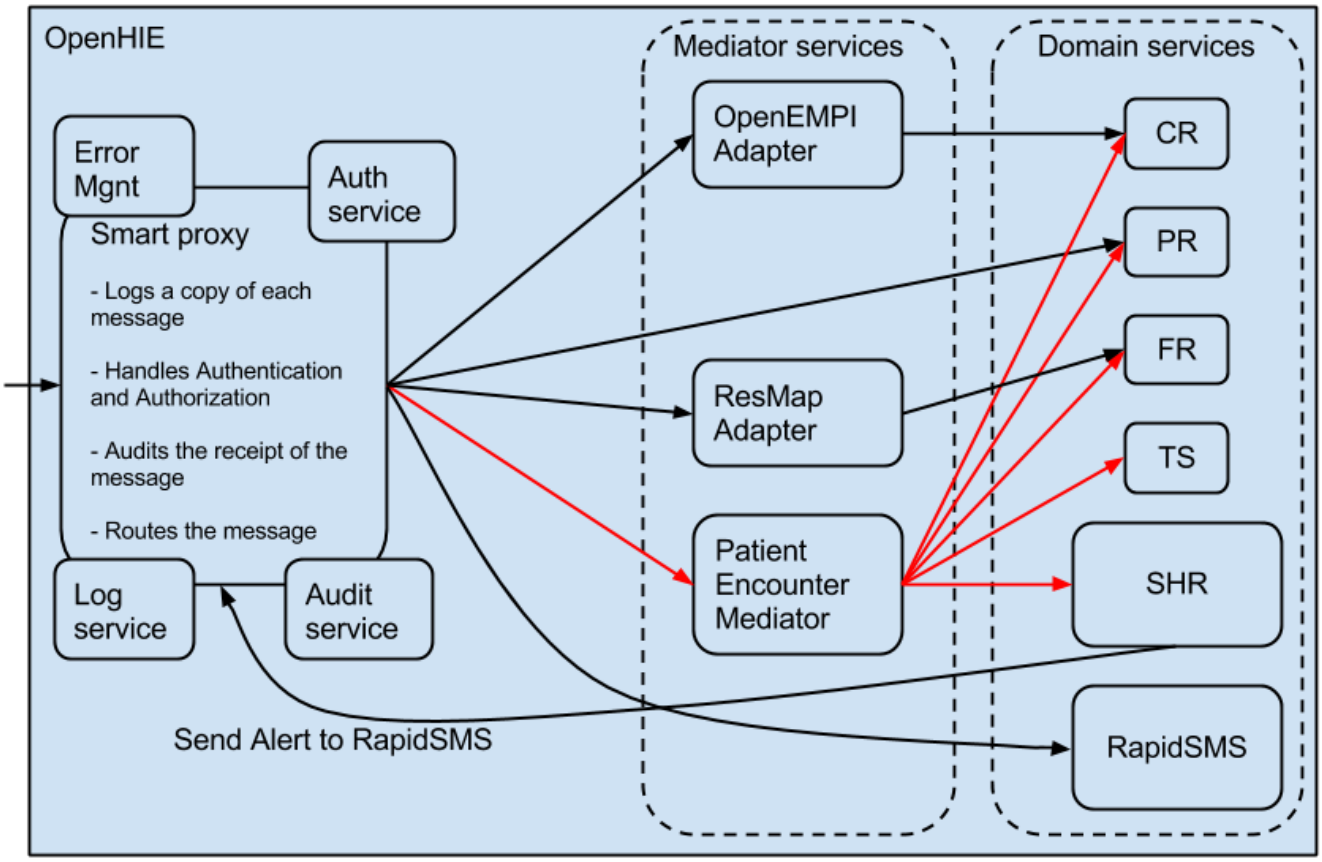
Ryan's diagram

This is an adaptation of Mark's diagram where I (Ryan) give my views on how the components of the interoperability layer should be structured. This links in with the descriptions given in the introduction. The key difference is that every message goes through the pass through component and the messages that need further orchestration or processing are forwarded on to a processor/mediation component to perform this orchestration. This allows security, persistence of messages, logging and auditing to occur in a single location.



Considering the Rwandan HIE architecture with the new Interoperability Layer thinking

Here is a diagram depicting how the current Rwandan HIE architecture could be structured to conform to the generic interoperability layer architecture given above.


OpenHIE
Mediator services
Domain services
Error Mgnt
Auth service
Smart proxy
- Logs a copy of each message
- Handles Authentication and Authorization
- Audits the receipt of the message
- Routes the message
Log service
Audit service
OpenEMPI Adapter
ResMap Adapter
Patient Encounter Mediator
CR
PR
FR
TS
SHR
RapidSMS
Send Alert to RapidSMS