OpenTele Modularization

A proposal for discussion

Authors:Torben Bisgaard Haagh, Alexandra Institute, Michael Christensen, Aarhus University

1. Introduction

Objective: The intention of this paper is to serve as input for 4S community discussions on a restructuring and further modularization of the OpenTele platform. Thus, it is not a recommendation in itself but rather an initial presentation of ideas that serve as a proposal for discussion.

Assumptions: It is assumed that the reader of this paper is a person with a technical background (software developer or software architect) who is familiar with the OpenTele platform.

Background: OpenTele is an open source project, in a setting where multiple vendors and multiple customers need to collaborate. This along with extra challenges in relation to handling medical, product, and data protection liability directives, are incentives to revisit the OpenTele architecture. As outlined in the proposal for a 4S Roadmap [9] some of the drivers for a further modularisation and componetisation of OpenTele are:

- Increased opportunities for reuse across 4S software and across individual supplier solutions
- Efficient development and collaboration in smaller teams
- Increased and modularised support for the Danish reference architectures

With this in mind we propose to have a closer look at such software architectural qualities as *development distributability*, *deployability* and *variability*, cf. [1], complementing the earlier *OpenTele analyserapport* [2] that specifically focused on the qualities *perfomance*, *compatability*, *reliability*, *security* and *maintanability*.

Proposals: The above mentioned focus leads us to revisit the structure of the OpenTele platform with a particular attention to how and where the principles of separation of concerns and loose coupling can be further strengthened and applied. We propose to investigate restructuring in the following (not mutually exclusive) categories:

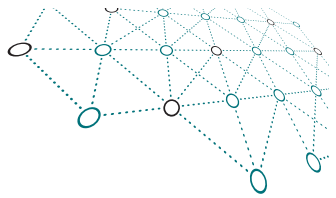
1. Compile time modularisation
2. Runtime separation of concerns
3. Microservice orientation [5]

A typical example of the first category of modularisation is the current work in relation to 4SDC: a library for easy integration to measurement devices from mobile device code. This is done by providing a cross-platform library written in C++ that can communicate through a platform abstraction layer (PAL) to the platform specific interfaces. For more, see [4].

Modules for server as well as client side support for IHE PCD'01 [10] are also good new candidates for compile time modularisation.

An example of the second category would be extracting the questionnaire editor from OpenTele, into an application of its own. Ideas about this are briefly outlined in the section below on 'Candidate Extractions'.

In the following we very briefly outline the current OpenTele architecture and then delve into a bit more detail about the new ideas for applying microservice style architecture on parts of the



OpenTele platform. Finally we conclude with some very preliminary ideas and examples of what could candidates for further modularisation and separation.

OpenTele as-is

The OpenTele platform as whole is currently structured as a fairly classical client-server based application. Citizens use a tablet based client to communicate with a server side, about such data as questionnaires, messages and measurements. Clinicians on the other hand can connect to the server side via a browser based web UI. Via this UI they are, for instance, able to view measurements taken, administrate questionnaires and communicate with citizens.

On the server side a shared SQL-based database is responsible for storage and servers are structured around an MVC style architecture, inherited through the use of the Grails Framework [3].

2. Microservices

Traditionally architectures have been layered, most typical in a 3-tier architecture with a data, business logic, and presentation tier. This gave a nice separation of concerns, but also tied the vertical functionalities together in horizontal layers. Amazon, Netflix, Uber and other companies have changed the way they develop and deploy software – they use an architectural style called microservices [5]:

> *The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.*

They also heavily use and release open source packages, e.g. Netflix: http://netflix.github.io.

There is a lot of religion in different software development strategies. Some microservices purists will, for instance, make an argument for “one action per single-function microservice”. We believe that it is possible with a more pragmatic approach, where we aim to keep the services small, but will allow services that arguably are not microservices.

Conway’s Law states [7]:

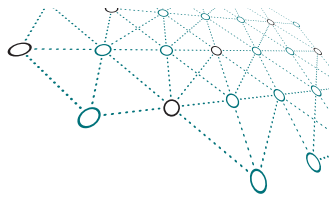
> *Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.*

What we would like to achieve here is an *Inverse Conway’s Law*: the architectural structure facilitates a possible organizational structure, i.e. a loosely coupled architecture facilitates a loosely coupled organization.

2.1. Characterics of a Microservice Strategy

Characteristics of a microservice system as summarized in [6]:

- *Componentization via services: Components in a microservice system are services at runtime. A service is a self-contained piece of functionality that reacts to one or more actions as defined in its service contract.*
- *Out-of-process communication: Each service is confined to its own process and communicates with other services only through RESTish protocols or messaging systems.*
- *Organize teams around business capabilities: Each service is designed by only one team and handles a well defined part of the business: a business capability.*



- *Continuous deployment: Services are continuously being deployed and each service can be deployed independently of others.*
- *Monitoring of services and business metrics: High quality monitoring of services, queue depths, event throughput, et cetera, is paramount. Business metrics are used to drive service development.*
- *Multiple programming languages and tools: Services may be implemented in different programming languages and use different data storage technologies.*
- *Decentralized data management: Each data store can only be owned by one service. Others services must not access it directly and may only get access to its data through the owner's service API.*

Some further points:

- *Deployment:* Each microservice has its own code repository, build, build life cycle, and can be deployed independently of other services. This means that each separate service can be handled in isolation.
- *Customization:* Some functionalities can be implemented differently for different uses. E.g., one customer might want to use an open source video solution based on WebRTC, and another customer might want to use a closed source video solution from Vidyo.
- *Reuse:* Modules implemented for OpenTele could potentially be used in another context.
- *Isolation of Critical Elements for Certification:* Elements, which require certification, have to be handled in a special way, undergo risk assessment, etc. This requires an extra effort, hence it is beneficial if functionality that needs certification is encapsulated in separate modules. Separating this away, will allow for easier handling of changes in other functionalities. See section 3.2 for an example.
- *Isolation of Faults:* Isolation of functionality also means isolation of faults, e.g. a memory leak in one service only affects that service. A service architecture also means that you are more inclined to design for errors tollerance, because simple networks issues can cause errors.
- *Technology Stack:* A separation into multiple services will also allow for a free choice of platform.

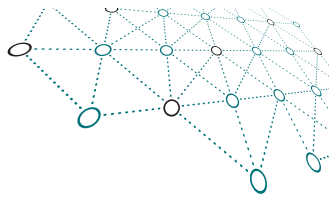
3. Candidate Extractions

The proposal is to incrementally extract modules from OpenTele. The first candidate for extraction is the questionnaire editor.

3.1. Questionnaire Editor

A profilation of the questionnaire CDA templates from HL7 (QFDD and QRD) is under way. It would be advantageous for the community if there were an open source platform for creating questionnaires. The questionnaire editor in OpenTele is a single page web application, and thus a good candidate for extraction. Extracting the questionnaire editor from OpenTele and implement support for the HL7 CDA templates, will give an open source tools for creating standard based questionnaires for the community.

Notice that the editor initially only will support the constructs supported in OpenTele.



3.1.1. Questionnaire Storage

The storage of the questionnaires could also be extracted as a separate service. Then it would be possible to centralize the storage, where multiple OpenTele installations could utilize the same questionnaire repository. It will also allow for questionnaire from another source.

3.2. Measurement Assessment Service

Today OpenTele offers to assess the measurement with four different colors, which hints the clinician about the actions he needs to take. This is critical in relation to certification according to the medical directive, as a fault in the assessment can be critical for the patient's health. Thus, it would be beneficial to extract this functionality as a separate service, because changes in this functionality requires special handling. Separating this away, would allow for easier handling of changes in other functionalities.

This is a good microservice example.

3.3. Message System

OpenTele has a proprietary end user message system, when the clinician and the patients communicate within the OpenTele system. Anthropological studies have shown that this is problematic, as some patients view the OpenTele client as a medical device, which they preferably do not want to relate to, when they are not making measurements. Thus, a message from the clinician can be unnoticed. It would be beneficial, if the human messaging system could be integrated with the patient own messaging system, e.g. email or phone text messages. This type of functionality has been referred to as *integrating your own device*, *IYOD*.

This task would result in a number of isolated services and modules.

4. References

[1] Len Bass, Paul Clements, Rick Kazman. *Software Architecture in Practice*, 3rd ed., Addison Wesley, 2013.
[2] LakeSide, OpenTele analyserapport, 2014-04.
[3] Grails project. *Grails Framework*, https://grails.org/, retrieved 2015-04-07.
[4] 4SDC wiki: http://4s-online.dk/wiki/doku.php?id=4sdc:overview, retrieved 2015-04-10.
[5] Martin Fowler, Microservices, 2014-03-25, blog
[6] Morten Larsson, Microservices, Master Thesis, september 30, 2014.
[7] Mel Conway, Conway's law, 1967, definition
[8] Alistair Cockburn, Hexagonal Architecture, 2005-04-01, blog
[9] 4S, Notat om roadmaps for softwaren i 4S, 2014-06-11, notat
[10] http://wiki.ihe.net/index.php?title=PCD_Technical_Framework