Architecture-Driven Development

University of
St Andrews

170002815 · MATRIC NO · MATRIC NO · MATRIC NO

CS5033 · Practical 2

22 March 2022

1 Overview

This report explores an example architecture for a healthcare system which handles the management of patient information and appointments. The process of creating the architecture is explored in Sections 2 and 3, with the resulting architecture being described in Section 4; Sections 5 and 6 provide an implementation plan for and a critical evaluation of the chosen architecture respectively. The report is completed by a conclusion (§7) and set of references.

2 Architecting Process

2.1 Resources

The primary literature resources used to influence the design, evaluation, and implementation plan of the chosen architecture were Robert C. Martin's Clean Architecture [1], and Mark Richards and Neal Ford's Fundamentals of Software Architecture [2].

Clean Architecture provides information on architecture from a practical standpoint, including fundamental principles to follow when architecting components and code, as well as a critical analysis of modern software-architecture conventions. The principles relating to the architecture of code which explored in the book are the SOLID design principles, which are summarised as follows.

summaris

- **Single-Responsibility Principle (SRP)**:
- **Open-Closed Principle (OCP)**:
- **Liskov Substitution Principle (LSP)**:
- **Interface Segregation Principle (ISP)**:
- **Dependence Inversion Principle (DIP)**:

The principles relating to the architecture of components which are explored Clean Architecture are divided into two groups: those which deal with component cohesion, and those which deal with component coupling. The principles relating to component cohesion are as follows.

- **Reuse/Release Equivalence Principle (REP)**:
- **Common Closure Principle (CCP)**:
- **Common Reuse Principle (CRP)**

The principles relating to component coupling in Clean Architecture are as follows.

- **Acyclic Dependencies Principle (ADP)**:
- **Stable Dependencies Principle (SDP)**:
- **Stable Abstractions Principle (SAP)**:

3 Requirements Analysis

3.1 Assumptions

4 Architecture

4.1 Modelling Language

To formally describe the final architecture produced for this report, the C4 model is used in combination with a notation similar to UML.

C4 Model

The C4 model primarily consists of four different architectural viewpoints, described in the form of diagrams, which represent an architecture as a hierarchical set of abstractions [3]. The viewpoints are designed to reflect to how software architects and developers think about and build software, and are summarised as follows.

- **System Context Diagram**: A viewpoint which describes a system's architecture as a whole, allowing viewers to see the "big picture". The diagram has a focus on the users of the system, rather than specific details, such as which technologies are used. Section 4.2 explores the system context diagram for the chosen architecture in this report.
- **Container Diagram**: A viewpoint which shows the details of a single subsystem within a system context diagram. The diagram is composed of containers, which represent an application or datastore (e.g. a web application, filesystem, database, etc.), and high-level references to other subsystems. Section ??? explores several container diagrams for the chosen architecture in this report.
- **Component Diagram**: A viewpoint which describes a single container within a container diagram, with respect to the major building blocks (i.e. components) that make up the container. Component diagrams contain some specific details, such as the technology used to implement or communicated between services, but still provide a relatively high-level overview of a container. Section ??? explores some component diagrams for the chosen architecture in this report.
- **Code Diagram**: A viewpoint which describes a single component within a component diagram as it is implemented within code, including the classes and interface involved, and their relationships. Code diagrams are not explored in this report, as they reference relatively low-level implementation details, and could be automatically generated from code written during the system's development.

The C4 model also includes three supplementary diagrams: the system landscape diagram, the dynamic diagram, and the deployment diagram. However, none of these are reference within this report, as the four core diagrams are sufficient to capture the chosen architecture [3].

Notation

find out from Ben & Jordan what notation is used

4.2 Overview

The architecture derived for the scenario listed in Appendix A is fundamentally a service-oriented architecture (SOA), as the key functionality of the system is implemented by multiple independent, distributed services. The core services used and their roles within the architecture are as follows.

- **Patient Information Service**: Handles create/read/update/delete (CRUD) operations relating to patient information, employing business rules in combination with authentication to authorise information appropriately. . This service is described further in Section 4.3.
- **Appointment Service**: Handles CRUD operations relating to appointments, employing business rules in combination with authentication to authorise information appropriately. This service is described further in Section 4.4.
- **Ambulance Service**: This service is described further in Section 4.6.
- **GP Notification Service**: This service is described further in Section 4.5.

does what? (see reqs)

does what? (see reqs)

which reqs are satisfied by which svc?

In order to enable user interaction with services, a client-server-like architectural style is employed for user-service interactions; Section 4.7 discusses this further.

Figure 1 displays the system context diagram (see §4.1), where the set of core services, users, and interactions can be seen.

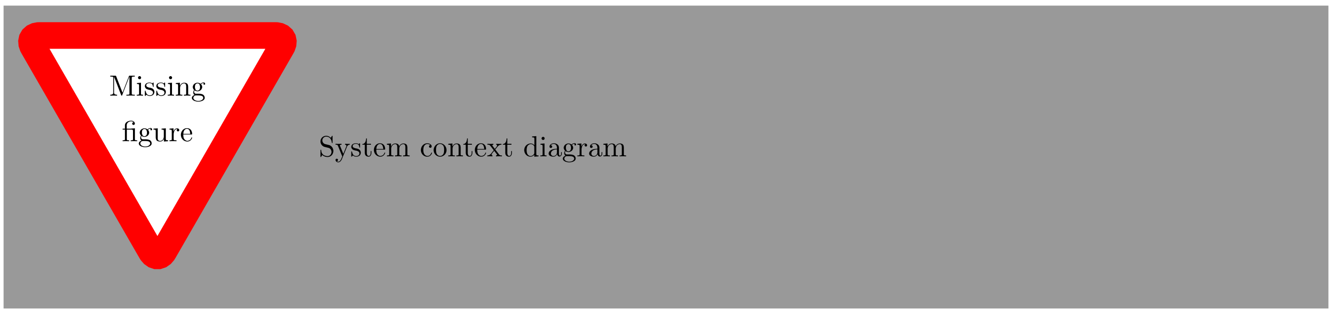

Figure 1: The system context diagram for the chosen architecture, displaying the service-oriented approach used as the fundamental architecture for the system.

As the target system's domains can be partitioned with relative ease, an SOA is a suitable choice for the overarching architectural style [4]. Although there are other applicable architectural styles for the target system, the following benefits of an SOA were the primary driving factors in the decision to prioritise it over other architectural styles.

- Services can be developed independently, reducing time-to-market, as independent teams can develop each service, and improving business agility, as each service's business rules can be adapted independently, provided the boundaries within the architecture are enforced effectively (see §)[5]. This is primarily an advantage over more rigid architectures, such as those using a monolithic style.
- Services can be deployed independently, facilitating the maintainability, short-term scalability (elasticity), and long-term scalability of the target system. This is primarily an advantage over architectures which are designed to have a one-to-one mapping from system instance to deployment machine, such as an architecture employing a layered monolithic style.
- Services are more independent with respect to failure than other architectural styles. This means that if one service fails, the failure of other services is not guaranteed, which is particularly beneficial for the target system, as the domains are mostly disconnected. For example, if the GP Notification Service fails, the other services in the system are not guaranteed to fail, meaning that the system will still provide the majority of its functionality to users. Overall, this makes the system more resilient to

ref SOA not an arch in Eval sec

- Similarly to failure, the services are more independent with respect to performance than other architectural styles. For example, if the GP Notification Service handles requests slowly due to a bug in its implementation, the Ambulance Service would be unaffected in terms of performance, provided it is deployed on a different machine; this is particularly beneficial for the target system, as some services, such as the Patient Information Service, require a certain level of performance to function as required.

- Due to the network-based nature of an SOA, services and clients are typically designed to be more robust to failure, improving the overall reliability of the system.

- Integration with external systems, such as the , is facilitated, as the target system will be deployed on a network due to the nature of an SOA. Similarly, integration with legacy systems which might need to be included in the newly developed system is facilitated [5].

hospital system

- The enforcement of architectural boundaries of the target system's domain is improved, as each service's boundary can be mapped directly to each domain's architectural boundary.

- The domain-driven separation of services, in combination with their low levels of interaction, mitigate the negative effects of an SOA, including selecting an appropriate granularity and handling service choreography/orchestration [4].

4.3 Patient Info Service

4.4 Appointments Service

4.5 GP Notification Service

4.6 Ambulance Service

4.7 Clients

client-server pattern / style

container diagrams for client

4.8 Authentication

5 Implementation

deploy on different nodes to ensure failures are indep If the target user-base was small, it might be worth using a single node for cost, but the size of th NHS user base is not small

6 Evaluation

7 Conclusion

see Clean Arch - says start off with single service with clear arch boundaries, the

References

[1] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017.

[2] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly, 2020.

[3] "The C4 model for visualising software architecture," Accessed: 2022-03-23. [Online]. Available: https://c4model.com/

[4] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly, 2020, ch. 13, pp. 163–177.

[5] "SOA (Service-Oriented Architecture), IBM," Accessed: 2022-03-23. [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/soa

[6] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Pearson, 2017, pp. 239 – 247.

A Scenario

You are required to develop an architecture for a patient digital appointments and management system.

This system should act as a central point for creating and delivering patient appointments as well as storing patient data. Digital appointments should be created via this system, allowing a GP to ascertain and monitor vital patient data. The patient's own data should also be stored in the system allowing for a full 360 degree view of patient information. This system is being built in order to alleviate pressure on the health service.

Each person resident in Scotland is registered with a General Practitioner (GP) practice. The health care service will maintain a record of each person's name, a unique health care id, date of birth, contact details, information on next of kin, the GP practice they are registered with and medical history. Patient records contain highly sensitive information. A patient may be seen via digital appointments or at the practice by a nurse (for minor injuries or tests) or a GP. A patient can only access the GP practice if they are referred to after a digital appointment, except in case of emergencies when they can either call an ambulance or go directly to the accident and emergency (A&E) department of a hospital. An ambulance crew will either treat a patient on site or take them to a hospital. A GP will either treat a patient themselves or refer the patient to a hospital run by the local health board, except when the required service is not available locally. Each appointment or treatment will result in an entry in the person's medical history, which should be accessible across different health boards and services. When a patient has been treated at a hospital or by an ambulance crew, a notification should be sent to their GP to flag up the new entry.

The GP should be able to generate digital appointments for patients. These appointments should allow the GP (via the patient's personal device) to obtain vital information about the patient – namely their blood pressure, heart rate, heart rate variability, oxygen saturation and respiration rate. This data should be generated in real-time and stored securely in the patient information system. The system should provide different functionalities depending on the category of user. A patient must be able to use the service to make appointments at their GP practice. They should also be able to see all the vital data generated by all their digital appointments in the system. GPs should be able to access the full record of any patient without delay, initiate appointments and add entries to the patient record. They should also be able to order one or more tests for the patient within the practice or refer the patient to a hospital, either within or outside the local health board. Nurses can see a limited part of the patient record and add entries relating to treatment, tests or test results. Practice administrators can make or cancel appointments for patients and produce statistical reports on the performance of the practice without accessing details of any patient.

An ambulance service administrator must be able to log calls to the service and dispatch ambulances to patients who require them. This aspect of the service is highly time sensitive. They can also update patients' records according to the service delivered. Ambulance crews indicate when each callout has been dealt with. Hospital doctors can view all the details of any patient they see and add entries on diagnoses made and treatment given.

A well-designed and implemented system should also support the following features:

- An intuitive UI appropriate to the user category,
- Access authentication for different categories of users and restriction of available information and functionality accordingly,
- Support for multiple types of devices,
- Concurrent access,
- Support for several different views and analyses over the data,
- Validation of input data where applicable,
- Generation of digital appointments and using patients' devices functionalities to extract the necessary patient data, and
- Deal with potential uncertainties when patients' devices cannot extract this data.