Edinburgh Napier UNIVERSITY
School of Computing, Engineering & the Built Environment

SET10101 Software Architecture Coursework

§ Architectural proposal

The design of the KwikMedical system must ensure certain requirements since the main objective is to help save lives. (Jordan, 2024) Hence the application's essentials are:

- Accessibility: The app must be easily accessible via multiple devices and locations such as Head Office, Regional Hospitals, and Ambulances. The main point of this is to enable users to retrieve and update data in real-time, ensuring that necessary data is delivered on time.
- Swiftness: In medical emergencies the response time and latency are crucial. Therefore all the delays and processing time must be minimal, including that only necessary information is transmitted as fast as possible, such as patient data(e.g. name, NHS number), locations(of emergency and ambulance) and dispatch requests.
- Reliability: Since the application is a lifesaver assistant, it must be fault-tolerant and robust. It needs to be able to handle high demand, provide accurate and damage-free data and most importantly be self-sufficient.

The two most suitable architecture types that align with those requirements are:

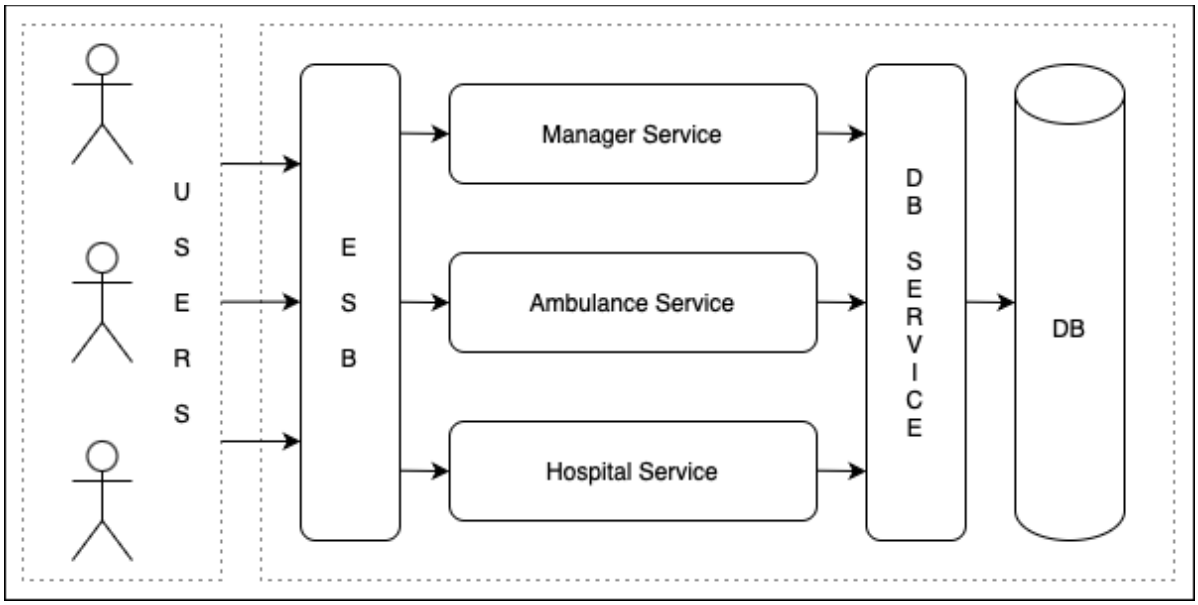
1. Service-Oriented Architecture (SOA) – is an architectural design type that separates parts of the application into modular services that communicate together with APIs. Services correspond to a specific function like patient data retrieval or dispatch requests.

   It is a viable option because of its flexibility which allows a high level of accessibility between different devices and locations. Furthermore, it supports real-time data sharing, is easily scalable and allows independent updates/changes to a service.

2. Event-driven microservices Architecture (EDMA) – is an architectural pattern that splits all the functions into microservices that work completely independently. Their communication is implemented with asynchronous events.

   This type of architecture is suitable because of its swiftness that's allowed by asynchronous processing. This type of architecture is superior regarding reliability, since the microservices in case of crash or fail can restart independently therefore not disrupting any of the other functions within the application.

Service-Oriented Architecture (SOA)



This SOA uses the main concepts of this architecture type such as modularity, scalability and loose coupling of the services. This allows the services to perform their assigned tasks, described below, the communication between services is made via Enterprise Service Bus. Due to this communication style, the system allows efficient coordination among all the users including operators, ambulances, hospitals and a centralised database.

Components

1. Users:
- Operators, Ambulance and hospital staff.
- Access the system via UI, which is connected to the ESB and consequently accesses back-end services.
2. Enterprise Service Bus:
- Acts as a communication hub that is mainly responsible for addressing requests to correct services.
- The only possible option of communication between the services.
- Handles main security features such as authentication and authorisation.
- Allows services to act autonomously which improves scalability.
3. Manager Service:
- Acts as a main information service that gathers essential data to initiate the rescue operation.
- Receives patient information from the operator.
- Gathers patient data from DB Service.
- Sends the ambulance dispatch request to the Ambulance Service.
- Share data with the Hospital Service to inform them about the ambulance that will be coming to the regional hospital.

4. Ambulance Service:
- Is accessed on portable devices (e.g. phones, tablets) installed in Ambulances.
- Allows staff to receive and confirm dispatch request(s)
- Access patient data and send Global Positioning Service (GPS) updates.
5. Hospital Service:
- Is a service that's used by Regional Hospitals.
- Receives ambulance arrival/position information from Ambulance Service.
- Sends access requests regarding patient data to the DB service.
- Updates patient records after rescue operation using DB Service.
6. DB (Database) Service:
- Is a service that is responsible for handling all database requests.
- Ensures data integrity and consistency due to indirect database access.
- Stores and manages all patient data.
7. Central Medical Database (DB)
- A centralised relational database that holds all patient data, dispatch requests and other essential data.
- Is only directly accessible to DB Service.

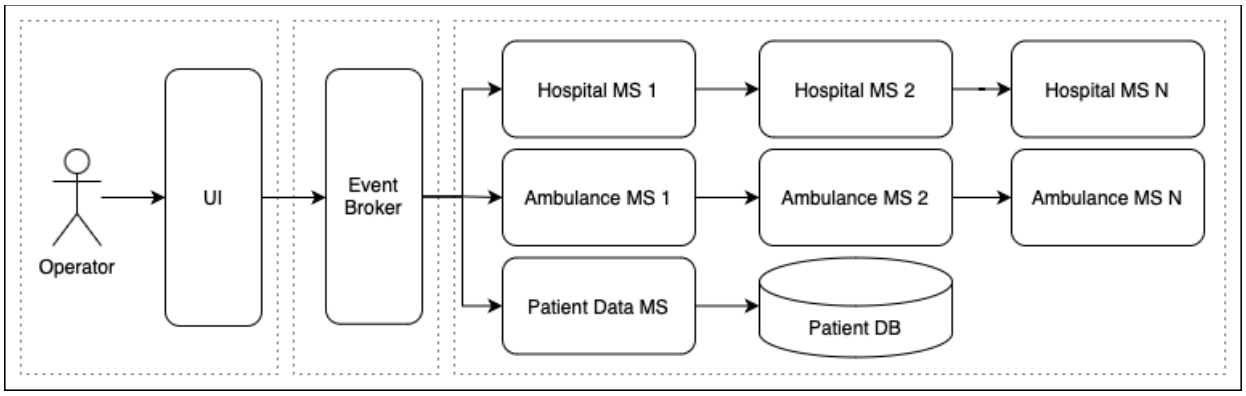
Pros & Cons

Pros	Cons
Loose Coupling (autonomous operating of components)	**Complexity (Requires complex implementation to ensure reliability and speed)**
Scalability	**ETS is a performance-heavy component**
Reusability	**Network dependent**
Maintainability	**Costly Deployment and Maintenance**
Technology-independent (services can be written in different programming languages)	**Data synchronisation and consistency is a questionable attribute in such architecture**
High Fault Tolerance	**-**
3rd party service Connectivity (due to REST standardised protocol usage)	**-**

Conclusion

The Service-Oriented Architecture (SOA) in the KwikMedical scenario is a viable option, since it provides a scalable and modular system that is split up into multiple independent components such as Manager, Ambulance, Hospital, Enterprise Service Bus(ESB) and DB Service. However, SOA also includes several unnecessary challenges, that could be evaded including increased complexity, potential delays through the ESB, and higher financial demands.

Event-Driven Microservices Architecture (EDMA)



The EDMA is a distributed architectural style, which is based on breaking down an application into separate and autonomous microservices, their communication is conducted via asynchronous events. This style is suitable for the KwikMedical application because of its real-time responsiveness, high scalability and minimised bottlenecks.

Components

In the emergency rescue app scenario, the app will be divided into a set of Microservices that are responsive to a particular task.

1. UI
- An operator upon receiving an emergency call enters basic identification information of the patient (e.g. Name, NHS reg. number, optionally Reason of Emergency)
- UI sends a dispatch request to the Event Broker, which then sends the event to the appropriate microservices.
2. Event Broker
- The Event Broker is the orchestration mechanism that ensures that the events are sent to subscribed microservices.
3. Hospital Microservices (Hospital MS 1, Hospital MS 2, ...)
- Each Regional Hospital has its own microservice that operates independently.
- When this microservice receives a dispatch request, it prepares for the patient's arrival based on the information provided.
4. Ambulance Microservices (Ambulance MS 1, Ambulance MS 2, ...)
- Those Microservices are subscribed to dispatch requests and are capable of accepting them accordingly. That allows rapid rescue operation of a person in distress.
5. Patient Data Microservice
- This microservice is an entry point to the database, it is subscribed for events such as obtaining patient data, gathering the information from the database and returning an update.
6. Patient Database – is a centralised database that stores all the needed data.

Pros & Cons

Pros	Cons
Enhanced Accessibility (Personal microservices for hospitals and ambulances)	**Complexity of the application (Message queuing, complicated debugging)**
Asynchronous data transmission (Low Latency)	**Event broker dependency (the system heavily relies on the singular entry point)**
High Reliability (Microservices ensure independent workflow, therefore they automatically restart a single component instead of a system)	**Development costs**
Easily Scalable (Because of its modular autonomous structure)	**-**
Performance (Could handle a large number of events)	**-**

Conclusion

The Event-Driven Microservices architecture is an exceptional pick as the emergency service rescue application architecture. It satisfies the core requirements such as:

- Accessibility and Reliability, since all the parts of the rescue operation are represented as separate microservices it allows easy access from any device in real time.
- Scalability, within this architecture scalability, is simple, because it doesn't require changing previously written microservices, which means, all of the new features can be built "on top" of other components.
- Swiftness, in emergencies the speed is the critical factor, and EDMA's asynchronous communication ensures the minimal possible latency.

The event-driven, decoupled structure aligns perfectly with KwikMedical's requirements, of high accessibility, low latency, resiliency and scalability. This architecture ensures prompt and secure data transfer in medical emergencies, which makes it a primary choice for the architecture.

Critical Evaluation

Considering those architectures for KwikMedical, both SOA and EDMA offer high advantages and theoretical disadvantages. However, EDMA occurs as a superior solution due to its higher responsiveness and flexibility.

EDMA is a distributed system in which autonomous microservices communicate via event broker allowing asynchronous, real-time data share. This principle minimises latency and ensures that essential data reaches the endpoint instantly. While SOA, which is based on synchronous responses might introduce delays, therefore making it inferior in an emergency scenario.

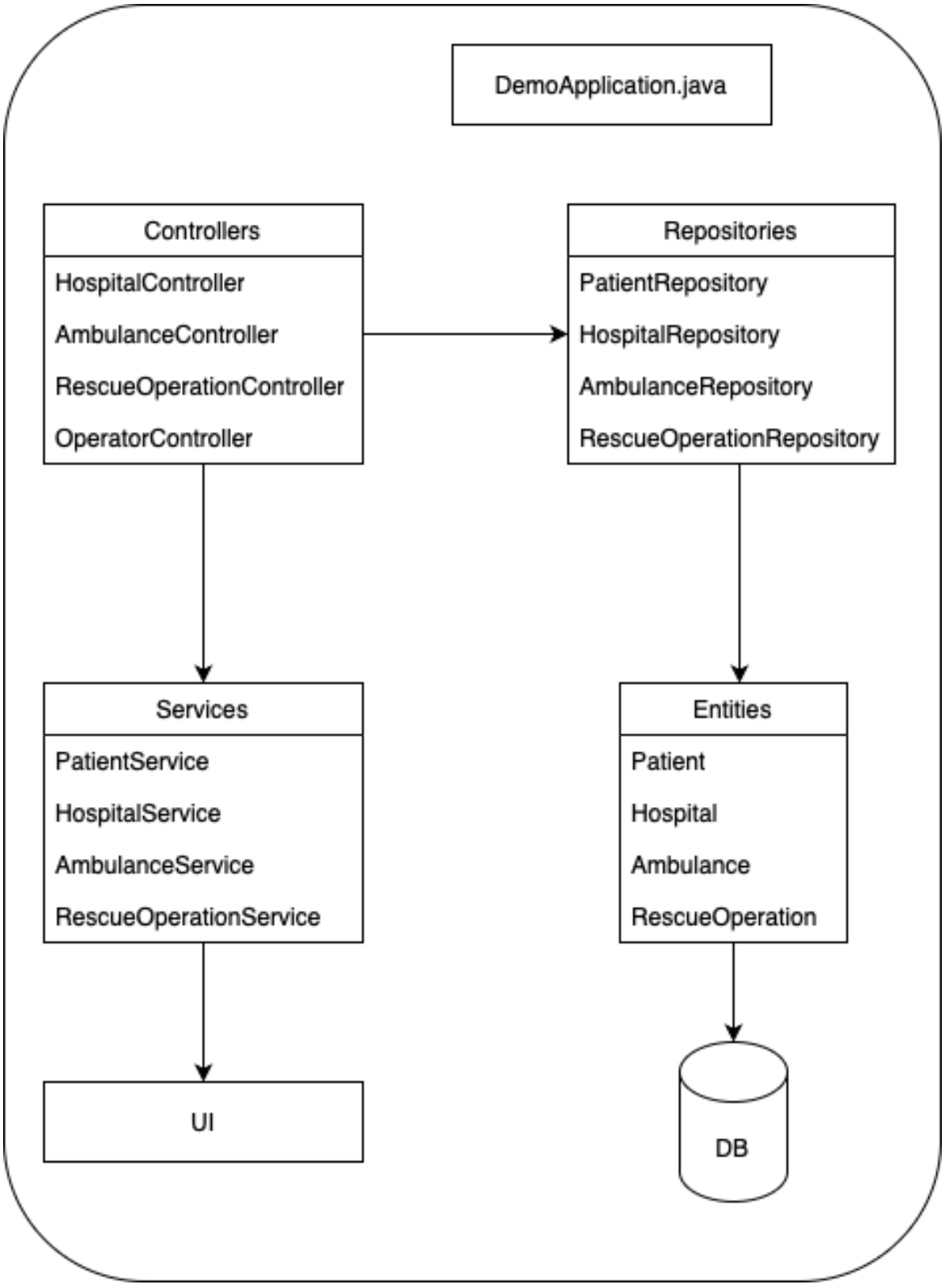
Furthermore, Event-Driven Microservices Architecture is more scalable and flexible, because the microservice can be scaled or updated independently, without disrupting other system components, in comparison to SOA which has large subsystem mechanisms that are autonomous but the functions within are not working independently.

Due to decoupled microservices, the system will remain stable while some of its component(s) might fail, additionally, they can restart automatically, which introduces the ease of maintenance in terms of speed and cost. In contrast, SOA has coupled services that might be vulnerable, if one of the components is malfunctioning it might lead to a system crash.

To recapitulate, SOA offers structured communication and simpler implementation in terms of complexity and building “on top” of existing systems, yet its synchronous style is affecting time efficiency, which is the most important factor in emergencies. On the other hand, EDMA offers asynchronous communication, better flexibility and scalability. Therefore, as stated above, it is a superior choice for an architectural style, that is based on fast and reliable delivery in medical emergency scenarios.

§ Prototype

Application Diagram (UML-based)



Description

The UML diagram represents the architecture and key components of the project. It consists of: Controllers, Services, Entities and Repositories.

Controllers

1. HospitalController
- Endpoint to access the Hospital page
- /hospital?hospitalId={id}
2. AmbulanceController
- Endpoint to access the Ambulance page
- /ambulance?ambulaceId={id}
3. RescueOperationController
- Backend endpoints for the creation and updates of rescue operations
- /rescue-operation/create
- /rescue-operation/update
4. OperatorController
- Endpoint to access Operator page
- /operator

Services

1. PatientService
- Gathers patients' data
- Validate data before sending dispatch requests
2. HospitalService
- Gathers Hospitals' data
- getAllHospitals() – returns all the hospitals
- getHospitalById() – returns specific hospitals using ID
- getHospitalsByLocation() – returns specific hospitals by location
3. AmbulanceService
- Gathers ambulances' data
- getAvailableAmbulances() – returns ambulances with the status 'Available'
4. RescueOperationService
- createRescueOperation() – creates a new rescue operation with parameters gathered from UI
- getRescueOperationsByStatus() – returns specific operations based on status

Repositories

1. PatientRepository – finds all the patients.
2. HospitalRepository – finds hospitals by address
3. AmbulanceRepository – finds ambulances by their status
4. RescueOperationRepository – finds rescue operations with assigned hospital & ambulance

Entities

1. Patient
- String: patientId {PK}
- String: name
- Int: age
- String: medicalHistory
- String: location
2. Hospital
- String: hospitalId {PK}
- String: name
- String: address
- String: contactNumber
3. Ambulance
- String: ambulanceId {PK}
- String: location
- String: status
4. RescueOperation
- String: operationId {PK}
- String: location
- String: status
- DateTime: createdAt

Critical Evaluation

The application prototype presents the integration of architectural principles studies over the duration of the module. It focuses on critical needs that are outlined in the initial planning of the application such as real-time responsiveness, medical needs and scalability. However, several key issues were identified that obstruct maximum performance and the proposed architecture of the app.

One of the main concerns lies in the redundancies that occurred due to inconsistent development. For instance, the PatientService and RescueOperationService have certain responsibilities that overlap. This issue increases complexity and hardens maintainability. To fix it simple refactoring is required.

Another concern lies in the incomplete usage of Event Driven Microservices Architecture (EDMA) which was chosen for asynchronous communication. Even though the application still uses Microservices architecture, not all of the components communication is made through the event broker, which is the main component of EDMA. To fix this issue, certain methods should be labelled as events and treated as such through the event broker, whilst at the moment functions such as confirm delivery from the hospital screen, are synchronous, which creates performance bottlenecks.

Furthermore, for future development the ambulance page should correspond to a particular ambulance (like a hospital page), to maximise productivity and minimise the connectivity of individual components to the larger functions.

Conclusion

The KwikMedical system demonstrates a robust understanding of different architectural styles, focusing on certain parameters of the app, as a guide to choosing the architecture itself. In this case, those parameters were accessibility, reliability and swiftness. By evaluating and comparing SOA and EDMA, the paper highlights the vital benefits and limitations of both of them, in the end, EDMA is chosen because of its asynchronous communication, high fault tolerance, scalability and lower latency.

However, the prototype implementation of EDMA reveals areas that require future development to finalise the product, such as resolving redundancies, enhancing the event-driven communication and utilising it in all sectors of the app for maximal efficiency.

To recapitulate, the Event Driven Microservices Architecture and its prototype is a superior choice for the given problem. It gives a proper basis for a reliable and scalable emergency rescue system. By fixing the identified issues, the system has the potential to assist in life-threatening situations with speed and precision, which are much needed in such scenarios.