

Smart clinical alarms service for healthcare professionals

Rafael Mulero Vellido, *Student, EPSEVG*

Abstract—This Final Degree Project (FDP) shows the process of creation and implementation of a service that will aim to inform physicians about the status of their patients in real time, speeding up the reaction time in cases where the health of these need immediate action. The service will follow a client-server model where the server will receive inpatient clinical data and, in case that this data gives signs of some critical or abnormal health status, will generate an alarm that will be sent to the client application installed on the smartphone from the physicians who are in charge of the patients.

Keywords—*Hospital, Clinical Alarm System, Clinical Decision Support System, Smartphone, e-health, Android, Web.*

I. INTRODUCTION

The management of information in medical area has undergone a great change in the last years, trying increasingly to change from physical to digital format (digitisation of clinical history). However, the speed with which this information arrives to the destination it should be improved, especially in cases where patients' life depends on, for example, a laboratory result. Nowadays, the amount of time elapsed between the publication of a laboratory result and the arrival of this result to the physician can vary between minutes, hours or, in the worst cases, days that can be crucial for patients. To solve this problem, there is what is known as Clinical Alarms Systems (CAS), which allows sending alarms to physicians when the health of a patient has changed or get worse. The main problem of CAS is that it may send too many alarms, not all of them being really important, making healthcare professionals ignore the alarms because they are being interrupted constantly.

Despite the use of CAS, once the information arrives to the physician, she has to review and check if the patient status is right or not. This checking can be easy if the amount of data is not very long, but as the amount increases or the information has to be cross-checked with other results it is easy to miss some information, which may increase the physician response time. To avoid this cases, there is what is known as Clinical Decision Support Systems (CDSS), which allows processing patient data to obtain different results that will be used in order to check if the physician has missed some information reviewing the results. Apparently CDSS seems to be very useful, but they lack of a system to make this information arrive to the final destination in a short time period.

At the same time, we are living in the age where mobile technology has become a key element in the day-to-day life,

This project has been done in Hospital Clínic i Provincial de Barcelona (HCPB) as a proposal from the Departament de Sistemes d'Informació (DSI) of the hospital itself with the goal of creating a useful service for the healthcare sector.

either for personal or professional area, and we are able to stay connected to the world from anywhere and to send and receive information in a matter of seconds. This project tries to enhance CDSS by using mobile technology as CAS in order to improve the response time of healthcare professionals in case of a potential risk for patients.

II. OBJECTIVES

A. Motivation

The motivation of this FDP comes from the need of reducing physicians' response time when the status of a patient is (or can become) critic, improving the quality of clinical service and patients' security.

The main idea of the project was to create a system that could be easily applied to any medical centre, but after the analysis and documentation reading done in order to check if this approach would be feasible, the conclusion was that it was very difficult to achieve this goal because of the high difference between the systems of each medical centre. That said, the service has been designed to be used by Hospital Clínic i Provincial de Barcelona services, concretely the emergency and internal medicine services. However, the system has been designed to be extensible, allowing continuing working on it to adapt it to more services inside the hospital.

B. State of the art

1) *Clinical Decision Support Systems*: First CDSS were created in the 70-80's thanks to the evolution of Information Technology applied to business [1]. First systems, known as **Knowledge-Based CDSS**, were very rigid and their basic functionality was to give a concrete diagnose based on data introduced by users. Later, in the 90-2000's, they became more flexible and they were able to offer some recommendations or possible diagnoses that the user had to check. In these CDSS, the intelligence of the system consists on a set of instructions that check if the introduced data generates an alarm.

The development of artificial intelligence and artificial neuronal networks [2] helped to create **Nonknowledge-Based CDSS** which differs from the previous ones in the fact that the intelligence of the system is not based on instructions because the system "learns" which results are expected as the user introduces different data. This type of CDSS is more complex but gives results with a higher precision than Knowledge-Based ones.

2) *Clinical Alarms Systems*: About CAS there are different articles and studies about the impact of this solution over the performance of the work on the hospital. In November 2015, a study of Colorado University concluded that 98.9% of the alarms generated by a system controlling the effects of opioid were not clinically relevant [3]. On the other hand, a study from Marshfield Clinic made on April 2015 concluded that applying some kind of logic on the alarms generated by the system, made them effective in 80% of cases [4].

3) *State of the art conclusions*: The conclusions extracted from the previous cases have been useful to have conscience that an alarm system without any kind of logic, intelligence or data processing generates too much "noise" because of the lack of relevance of the obtained alarms, making that professionals will end up obviating them. This FDP will try to combine a CAS with a Knowledge-Based CDSS in order to generate a set of alarms that will be useful for professionals without generating too much noise.

C. Requirements

The requirements gathered from the **users** have been classified in system, alarms, application and medical services groups.

System requirements

- The alarm must arrive as soon as possible to the physician
- The arrival of the alarm must be guaranteed
- Data privacy must not be compromised at any time
- Users must exist on HIS authenticating them via LDAP

Alarm requirements

- The alarms should indicate if they have been relevant or not
- The alarms must contain which parameters have generated them
- The alarms must contain who has seen them

Client application requirements

- The application must be easy to use
- The interface has to be user-friendly

Medical services requirements

- Some alarms have to compare previous and current data
- Some alarms have to check medicines administered to a patient
- It should be possible to automatically resend a generated alarm if no one has seen it in a certain time period

Each medical service has defined the **alarms** that should be checked by the system. Six alarms have been defined by the internal medicine service and 7 alarms by the emergency service.

Internal medicine service alarms

- Sepsis risk
- Hypotension with anti-hypertensive or diuretic medicine
- Heart failure
- High creatinine with a specific medicine
- Positive culture
- Hyperglycemia or hypoglycemia

Emergency service alarms

- Medical test validated
- Altered gasometry
- Renal insufficiency
- Ionogram
- Altered glycemia
- Altered haemoglobin
- Insufficient test sample

The **technical** requirements gathered have been classified in client application, server, database and software groups.

Client application requirements

- The application should run on mobile platforms
- The application should run on web platforms
- Mobile platform application must receive push notifications

Server requirements

- Low resource consumption
- Allow creating Representational State Transfer (REST) Web Services (WS)
- Allow configuring Hypertext Transfer Protocol Secure (HTTPS) access
- Easy to configure
- Must be able to send push notifications

Database requirements

- Accepting medium-high number of requests
- It has to be fast responding to requests
- It has to guarantee that data is maintained

Software requirements

- Free to use (open-source, free versions...)
- It has to be reliable (error-free, stable...)
- It has to be light (low disk storage consumption)

III. CHOSEN SOFTWARE AND TOOLS

A. Client application

The chosen software for implementing client applications has been Android for mobile application and HTML, CSS and JS for web application. **Android** [5] is an operative system based on Linux designed for mobile devices like smart-phones and tablets and has become one of the most used systems around the world. Its interface is based on the interaction between the user and the device screen, which makes it easy to use. **HyperText Markup Language (HTML)** is the standard language for creating web pages. As it is interpreted by the web browser instead of the server, it's possible to view web pages that are stored locally (it's not necessary to connect to a web server). **Cascading Style Sheet (CSS)** is a styling language used to define the appearance a markup language document will have. It was designed to split the content of a document from its presentation, making it easier to be interpreted by browsers, to change its styling and to reuse the same style across different documents. **JavaScript (JS)** is an interpreted programming language (executed using an interpreter, not a compiler). This interpreter is included in almost every web browser, that is, can be executed locally without the need of a dedicated server, as it happens with HTML. Not needing a compiler makes it flexible when developing web applications because it doesn't need neither too many resources nor too many time for being tested.

B. Server

The chosen software for implementing the server side has been Apache Tomcat for the server software, PostgreSQL as database manager and Firebase Cloud Messaging for sending push notifications. **Apache Tomcat** [6] is an open-source software which implements Java technologies for servers (servlets, websockets...). Allows creating servers without needing too many resources and with a very acceptable performance. **PostgreSQL** [7] is an open-source relational DB system. Allows manipulating a high amount of data in short time thanks to its scalability and assuring the stored data integrity. **Firestore Cloud Messaging** [8] is a web application development platform which provides connection between server and devices through cloud. It's totally free and provides support for Android, iOS, Web applications, C++ and Unity.

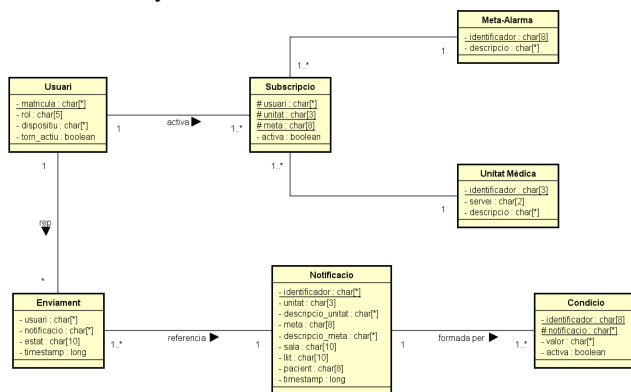
C. Tools

The chosen software for developing the project has been, Eclipse IDE, Astah modelling tool and Bitbucket code repository. **Astah** [9], previously knew as JUDE, allows creating different diagrams (use case, classes, sequence...) for designing the system before implementing it. **Eclipse IDE** [10] is an open-source multi-platform Integrated Development Environment (IDE) which offers different tools for Java programming, although there are different plug-ins for programming in many other languages. **Android Studio** [11] is an IDE focused on Android application programming. It offers tools for programming applications and an emulator to test these applications without the need of installing them into a real device. **Bitbucket** [12] is an on-line code repository service for projects using Git or Mercurial control version systems. It allows to have all the code stored securely and to make it easily portable to other systems.

IV. SYSTEM DESIGN

A. Database

Database is used by SCAS server for storing data related to service administration and alarm processing and sending. The following figure represents the relational model followed by DB, where primary keys are underlined and foreign keys are marked with # symbol.



User table makes reference to physicians and system administrator. They are defined by their user name in the HIS, a device token, a role and a field to indicate if they are on their work shift. The token is used to identify to which device will the push notification be sent, and the role will say whether the user is an administrator or not.

A **medical unit** makes reference to a concrete section from a hospital service (for example, thoracic pain unit from emergency service) from where the patient data will be analysed. It is defined by an identifier, the service to which the unit belongs and a brief description. A **meta-alarm** is the definition of which conditions have to be presented in patient data in order to raise an alarm. Into DB there are only stored the meta-alarm identifier and its description.

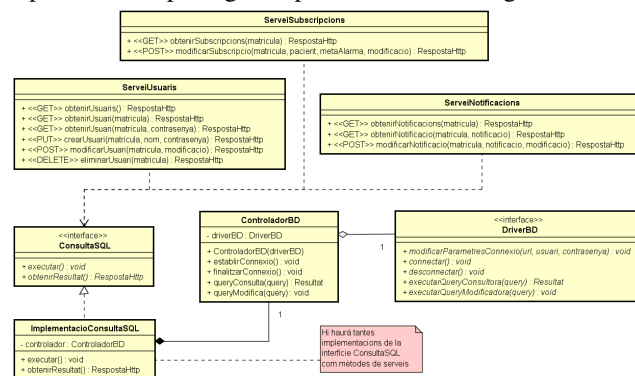
To allow the users to decide which alarms they want to receive and which not, a **subscription** that joins a meta-alarm and a medical unit is used. Every user will have a subscription to every unit and meta-alarm combination, but they will choose which ones they want to activate or deactivate.

Once the server checks if the patient data shows some risk, the system will generate a **notification** containing an identifier, the meta-alarm that has been used to determinate the risk, the medical unit, the room and the bed where the data has been taken from and the creation timestamp of the alarm. Along with the notification, the **conditions** matching the meta-alarm criteria will be included, so the physician can know more directly why the alarm has been triggered.

For each notification there is, at least, a **message** associating the user who will receive the notification and the notification that has been sent. The message will also have the timestamp when the user has received the alarm and a status that will help to know if the user has indicated, for example, that the alarm is a false positive.

B. Server

The server has been designed considering two functionalities: administrative and processing. Server **administrative part** allows clients to manage DB information related to users, subscriptions and notifications via Representational State Transfer (REST) Web Services (WS). The diagram of the components composing this part is the following:



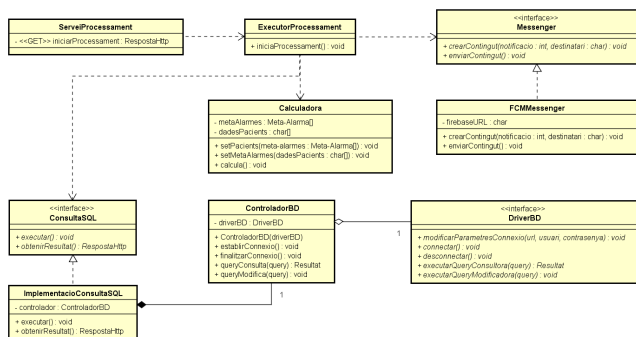
Service classes (ServeiUsuaris, ServeiSubscripcions, ServeiNotificacions) are in charge of publish the necessary

methods for managing the different operations (create, update, read and delete) that can be done into the DB

The SQL queries are defined by the ConsultaSQL interface. Each query requested to the DB is done using a class (ImplementacioConsultaSQL) which implements this interface. In this way, each class will have a single responsibility instead of grouping a set of queries into a single class.

Each query includes a DB controller (ControladorBD) which is in charge of sending the queries to the DB server and returning the response. To send the query, the controller uses a driver (which implements the DriverBD interface) that creates the connection to the DB and executes the queries over the data.

Processing part of the server is in charge of getting patient data, determine if they generate an alarm and, if they do, send the notifications to the users. The processing will be initiated by the SCAS or by the HIS instead of the user. The diagram of the components composing the server processing part is the following:

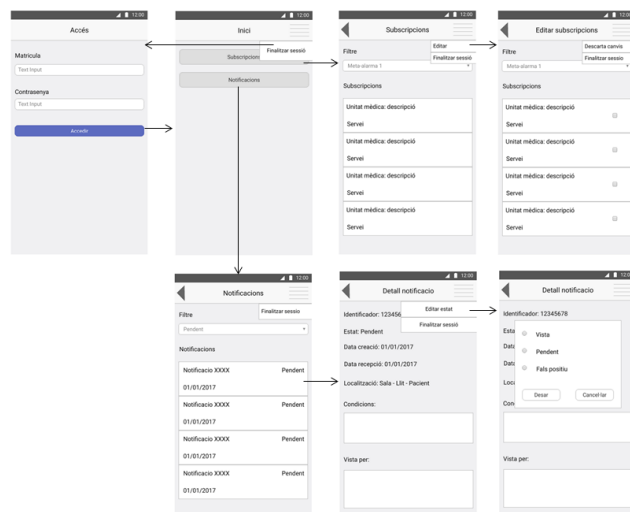


In this case, the **service** class (ServeiProcessament) publishes the method needed to let the server know that the processing should be initiated. Then the ExecutorProcessament class will act as an **orchestrator** to get data from HIS, calculate the alarms (Calculadora), perform the DB queries and send the **push notifications** (FCMMessenger) to the involved users.

C. Client application

Client application allows the user to interact with system data and, using the mobile version, receive push notifications generated by the service. To access the application, the user needs to authenticate using her user name and password. Once the user has been authenticated, she will be able to navigate to different sections depending on her role. In this way, users (with user or administrator role) can manage their subscriptions and notifications, but only administrator users are allowed to manage the system users.

Following figure shows a wireframe representing the approximate design the Android application will have and how the user will navigate through the different sections of it.



V. SYSTEM IMPLEMENTATION

The server side of the SCAS has been implemented on a Linux server using Java, Tomcat and PostgreSQL. Hypertext Transfer Protocol Secure (HTTPS) has been used in order to avoid that attackers can read the data sent between client and server if they intercept the connection. Also a connection pool has been configured for handling multiple connections to the DB at the same time. The specifications of the server are listed below.

- Operative System: SUSE Linux Enterprise Server 12 (x86_64)
- Hard-disk drive size: 40 GB
- RAM size: 4 GB
- Software specifications
 - Java: OpenJDK version 1.8.0_121
 - Tomcat: Apache Tomcat/8.0.36
 - PostgreSQL: version 9.4.9

As mentioned before, client applications have been developed in Android for mobile version and in AngularJs for web version. The functionality of notifications and subscriptions sections are nearly the same in both cases, but web application includes the user management section which is only accessible by an admin user. Android application does not include this section but allows receiving push notifications from the server. These notifications are encrypted in the server and decrypted in the application using an Advanced Encryption Standard (AES) algorithm to make sure that sensitive data is not compromised.

VI. RESULTS AND CONCLUSIONS

A. Testing results

Tests regarding **server security** have passed successfully, verifying that HTTPS was being used all the time and the users were authenticating correctly depending on if they were part of the HIS or not. The **requests to the server** retrieved the expected information for all the different requests done. **Client applications** were working as expected not only on showing the data but avoiding the access of the non-admin users to restricted sections.

Data processing tests have also been successful as the alarms were being generated correctly when the patient data was indicating a defined situation. Finally, **push notifications** tests have been done checking if the previously generated alarms were arriving correctly to the devices of the users who were subscribed to each alarm. All the alarms arrived correctly to the devices.

B. Conclusions

Working in a real environment and trying to include something new to it have been a challenging experience. Stick to times, deal with last-minute changes, depend on users' availability to gather the requirements and wait for the infrastructure to be ready have been the toughest parts of the project. But it has also been a very enriching experience as the learning has been constant and the interest of the people on the project has been very motivating.

The service has been designed and developed as a prototype to be tested by the users of the hospital. Currently the Android application has been installed in two devices, the server software has been deployed on a Linux server provided by the hospital in a local network for testing purposes, and the web application is located in the same server and it can be accessed from the hospital internal network. It's also important to say that the tests that have been done until now have been successful.

Regarding the personal learning during this FDP, I have learned how a project should be properly scheduled, to design a system and to develop it with clean structure and code, I have improved my knowledge on server and client side areas and I have discovered new security techniques for creating a secure system. The feelings about the project are very motivating as it has been kindly accepted by the people of the hospital.

C. Future work

The **short term** actions are related to the integration between SCAS and HIS as the access to real data is not possible right now because there are some services that are being implemented currently.

Mid term actions are related to the testing of the SCAS by the users of the chosen medical services in order to get feedback about different aspects to improve it constantly until the users feel comfortable with it. The service should be moved to the production environment and the users should keep using it for checking that it's still working properly. If everything works as expected, it should be expanded to more medical services and keep working on improvements until it is finally fully integrated into the hospital.

Finally, **long term** actions are related to moving the service to a more sophisticated system using, for example, artificial intelligence or adding new functionalities like a meta-alarm generator for the users.

REFERENCES

- [1] Martin Alther and Chandan K. Reddy. "Clinical Decision Support Systems". In: *Healthcare Data Analytics*. Chapman and Hall/CRC, 2015. Chap. 19, pp. 625–656. URL: <http://www.crcnetbase.com/doi/abs/10.1201/b18588-23>.
- [2] Wikipedia. *Artificial neural network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.
- [3] Emma K. Genco. "Clinically Inconsequential Alerts: The Characteristics of Opioid Drug Alerts and Their Utility in Preventing Adverse Drug Events in the Emergency Department". In: *Annals of Emergency Medicine* 67.2 (2016), pp. 240–248. URL: [http://www.annemergmed.com/article/S0196-0644\(15\)01308-6/abstract](http://www.annemergmed.com/article/S0196-0644(15)01308-6/abstract).
- [4] Sara Griesbach et al. "Best Practices: An Electronic Drug Alert Program to Improve Safety in an Accountable Care Environment". In: *Journal of Managed Care & Specialty Pharmacy* 21.4 (2015), pp. 330–336. URL: <http://www.jmcp.org/doi/abs/10.18553/jmcp.2015.21.4.330>.
- [5] Google. *Android*. URL: https://www.android.com/intl/es_es/.
- [6] Apache. *Apache Tomcat*. URL: <http://tomcat.apache.org/>.
- [7] The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/>.
- [8] Google. URL: <https://firebase.google.com/products/cloud-messaging/?hl=es-419>.
- [9] Change Vision. *Astah - Software Design Tools for Agile teams with UML, ER Diagram, Flowchart, Mindmap and More*. URL: <http://astah.net/>.
- [10] Eclipse. *Eclipse IDE*. URL: <https://www.eclipse.org/>.
- [11] Android. *Android Studio*. URL: <https://developer.android.com/studio/index.html?hl=es-419>.
- [12] Atlassian. *Bitbucket*. URL: <https://bitbucket.org/>.