THIN CLIENT FOR REAL-TIME MONITORING OF COMMUNICATION INFRASTRUCTURE

Iulian ILIESCU, Titus BĂLAN, Oana GAROIU, Sorin ZAMFIR

"Transilvania" University, Braşov, Romania (titus.balan@unitbv.ro)

DOI: 10.19062/2247-3173.2016.18.1.35

Abstract: *Monitoring the telecom infrastructure reflects not only a technical perspective but also a business perspective, highlighting, besides possible technical issues, the network areas that need improvements or investments. This paper describes the method for tele-monitoring of communication infrastructure responsible for switching and signaling in the mobile converged core network. The application allows the display of real-time information on a thin client. Thus, decisions related to infrastructure (resource management, optimizations, redeployments) can be taken based on notifications received on a mobile device (e.g. an Android smartphone)*

Keywords: *Real-time monitoring, thin client, Spring Framework, Data Access Objects, REST, Communication Infrastructure*

1. INTRODUCTION

Network operators are dealing with the challenge of reducing operational expenses OPEX and optimize the functionality of distributed infrastructure, within fast response times or methods to predict and avoid network issues, through automatic maintenance solutions.

The purpose of the implementations presented in this paper is to obtain a remote monitoring solution that provides mobility and which has an interface accessible to any user. The user has access to data wherever they are, as long as an Internet connection is available. The scope of the implemented demonstrator is to obtain an intuitive system, easy to set up and use, with the possibility of further development, while providing a very good performance / price ratio. The user has the possibility to view graphs that highlight errors, can analyze detailed view of logs and can perform real-time monitoring of the activities of system responsible with backup actions. Data communication between the server and the Android application is achieved through REST Web services ("Representational State Transfer") [1]. The multiplicity of standards and protocols used in the Internet have made possible the communication between two or more systems connected to the Internet that is a distance from one another. Industry software development, architectural models based on web services are seen increasingly more often, offering numerous advantages, especially when used together with an DAO ("Data Access Objects")[2] type architecture that brings an innovation in the interpretation of data and modeling of data objects. This concept will be detailed later, with examples related to the demonstrations.

2. SPRING FRAMEWORK AND DATA ACCESS OBJECTS

The used architecture is the type Model-View-Controller (MVC Spring framework)[2]. This architectural model allows the isolation of business logic from the

user interface, resulting in an application that allows easy modification of the user interface or even the modification of the lower levels without affecting other layers.

2.1 Spring Architecture

Spring framework was introduced in 2002 by Rod Johnson in "Expert One-on-One J2EE Design and Development" [3] that was initially a simple framework that provides the ability to connect applications together. This framework enables the development and execution of Java applications and offers various organizing features and various APIs (Application Programming Interface - Libraries specialized functions), such as JDBC, JSP, Servlet.

Spring Framework has a well-defined modular structure, consisting of 7 modules[4]:

- Spring AOP (Aspect Oriented Programming) - similar to the idea of object-oriented programming that modularize application in hierarchies of objects, oriented programming aspects breaks the program in aspects.
- Spring ORM (Object Relational Mapping) - is a tool used to connect to databases. It provides APIs for manipulating databases, tools such as JDO, Hibernate and iBatis. In our case, Spring ORM supports DAO pattern.
- Spring DAO (Data Access Objects) - has the role standardization of communication between the web application and database, representing an abstract level between physical data and objects used in the application.
- Spring Web MVC (Model View Controller) architecture implements the MVC web application. Separate application modules in the 3 defined by naming and thus organizing application on 3 levels.
- Spring Web - is a sublevel of Spring Web MVC module.
- Spring Background - is a package built around package "beans" to provide additional sources of APIs
- Spring Core - the most important component of the Spring framework, one that encompasses the injection of outbuildings.

2.2 Model – View – Controller Architecture

Architecture MVC (Model - View - Controller) implements the idea of separating data access and business logic from the user interface. As the name implies, the MVC pattern can be broken down into three components:

- *Model* - is data and data access rules. The software is an approximation of a process of everyday life.
- *View* - returns the contents of a model. Specify exactly how an object should be shown and this presentation is changed whenever the model is changed.
- *Controller* - is performing translation of user interactions with View into actions executed by Model.

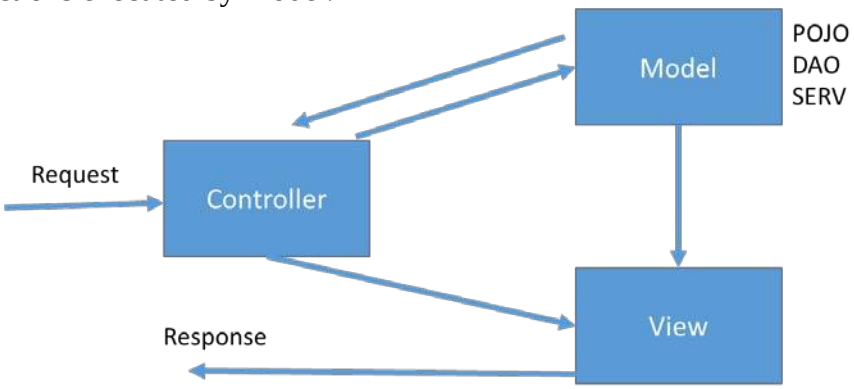

FIG. 1. The Model – View – Controller architecture

3. IMPLEMENTATION SCENARIO FOR MONITORING TELECOM INFRASTUCTURE ELEMENTS

We have chosen to demonstrate the concept by monitoring large Telecom infrastructure elements, main elements for mobile switching center and signaling central elements, that produce a big amount of logfiles, difficult to track and filter:

- EWSD (Elektronisches Wählsystem Digital/Electronic Digital Switching System)[5], produced by Siemens, is one of the most used switching systems, besides having capabilities for advanced Intellginet Networks, offering also services for ISDN (Integrated Services Digital Network), SS7 and Global System for Mobile (GSM).
- The hiS 700 system, developed by Nokia Siemens Networks [6] offers a hardware implementation for the Signaling System number 7 (SS7) used in the core network of mobile communication infrastructure. All the network signaling traffic is routed through this system, positioned in the center of the network, implementing the distribution, conversion and coordination functions for the signaling traffic.

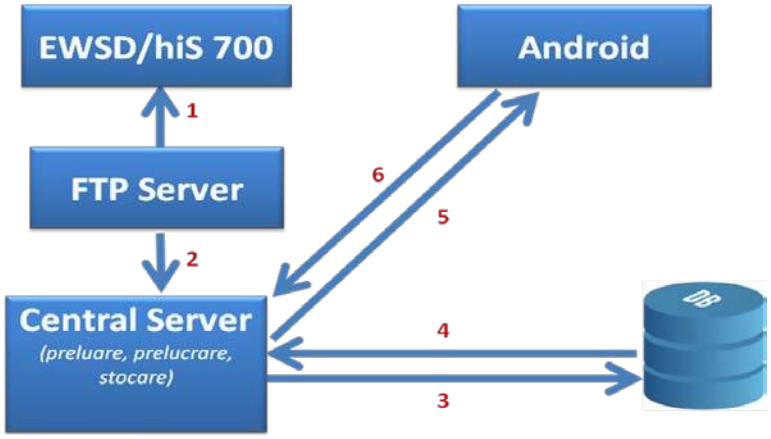

FIG. 2. The working model of the designed system

The system has five main modules that interact through web services of type REST and ORM services. (Fig.2.)
The 5 modules are:

- Switching equipment - these are the system EWSD equipment and signaling system hiS 700
- FTP server - is the server that communicates directly with devices (relationship 1 in Figure 4.1), transferring data via FTP communications and sending them to the web server
- Central Server - is the web server developed in this project, implemented in Java, left running through Apache Tomcat server; is the link between system modules: receiving data from the FTP server via web services (relationship 2 in Figure 4.1), communicated through services ORM to the database (relations 3 and 4 in Figure 4.1), posting data through Android app (5 the relationship of Figure 4.1)
- Database - represented by PostgreSQL, located on the same server as the web application
- Android client - in the project are used two terminals: GenyMotion for simulating mobile device Google Nexus 4 and Samsung I9500 Galaxy S4 smartphone; the data from the server are read by the mobile devices through Retrofit library

4. SOFTWARE SOLUTION ARCHITECTURE

4.1 Web Server

The server is represented by a Java web-developed under Eclipse IDE. This solution is designed to link the physical and application data Client (Android) using REST web services for reading and publishing data received from the server where the data are written. The web server communicates with relational database PostgreSQL through Hibernate ORM libraries [7]. Web services enable communication between the database and Android app.

Initially a *http get* call is performed, to read data from another server (that store physical logs), these data are entered into the database and posted (*http post*) to an IP address that can be accessed by application Client.

Communication via Hibernate ORM database allows access to secure data, by the usage of configuration files. Hibernate ORM enables data modeling and encapsulating them into objects. PostgreSQL is the database that stores application models (objects) in a table form. It is a database accessible from the web that works with SQL.

Transferred data are of two types (*Models*): „procedures" and „logs" (see Fig.3.) MVC architecture, with its elements: model, view, controller allow manipulation of data as objects. Controller is the main functional component of the system, that creates web services responsible for data transfer. Modeling objects with DAO (Data Access Objects) are used to organize communication with the database. For each type of object modeling involves using three different classes which must include:

- Defining the attributes and behavior patterns
- Defining interfaces for database persistence with standard functions (saveOrUpdate, findById, delete, findAll, deleteAll)
- Implement interfaces using the library "org.hibernate" to start the session database

Defining objects is done according to information received from the FTP server and according to what is meant to be inserted into the database.

```java
@Entity
@Table(name = "procedures")
public class Procedures {

    @Id
    @Column(name = "id")
    private int id;

    @Column(name = "element_name")
    private String elementName;

    @Column(name = "begin_timestamp")
    private Date beginTimestamp = new Date();

    @Column(name = "end_timestamp")
    private Date endTimestamp = new Date();

    @Column(name = "procedure")
    private int procedureInt;

    @Column(name = "status")
    private int statusInt;

    @JsonManagedReference
    @OneToMany(fetch = FetchType.EAGER, mappedBy = "procedure")
    private List<Logs> logs;
```

(a)

```java
@Entity
@Table(name = "logs")
public class Logs {

    @Id
    @Column(name = "id")
    private int id;

    @JsonBackReference
    @ManyToOne
    @JoinColumn(name = "procedure_id")
    private Procedures procedure;

    @JsonDeserialize(using = DateDeserializer.class)
    @Column(name = "timestamp")
    private Date timestamp;

    @Column(name = "operation")
    private int operationInt;

    @Column(name = "status")
    private int statusInt;

    @Column(name = "step")
    private int stepInt;

    @Column(name = "command")
    private String command;

    @Column(name = "information")
    private String information;

    private int procedureId;
```

(b)

FIG. 3. Structure of the models for (a) "procedures" and (b) "logs"

4.2 Client Application

Client application interface consists of an application developed for the Android operating system, adaptable to different system version. Is developed in Eclipse IDE plug-in for Android [8], which allows the integration of many bookstores, such as for graphing to web services REST, reading data in predefined formats such as JSON. These libraries will be elaborated further by example and illustration [9].

The same MVC architecture is used to implement Client application model and is defined by the type of data (procedures and logs), vision being implemented in major classes of the Android app (activities). This is the one that uses models as data controller CPC as in the definition of REST web services.

The end user of the developed system accesses data by the use of this application. Security is achieved through the mobile app authentication [10] and access restricted to a web server company level "Intranet". Once authenticated, user can monitor the logs in real time, can produce reports and can supervise the copying files system by viewing the status of orders executed.

5. ANDROID CLIENT IMPLEMENTATION

The purpose of the application is tele-monitoring a system used for storing records retrieved from a switching equipment using an intuitive workflow. A user firstly accesses the application to see if there are uncompleted or completed executions error. To do this, the user must log into the system and access the menu option errors. This option, illustrated in Fig.4., lets the user to view records of procedures executed with error. A list of jobs that are with active error status is displayed and the details of the errors can be expanded, detailed steps are displayed for each job. For each details, steps can be viewed, so the user can know at which stage the error occurred.

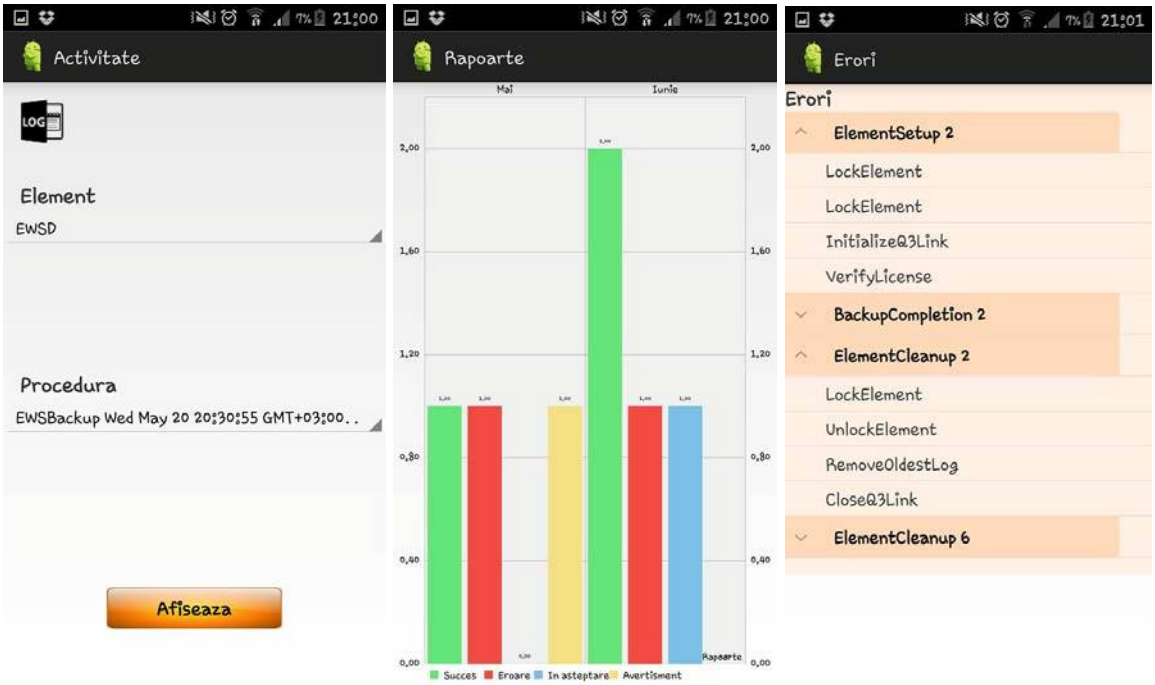

FIG. 4. The Android user application interface: Activity Options, Reports and Execution Errors

If the user wants to watch the entire list of records he will accessed the *Activity* options from menu (Fig.4.) This displays the list of all entries according to criteria chosen by the user (item were executed and procedure executed).

Another functionality of the application is the report generation . The reports can be viewed in two ways:

- Linear graphs with the number of procedures performed each month, depending on their status
- Circular graphs with the total number of procedures, separated according to their status resulting from execution

7. APPLICATION TESTING

Testing of the whole process was achieved by simulating the data transmission-reception.

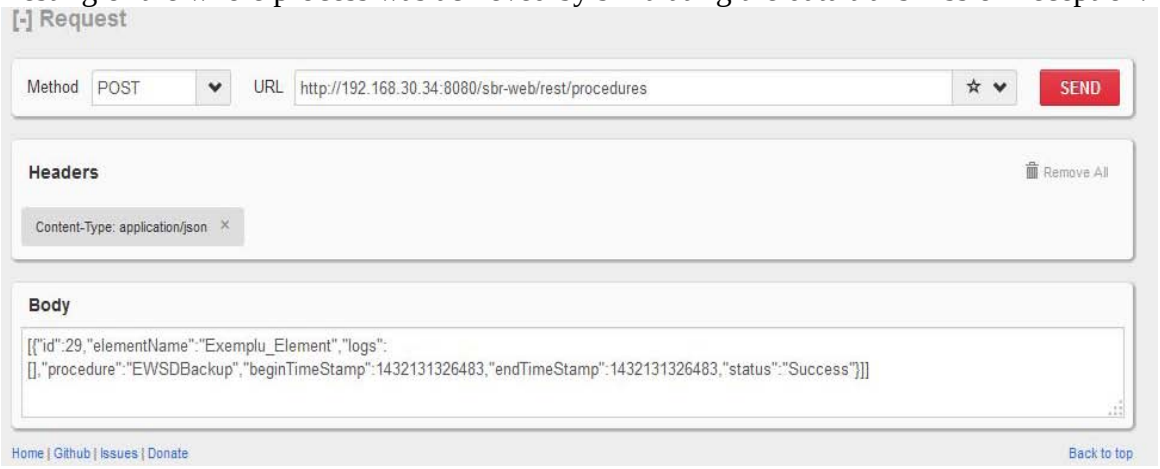

FIG. 5. Simulating a REST client

For transmission, Rest Client application was used from the Mozilla browser by using a sequence of a JSON file. There were two different formats of JSON files, which are retrieved from the server response. REST client interface provided by Mozilla is shown in Fig.5. This allows testing of REST web services and viewing the result. If execution is successful, the outcome is 200 OK.

CONCLUSIONS

The implemented system emphasizes the need for a "console", represented in this case by Smartphone, in monitoring the switching infrastructure of a telecommunications network. "The Console" allows the access to information infrastructure and at the same time provide users mobility.

Client-server model was developed by exploiting the model objects of Object Oriented programming, the data being transmitted and received as models. To transfer data between client and server, REST web services were used for reading and writing files in JSON format and transmitted to the server. Architecture used is the type Model-View-Controller (MVC Spring framework-).

As implementation, the system has two major advantages:

- It is adaptable to the desired execution platform, being developed mainly in Java, whose slogan is "Write once, run anywhere". The system is designed to have as many logical interfaces, not physical, exploiting the idea of class objects.
- The interfaces are generic, allowing modification of the application modules without impacting the system as a whole.

REFERENCES

[1] D. Amritendu, „Spring, Hibernate, *Data Modeling, REST and TDD: Agile Java Design and Development"*, Cathy Reed, 2014
[2] G. Amuthan G, „*Spring MVC: Beginner's Guide"*, Packt Publishing, 2014
[3] R. Johnson, *One-on-One J2EE Design and Development*, Wiley, ISBN: 978-0-7645-4385-2, October 2002
[4] Spring Framework, http://docs.spring.io/spring/docs/current/spring-framework-reference/html/, Accessed March 2016
[5] Siemens Networks LLC, „*Documentation Catalog for EWSD – Digital Electronic Switching System"*, 2006
[6] Nokia Siemens Networks, „*Updated brochure of hiS 700"*
[7] C. Bauer, G. King, G. Gregory, „*Java Persistence with Hibernate"*, Manning Publications Company, 2015
[8] Ziguard Mednieks, Laird Dornin, Blake Meike, Masumi Nakamura, *"Programming Android: Java Programming for the New Generation of Mobile Devices"*, O'Reilly, 2011
[9] Chris Haseman, „*Android Essentials"* Apress, 2008
[10] Ziguard Mednieks, Laird Dornin, Blake Meike, Masumi Nakamura, „*Programming Android: Java Programming for the New Generation of Mobile Devices"*, O'Reilly, 2011

ELECTRICAL AND ELECTRONICAL ENGINEERING / RENEWABLE ENERGY AND ENVIRONMENT