Received May 21, 2019, accepted July 14, 2019, date of publication July 22, 2019, date of current version August 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2930401

Layered Software Architecture for the Development of Third-Generation Video Surveillance Systems

YAIR VIVEROS MARTÍNEZ[1], EDUARDO LÓPEZ DOMÍNGUEZ[1], YESENIA HERNÁNDEZ VELÁZQUEZ[1], SAÚL DOMÍNGUEZ ISIDRO[1], MARÍA AUXILIO MEDINA NIETO[2], AND JORGE DE LA CALLEJA[2]

[1]National Laboratory of Advanced Informatics, Veracruz 91100, Mexico
[2]Universidad Politécnica de Puebla, Puebla 72640, Mexico

Corresponding author: Eduardo López Domínguez (eduardo.lopez@lania.edu.mx)

This work was supported by the Consejo Nacional de Ciencia y Tecnología, project assigned to LANIA, under Grant 296230.

ABSTRACT Mobile distributed systems of third-generation video surveillance (MDSV) have become a useful tool to provide multiple security services to people. For this type of systems, three key aspects must be carried out: 1) protection, which consists of preventing undesirable events; 2) detection, which refers to determining the exact moment in which the event occurred; and 3) reply, in this regard, actions such as activating alarms and generating warnings are executed. Previous works have proposed software architectures to development video surveillance systems on mobile distributed systems (MDS). However, these architectures focus mainly on providing services/aspects of protection and detection; without considering in its design the requirements that arise from the characteristics of the MDS, such as limited processing and storage capacities of devices, frequent disconnections, among others. In this paper, we introduce a layered software architecture to build MDSV. The proposed architecture considers and satisfies the requirements that arise from the critical aspects of protection, detection, and reply, including the characteristics of the MDS. Based on our architecture, an MDSV prototype was implemented. The tests carried out on the prototype show that the proposed architecture correctly provides users with various essential services in terms of protection, detection, and reply. From our point of view, the most important advantages of our proposed software architecture are the following: define the basic technical guidelines that an MDSV must have and accomplish; streamline overall development, providing a solid framework for developers; and contribute to satisfying the requirements that arise from quality attributes that the MDSV must possess.

INDEX TERMS Mobile distributed systems, third generation video surveillance, layered software architecture.

I. INTRODUCTION

Video surveillance systems can be defined as a set of devices with the ability to capture images of events within a specific area [1], [2]. Based on the use of image processing technology and the type of communication, video surveillance systems are classified into three different generations [2]. The first generation is characterized by the use of an analogous closed circuit, which depends entirely on people that watch screens to detect irregular situations [2]. The second generation also uses a closed-circuit but offers semi-automatic solutions by implementing machine learning and computer vision techniques to process the captured information of the area; technically, the second generation loads and processes information at the server-side [1], [3]. The third generation of video surveillance systems includes computer vision and data recovery techniques, which is mainly characterized by the use of distributed and heterogeneous systems. The third generation takes advantage of the devices on a network to distribute processing and to integrate sensors that enable the detection of intruders inside a monitored area, providing scalability and robustness to video surveillance systems [1], [4]. Some elements of these systems are the following: specific sensors, mobile devices, LAN or WAN type networks, databases and

The associate editor coordinating the review of this manuscript and approving it for publication was Shuiguang Deng.

proxies processing servers [4]. According to [5], the third generation of video surveillance systems that use mobile devices should support services for three key aspects:

- Protection: consists of preventing undesirable events.
- Detection: determines the specific time on the occurrence of an event and is helped by the protection mechanisms.
- Reply: refers to actions after an event has occurred such as activating alarms or generating warnings.

Previous software architectures to support the development of third-generation video surveillance systems by using mobile devices (MDSV) have been reported in the literature [3], [4], [6]–[12], nevertheless, these architectures are mainly focused on providing protection and detection services and they do not take into account the features of mobile distributed systems (MDS) such as devices with limited processing and storage capacities, frequent disconnections, communication channels with limited bandwidth and information management from heterogeneous sources.

This paper presents a layered software architecture to develop MDSV systems. The proposed architecture supports protection and detection services as well as reply. The designed architecture is based on a pattern that uses layers and tiers, the vigilant tier, and the analyst tier. In the proposed architecture, the tiers complement the layers; the vertical organization of services is modeled in abstract layers, while tiers organize the functionality of a specific layer into nodes. On the one hand, the vigilant tier is oriented to work with limited processing and storage capabilities of mobile devices. On the other hand, the analyst tier is designed to work with devices with greater processing and storage capabilities such as servers; this is also related with the bandwidth of the MDSV network. Therefore, workload and processing information is higher in the analyst tier than in the vigilant tier. The proposed architecture identifies, classifies, groups and orders the services for protection, detection, and reply for MDSV systems. Furthermore, the paper describes a prototype that implements the proposed architecture. Prototype tests show that architecture provides users with the correct implementation of protection, detection, and reply services. From our point of view, the most important advantages of the proposed architecture are the following: a) define the basic technical guidelines that an MDSV must have and accomplish; b) streamline overall development, providing a solid framework for developers; and c) Contribute to satisfy the requirements that arise from quality attributes that the MDSV must possess.

II. STATE-OF-THE-ART

Previous software architectures to support the development of MDSV have been reported in the literature [3], [4], [6]–[12]; the design and development of these systems involve the consideration and satisfaction of different requirements that can be classified into two groups. The first group comprises requirements that regard with protection, detection and reply services [4], [5], for example, sensing, information processing, real-time monitoring, and detection of abnormal situations or control of alarms. The second group considers requirements that emerge from the features of an MDS such as lightness, the security of devices, minimize network traffic, robustness against disconnections, the abstraction of the data transmission medium, communication and temporary storage of information about events [13]. The main software architectures to support the development of MDSV systems were analyzed in terms of previous requirements. Research works reported in [3], [4], [6]–[12] cover protection and detection services of video surveillance systems, for example, the architectures proposed in [3] and [9] cover many of the described requirements, however, [3] they does not consider surveillance calibration, vigilant device status notification and control of alarms; the latter is related to the reply service, neither information processing, integration of information from heterogeneous sources and detection of abnormal situations are not taken into account in the architecture proposed in [9]. The architectures described in [8] and [10] accomplish most of the described requirements arising from the MDS characteristics, although they do not consider the security of devices, minimization of traffic network or requirements of group communication.

III. ANALYSIS AND DESIGN OF THE PROPOSED LAYERED SOFTWARE ARCHITECTURE

This section presents the analysis and design of the proposed layered software architecture to support the development of MVDS systems. This architecture is based on the following requirements:

1. Implementation of protection, detection and reply services, and
2. Features of mobile distributed systems.

Next section describes the details of those requirements.

A. REQUIREMENTS TO DEVELOP THIRD-GENERATION VIDEO SURVEILLANCE SYSTEMS

The design requirements of the proposed layered software architecture to support the development of third-generation video surveillance systems adopt the features of previous works in terms of protection and detection services described in [3], [6], [8], [10] as well as the reply service. The description of some requirements is as follows:

- *Sensing (Sen):* refers to media and actions to gather information about a monitored area [13].
- *Processing of information (PI):* analyzes if the processing of information is considered to mitigate possible errors [3].
- *Real-time monitoring (RTM):* provides of video or audio of the monitored area in real-time.
- *Environment change adaptation (ECA):* refers to the capacity of the architecture to operate under changes in the conditions of the monitored area such as modifications of scenes or lightning [10].
- *Integration of information from heterogeneous sources (IIHS):* considers the integration of information from

different devices on the same event in the monitored area [10].

- *Surveillance calibration (SC):* considers the system calibration for its correct operation (parameter setting for sensors operation, microphones and cameras) [8].
- *Abnormal situations detection (ASD):* considers training, detection and learning of abnormal situations to identify potential risk situations [6].
- *Vigilance device status notification (VDSN):* notifies the status of vigilant devices to guarantee their correct functionality.
- *Storage (S):* stores information from a monitored area.
- *Alarm control (AC):* activates or deactivates notifications or implements specific actions such as calls, doors and windows closing.

B. REQUIREMENTS FOR MOBILE DISTRIBUTED SYSTEMS

The development approach of video surveillance systems on mobile distributed systems considers the following requirements [6], [13]:

- *Lightness (L):* mobile devices have limited processing and storage capacities; these features must be considered for light management [13].
- *Security device (SD):* security refers to the fact that only known devices can be connected to the video surveillance system.
- *Vigilant device health (VDH):* models the responsible elements for reporting critical features of devices such as level of charge of the battery, workload processing or temperature.
- *Minimize network traffic (MNT):* the architecture contributes to maintaining the possible lowest network traffic on the video surveillance system [6].
- *Robustness against disconnections (RAD):* models the response of sensors and mobile devices to possible disconnections [13].
- *Abstraction of the data transmission medium (ADTM):* mobile devices can communicate the captured information from an area to their destination regardless of the network in which they are connected [13].
- *Group communication (CG):* considers the order of the generated messages and their correct reception between the device to device or device to user communication [13].
- *Temporary storage of event information (TSEI):* the information captured from an event is temporarily maintained in the vigilant device without impacting its performance.

C. DESIGN OF THE PROPOSED ARCHITECTURE

This paper introduces a layered software architecture with tiers to support the development of MDSV systems, see Fig. 1. Each layer provides specific services to upper layers, hiding details of the lower layer's implementation. In the architecture, the tiers complement the layers; services are modeled in vertical abstract layers, whereas tiers deal with

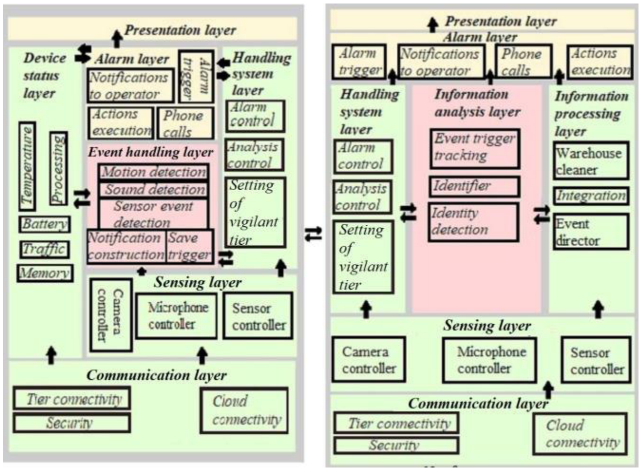


FIGURE 1. Design of the proposed layered software architecture: vigilant tier (left) and analyst tier (right).

the functionality of a specific layer into nodes, the vigilant tier is oriented to work with limited processing and storage capabilities of mobile devices, and the analyst tier is designed to work with devices with higher processing and storage capabilities such as servers. Therefore, workload and processing information is higher in the analyst tier than in the vigilant tier. Although there are layers in both tiers, the services and functionality are limited to hardware features of each tier. The architecture proposed in this work allows us identifying, classifying, grouping, and ordering the different services necessary to carry out aspects of protection, detection, and reply of an MDSV.

The description of the layers of Fig. 1 is as follows:

Presentation layer: displays a graphical user interface to access MVDS functions.

Alarm layer: provides access to functions related to notification, status, and actions of events; these functions are implemented in the following modules:

- *Alarm trigger:* activates sounds of the system to alert about the occurrence of an event of interest in the monitored area.
- *Notifications to the operator:* transmits messages directly to the system operator via email or SMS.
- *Phone calls:* makes phone calls to a number defined by the user in case the user has not deactivated the alarms during a specific period.
- *Execution of actions:* implements different actions after the occurrence of an event of interest, for example, doors and windows closing.

It is worth to mention that the services of the alarm layer depend on services from other layers.

Handling system layer: manages and executes different commands that are sent from the analyst tier to the vigilant tier by using the following modules:

- *Alarm control:* executes commands to activate or deactivate alarms in both tiers.

- *Analysis control:* manages the parameter setting of mobile devices to monitor an area and the level of information analysis performed by the analyst tier.
- *Setting of vigilant tier:* registers and distributes information about battery levels, bandwidth or CPU temperature of the devices in the vigilant tier.

Sensing layer: manages the life cycle of devices presented in both tiers by using the following modules:

- *Camera controller:* manages the life cycle of cameras.
- *Microphone controller:* manages the life cycle of microphones.
- *Sensor controller:* manages the life cycle of sensors.

Communication layer: provides of secure communication between tier and tier, tier and cloud as well as tier and user by means of the implementation of the following modules:

- *Security:* handles authentication to ensure that only authorized devices and users access to the video surveillance system.
- *Tier connectivity:* establishes the connection between both tiers for information sharing.
- *Cloud connectivity:* handles the connection between tiers and the cloud for information sharing.

Information processing layer: only belongs to the analyst tier and focuses on the management of information about events; this classifies, orders and groups the information for further analysis. The information processing layer has a module in charge of backups in the cloud to free up space for the analyst tier through the implementation of the following modules:

- *Warehouse cleaner:* deletes information stored locally after a complete transmission of this information to the cloud.
- *Integration:* gathers information of an event such as its occurrence time or its emergency level.
- *Event director:* directs the information to a specific layer like the alarm layer or the information management layer.

Analysis of information layer: only belongs to the analyst tier, these information processes are related to the search of events that interrupt security in a monitored area. The modules of this layer are the following:

- *Identifier:* detects people and animals in a video sequence of the monitored area.
- *Event trigger tracking:* follows an object in a video sequence to determine regular flow routes.
- *Identity detection:* detects faces in a video sequence and compares them with authorized persons. Also, it verifies for animals or objects to determine a possible intrusion.

Device status layer: this layer monitors hardware resources that could affect the proper functionality of vigilant devices such as battery level, use of memory or CPU workload. The modules of this layer are the following:

- *Temperature:* monitors that the CPU temperature must not exceed the defined limits.
- *Traffic:* monitors that the flow in the network must not exceed the defined limits.
- *Memory:* monitors that the consumption of RAM in the devices must not exceed the defined limits.
- *Processing:* monitors that the percentage of CPU workload must not exceed the defined limits.
- *Battery:* activates an alarm when the battery level is below a defined limit.

Event handling layer: detects and classifies events of the monitored area by implementing the following modules:

- *Motion detection:* analyzes captured images from the perimeter to detect movements.
- *Sound detection:* detects sounds in the monitored area
- *Sensor event detection:* analyzes information from sensors in search of variations that suggest the occurrence of an abnormal event.
- *Notification construction:* constructs messages with information about the level of emergency after an event has been detected.
- *Save trigger:* stores information after an event of interest has been detected in the monitored area.

IV. DEVELOPMENT OF THE LAYERED SOFTWARE ARCHITECTURE

This section describes the implementation of the layered software architecture described in section III-C. The first subsection presents the diagrams of components and the second one explains the implementation details of the proposed architecture.

A. DIAGRAMS OF COMPONENTS

Diagrams of components for each tier present the elements of the previous section; these diagrams are based on software patterns and recommended technologies according to the specialized literature. Fig. 2 shows the diagram of components for the vigilant tier; this is inspired by the Model-View-View Model (MVVM) pattern proposed by Google for Android

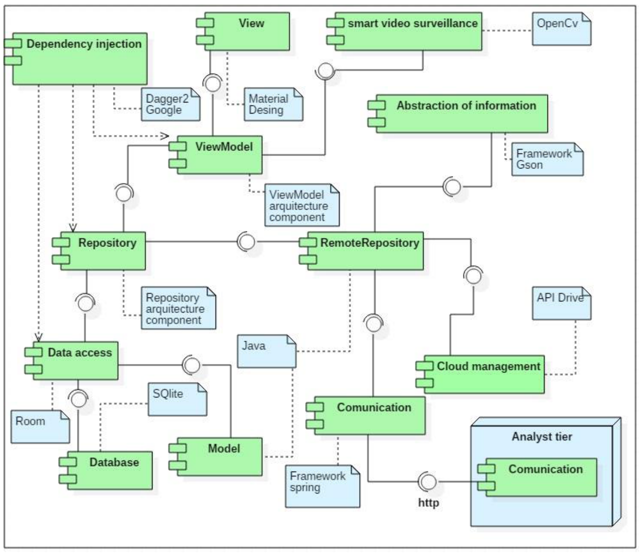


FIGURE 2. Diagram of components for the vigilant tier.

applications [14]. The advantages of this software pattern are the following:

- Decoupling of the model behavior with the view enabling the change of status individually that improves efficiency.
- Separation of responsibilities to facilitate changes or addition of new functionalities.
- Due to the granularity of the code, the execution of code tests is simplified.
- Promotion of reusability of code or functionalities.

The description of the components of the vigilant tier (see Fig. 2) is the following:

- *View:* handles the graphical user interface during the execution of the application.
- *ViewModel:* keeps information of a different life cycle assigned to the application to avoid loss of information when the setting of an application change, for example, when the screen is turned on.
- *RemoteRepository:* handles external storage of information for mobile devices, for example, the cloud.
- *Abstraction of information:* transforms the information into a format that enables its control and transmission from the vigilant tier (mobile devices of the video surveillance network) to the analyst tier (server).
- *Repository:* sends the information from the device that operates in the vigilant tier to local storage in the area.
- *Dependency Injection:* injects dependencies to minimize the impact of the use of devices and mitigates possible errors during the creation of different objects necessary to execute the system.
- *Data Access:* gives access to the database for mobile devices by using processes.
- *Database:* manages the database of a mobile device.
- *Model:* maps information stored in the database in such a way that process can handle and operate with it.
- *Cloud Management:* this is connected to the cloud and executes operations with the stored information.
- *Communication:* provides communication between the vigilant tier and the analyst tier. Besides, this support authentication to restrict devices that can communicate with the analyst tier.
- *Smart Video Surveillance:* provides the necessary mechanisms to process information gathered in the monitored area such as events of interest detection.

Fig. 2 proposes technologies to implement the components for the vigilant tier. A brief description of the proposed technologies is as follows:

- Dagger2 [15] is a static dependency injection framework for Java language. This operates at compile-time; thus, this minimizes the start-up time of applications and reduce possible errors during the creation of the required objects.
- Material Design [16] is a design standard for graphical interfaces that seeks to gather best practices.
- OpenCV for Android [16] is a library that implements computer vision techniques for mobile devices, this is launched under the terms of the BSD license.
- ViewModel [14] is a class that manages the own life cycle of data independently of a graphical interface.
- Java [17] is a multiplatform programming language that can be used to create different applications for personal computers, mobile devices or web.
- API drive [18] is an Android library that serves as a client; Google servers provide the services.
- Room [19] is a library that provides an abstraction layer over SQLite to allow quick access to the database.
- SQLite [20] is used through simple calls to subroutines and functions. This reduces the latency in the access to the database, because the calls to functions are more efficient than the communication between processes.
- Gson [21] is a library that converts JSON objects into Java objects during their transmission over a network or for further processing.

The replacement of any of the proposed technology must be compatible with the MVVM pattern.

The diagram of components for the analyst tier (see Fig. 3) was inspired by the Model View Controller pattern (MVC) [22].

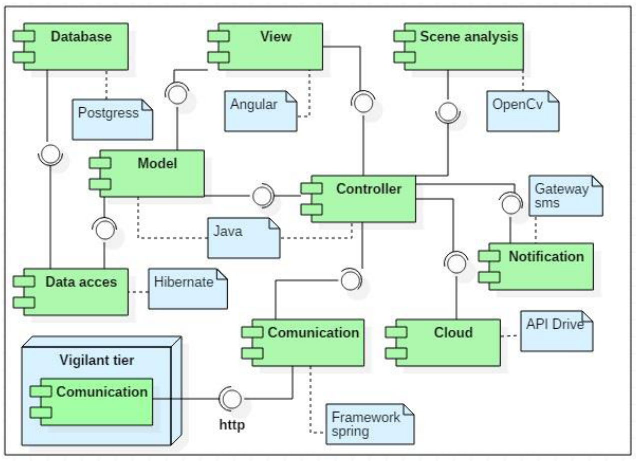


FIGURE 3. Diagram of components for the analyst tier (server side).

The description of components for the analyst tier is as follows:

- *View:* handles elements present in the graphical user interface.
- *Model:* maps the information that is stored in the database. This mapping is performed to manipulate the information from code before appearing at the graphical interface or database.
- *Controller:* controls the application logic and responds to requests from the graphical interface or some other process that perform requests.
- *Cloud:* handles the information stored in the cloud.

- *Notification:* provides mechanisms to generate notifications to the user about unusual events or the system status.
- *Scene analysis:* provides mechanisms that enable information analysis of the monitored area to identify an event of interest.
- *Data access:* provides the media to access the database of the analyst tier.
- *Communication:* identifies that the incoming messages to be from authenticated devices.
- *Database:* manages the analyst tier database.

Fig. 3 proposes technologies to implement the components of the analyst tier. A description of these components is as follows:

- PostgreSQL [23] is an open-source relational database system.
- Angular [24] is a web application development framework made in JavaScript by Google. Angular is flexible to work with different patterns.
- OpenCV [25] is a computer vision library launched under the terms of the BSD license. Therefore, its use is free for academic and commercial use.
- Java [17].
- Hibernate [26] is a tool that supports Object-Relational Mapping (ORM) for Java and.NET platforms.
- Spring MVC [27] is an open-source framework that works as a framework for the Java platform.
- Gateway SMS [28] is a server on the Internet that sends SMS messages through *rest services* from a computer to smartphones.

The replacement of any of the proposed technology for the analyst layer must be compatible with the MVC pattern.

B. DESCRIPTION OF THE ARCHITECTURE IMPLEMENTATION

This section describes the main classes for the implementation of the architecture presented in section III-C.

Communication layer: The main goal of this layer is to provide services only for known devices by the vigilant tier and analyst tier, in such a way that they can communicate with each other. The implementation of this layer is made in both tiers (the vigilant tier and the analyst tier) using the following classes and interfaces:

- Authenticator: This abstract interface includes the required methods to authenticate the devices that integrate the video-surveillance network.
- FileUploadProgressListener: This class monitors the file status that was sent to the cloud with the purpose of knowing when the load process was completed.
- FileDownloadProgressListener: This class verifies the status of the downloaded files from the cloud.
- CloudConexion: This class manages the stored files in the cloud.
- Encryptor: This abstract interface implements a function addressed to encrypt the information to be transmitted.
- Packer: This class inherits the capacity to split files into different packages in an object-oriented format such as JSON.
- StreamingHandler: This class has the required mechanisms for streaming management that will be generated by mobile devices.
- SyncronisedWaitingList: This abstract interface manages the order of the deliveries from the vigilant tier.
- ThreadSend: This class serializes the information to be transmitted.
- ThreadListenerOfServerComands: This class listens to the commands from the analyst tier.
- ThreadSendBatteryStatus: This notifies the current state of the battery from mobile devices to the analyst tier.
- ThreadSendFrames: This class sends the captured frames of the cameras in the monitored area.
- ThreadSendInfoOfSensor: This class sends the files generated by a special sensor of the video-surveillance system.
- ThreadSentVideo: This class makes the delivery of the video files generated in the monitored area.
- ThreadSentAudio: This class is like a secondary process, which sends the generated sound files from the guarded perimeter.
- HandlerRemoteConfiguration: This class sends and receives the setting of the devices that integrate the video surveillance system.

Sensing layer: this layer manages specifically the life cycle of different sensor types that enable the improvement of monitoring capacities of the perimeter in both tiers. Therefore, this layer is implemented in the vigilant tier through the following classes:

- SensorController: this class manages the life cycle of the sensor.
- SoundsController: manages the life cycle of the microphone and the sounds captured in the monitored area.
- CameraBridgeViewBase.CvCameraViewListener: controls the camera cycle as well as the captured frames.

The sensing layer of the analyst tier is implemented with the following classes and interfaces:

- Video: this class models the information from the database by using an object-oriented approach.
- AudioManagement: an abstract interface that provides methods to store, delete, and download sound files that were captured by the devices from the perimeter.
- SensorEventsController: this class implements the abstract interface ManagementOfSensorEventsController.
- VideoManagement: an abstract interface that provides methods to store, delete, and download video files that were captured by the monitoring devices.
- VideoController: the class that implements the abstract interface VideoManagement.
- VideoModel: this uses the Video class to model information from the database by using an object-oriented approach.

- Streaming: an abstract interface that provides methods to start and end streaming transmission generated by the devices of the video-surveillance system.
- StreamingController: the class that implements the Streaming abstract interface.
- RecordStreamingController: the class that implements the abstract interface RecordStreaming.

Alarm layer: manages the alarms that are generated by the system when an event of interest is detected in the monitored area. This layer belongs to the vigilant tier and is focused on generating alarms due to the presence of objects, humans and the critical status of the resources in the mobile device. This layer implements the following interfaces:

- DeviceAlarm: an abstract interface that has methods to generate different types of alarms regarding the status of the resources of mobile devices.
- AlertAlarm: an abstract interface that generates types of security alarms.

The implementation of the alarm layer in the analyst tier has a wide range of events from the generation of a sound to the delivery of messages and calls. The classes and interfaces of this layer are the following:

- Actions: abstract interface that enables to perform or to stop an action such as opening or closing doors, activating or deactivating sounds.
- ActionsController: the class that supports functionality to the methods of the Actions abstract interface.
- SendingOfMessages: an abstract interface to send messages.
- SendingOfMessagesController: the class that implements the SendingOfMessages abstract interface to enable functionality to its methods.
- CallsByTelephone: an abstract interface that has two methods to implement actions: the StartCall method that makes a telephone call and the StopCall method that stops a call.
- CallsByTelephoneController: the class that implements the abstract interface CallsByTelephone.

System manipulation layer: The goal of implementing the system manipulation layer is the parameter setting for the vigilant tier, which is composed of the following classes and interfaces:

- ControllerSetting: an abstract interface from the analyst tier for parameter setting.
- ControllerAlarm: an abstract interface that turns on and turns off the present alarms in the vigilant tier
- ControllerInformation: an abstract interface that manages the files stored in the devices.

The system manipulation layer of the analyst tier is implemented with the following classes and interfaces:

- SettingUpVigilantTier: class that matches the setting up information of the mobile devices by using an object-oriented approach.
- AbstractSettingUpVigilantTier: an abstract interface that defines the limits to capture information by each mobile-device related with the vigilant tier and with the required sensibility to make detections of events of interest considering the resources of each device and the possible intrusions in the monitored area. Besides, this interface manages the access credentials for each device and user that can communicate with the analyst tier
- SettingAnalystTierController: the class that implements the abstract interface AbstractSettingUpVigilantTier
- AnalysisSetting: an abstract interface that enables parameter setting of information registered by each device operated by the vigilant tier. On the one hand, this deals with the sensibility to identify objects, subjects or animals presented in the monitored area. On the other hand, this has methods for setting the level of use of resources in each mobile device (such as the battery, bandwidth, storage, processing) and the possible actions (such as sending messages, generation of telephone calls and sound activation)
- ManagementOfAlarmsController: the class that implements the abstract interface ManagementOfAlarms, this enables specific functionality to its inherit methods

Presentation layer: this layer is intended to consume the services provided by the other layers, organize the information generated due to that interaction and display it on the screen of the device on which each tier operates. The implementation of this layer in the vigilant tier focuses on providing the user with graphic access to each configuration option of the device on which this tier is executed through the following classes and XML files:

- SettingUp: the class and XML file for configuring the vigilant tier interface.
- Dashboard: the class and XML file that shows the different options for the actions at the vigilant tier application.
- IntelligentVideoSurvillance: the class and XML file to monitor an area to detect movement.
- Login: the class and XML file that manages the login and register in the analyst tier.

The implementation of the presentation layer in the analyst tier supports the following services: visualization of the status of devices, visualization of videos and streaming, storage in the cloud, parameter setting in the vigilant tier and analyst tier. In this case, the following elements were implemented:

- Index: JSP that shows information about the devices connected to the surveillance system and their respective actions. Furthermore, the index contains the options to address the screen setting and the system output.
- Login: Mechanism of the graphical interface that serves for authentication in the surveillance system.
- Setting: graphical interface for parameter setting that makes use of the AppController file and the different services provided by the remaining layers of the surveillance system.
- AppControler: this class works as a controller of the screen elements. Furthermore, this implements the requests of the SettingJpa class to consume the different services provided by the layers of the surveillance system.

Status of device layer: this layer is exclusive of the vigilant tier; due to its main function is monitoring the status of the bandwidth, memory, processing and temperature of the mobile device. This service is supported by the following classes and interfaces:

- BatteryManager: an abstract interface to get the current battery level of the device.
- NetworkMannager: an abstract interface that manages the transmitted information from the mobile device to the server.
- MemoryMannager: an abstract interface that monitors the use of memory in the device.
- WiFiMannager: abstract interface to detect if the devices are connected to a wireless network.
- TemperatureMannager: an abstract interface to determine the current temperature of the vigilant device.
- ProsecutionMannager: an abstract interface to obtain information about the processor consumption in the device.
- DeviceStatus: this class matches information from the database by using an object-oriented approach.
- WifiManager: the class to search for an available WiFi connection.

Event handling layer: the implementation of the event handling layer consists of a set of inter-related classes and interfaces to allow the vigilant tier monitors a specific area automatically and to register the events of interest through the following classes:

- AudioSyncronize: the class that implements the SynchronizedWaitingList and SynchronizedAudioByte to synchronize the captured information and to put in a waiting list the generated file in such a way that another process oversees its delivery.
- SyncronizedAudioByte: an abstract interface that provides a method to register the audio frames captured in the guarded perimeter.
- SynchronizedVideoFrame: an abstract interface that provides a method to register the captured audio frames
- SynchrinizeEventOfSensor: the class to synchronize the main process with the secondary process to register events from a sensor.
- GenerateAudio: the class that captures audio.
- GenerateInfoOfSensor: the class that works as a sub-process to capture a video event from a sensor.
- GenerateVideo: the class that captures a video FrameRecord: class that manages the video recording, this depends on FrameSychronize to record the video.
- FrameSynchronizer: the class that implements the SynchronizedWaitingList and SynchronizedVideoFrame to synchronize the captured information since the main thread and the secondary thread and to put the generated file in a waiting list while another process oversees its deliver.
- EventsOfSensroRecord: the class that controls the recording of events from the sensor.
- AudioRecord: an abstract interface that has a method to record audio, this depends on AudioSychronize class.
- FrameMotionDetection: an abstract interface that must be implemented for movement detection.
- SoundsDetection: an abstract interface to detect sounds using a modulator for noise sensibility.
- EventOfSensorDetection: an abstract interface that manages the detection of events out of the defined parameters.
- SynchronizedVideoFrame: an abstract interface to capture frames.
- PackerAudio: the class that extends Packer and implements the Encryptor abstract interface to package and encrypt sound information captured in the perimeter.
- PackerInfoOfSensor: the class that extends Packer and implements the Encryptor abstract interface to package and encrypt information from a sensor.
- PackerVideo: the class that extends Packer and implements the Encryptor abstract interface to package and encrypt information from a video.
- FrameSynchronizer: the class that implements the SynchronizedWaitingList and SynchronizedVideoFrame interfaces to synchronize the captured information from the main and secondary threads.
- ThreadHumanDetector: this class detects a human silhouette.

Information treatment layer: the goal of this layer is to make that the analyst tier manages information of an event presented in a monitored area. A set of classes oversees the storage size, integrating the captured information, and addressing the event information to the analyst tier or a user. The following classes and interfaces were designed to accomplish this goal:

- DBStore: an abstract interface in charge of cleaning the local store of the analyst tier when a critical level has been reached.
- BDStoreController: the class that implements the DBStore interface.
- EventAddresser: an abstract interface that enables addressing a notification directly to the user.
- EventAddresserController: the class that implements the EventAddresser an abstract interface by specifying its methods.
- Integration: an abstract interface with methods that transform information from mobile devices.
- IntegrationController: the class that implements the Integration abstract interface by specifying its methods

Information analysis layer: this layer processes the captured information from the monitored area by the different devices of the vigilant tier to detect the intrusion of not allowed persons, access to a zone outside of the allowed schedule or the presence of foreign objects. The following classes and interfaces were implemented to accomplish the goal of the information treatment layer:

- DetectionOfPerson: an abstract interface that provides a method to detect persons in a captured frame.

- DetectionOfPersonController: the class that implements the DetectionOfPerson interface.
- DetectionOfPersonIdentity: an abstract interface that has a method to detect the identity of a person that enters a scene.
- DetectionOfPersonIdentityController: the class that implements the DetectionOfPersonIdentity interface.
- DetectionOfAnimal: an abstract interface that provides a method to detect animals in a captured frame.
- DetectionOfAnimalController: the class that implements the DetectionOfAnimal interface.

C. DEVELOPMENT OF A MOBILE DISTRIBUTED SYSTEM OF THIRD-GENERATION VIDEO SURVEILLANCE BASED ON THE PROPOSED ARCHITECTURE

A prototype of a third-generation video surveillance system on mobile devices was created using the proposed architecture; the aim is to conduct a product-oriented evaluation. Section V-B presents the prototype tests. The prototype called SAVI (an acronym for the Spanish expression ''Sistema Automatizado de Videovigilancia Inteligente'') consists of a web application that acts as the analyst tier and a mobile application that acts as the vigilant tier. SAVI detects and records movement in a monitored area automatically. The system identifies if a person makes a movement and sends a notification via SMS to a specific phone number. In addition, SAVI includes the following functionalities: store and manage captured videos locally and in the cloud; configure an operation schedule; visualize the battery level of the devices that monitors an area; and, store the names and phone numbers of the people who will receive notifications. Fig. 4 shows the general schema of SAVI. The description of nodes that form the deployment diagram is the following:

- *Web Client:* contains the components to manage web pages. This node request streaming initialization to the analyst tier and communicates with the server node to ask for consuming the generated streaming.
- *Analyst Tier:* focuses on listening and responding to REST requests performed by the web client and the vigilant tier. Furthermore, this can manage stored information locally and in the cloud.
- *Cloud:* works as external storage of the generated information from the monitored area.
- *Vigilant Tier:* This operates as a vigilant in the area of interest, detects events in search of a risk situation that affects security. Also, this automatically generates videos of interesting events and transmits the captured scenes.
- *Streaming Server:* serves as a link channel between the streaming generated by the vigilant tier and the streaming player of the web client.

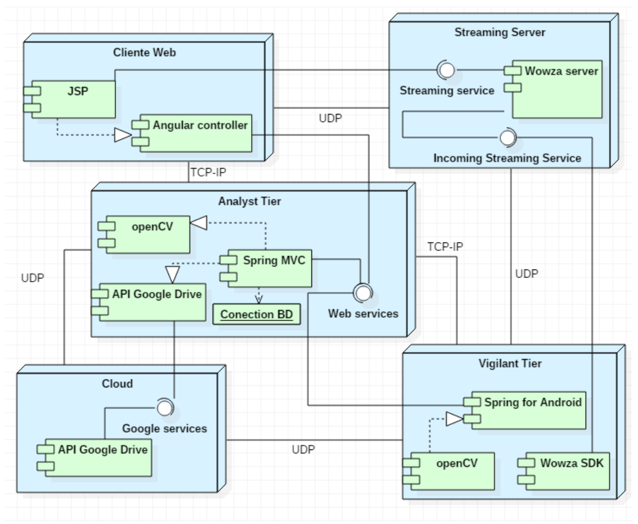


FIGURE 4. Deployment diagram of the SAVI prototype.

On the vigilant tier side, the SAVI prototype offers the following services:

Authentication for mobile devices: provides authentication in such a way that only authorized mobile devices to belong to the video surveillance network.

Vigilant tier device configuration: this service defines the recording time and the names of videos files for devices. Moreover, this configures the notifications via phone calls when the system detects a foreign person in the monitored area.

Monitoring area: this service aims to capture the information of the monitored area to determine events of interest such as intruders or risk situations. Fig. 5 shows an example of a monitored area. This service support functions such as motion detection, video recording, delivery of captured videos to the analyst tier, detection of people, execution of instructions in the analyst tier, delivery of captured frames to the analyst tier or notify about the battery level.

FIGURE 5. Example of a monitored area.

On the other hand, the implemented functionalities for the analyst tier are:

User authentication: This service authenticates users that will interact with the video surveillance system.

Streaming generation: This functionality was implemented through a REST service created in the analyst tier (backend) that is called by the frontend through the clicking event on the "generate streaming" button. As a result, the backend communicates with the designated device for streaming initialization.

Streaming visualization: This functionality works with a server that receives the video generated by a mobile device as follows (see Fig. 6):

- *Step 1:* The mobile device begins to generate the streaming of the monitored area.
- *Step 2:* Streaming is transmitted from the mobile device to the streaming server.
- *Step 3:* Subsequently, the transmitted video is transformed to MP4 format so that different clients can reproduce this.
- *Step 4:* Finally, the analyst tier consumes the streaming that is available on the streaming server and starts its reproduction.

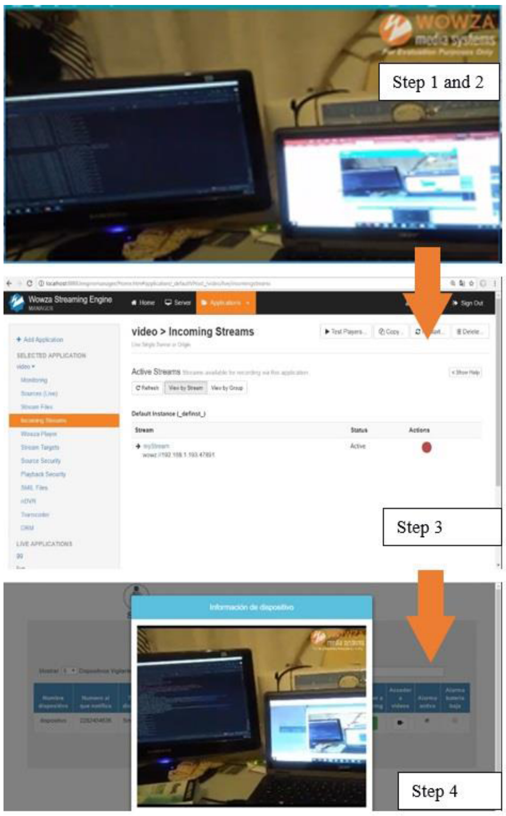
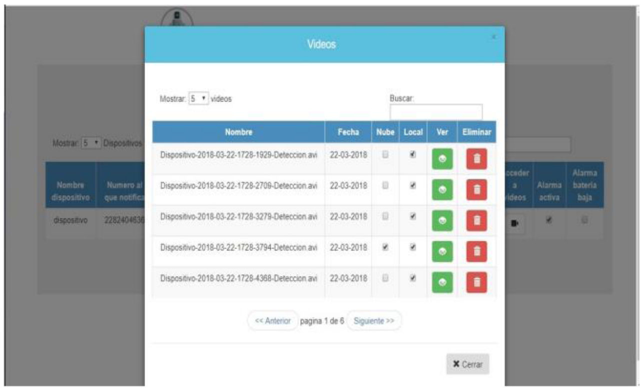


FIGURE 6. Flow to visualize the captured streaming of the monitored area.

Information on cloud storage: this functionality was implemented with three REST services. The first one uploads a file to the cloud. The second one deletes specific videos. Finally, the third REST service downloads some videos. All these services are performed when the user check and uncheck the checkboxes, illustrated in Fig. 7.

Device management (vigilant tiers): This functionality consists of editing or eliminating mobile devices from the video surveillance network by using the edit or delete buttons.



Videos

Mostrar: 5 videos

Buscar:

Nombre	Fecha	Nube	Local	Ver	Eliminar
Dispositivo-2018-03-22-1728-1929-Deteccion.avi	22-03-2018	☐	✓		
Dispositivo-2018-03-22-1728-2709-Deteccion.avi	22-03-2018	☐	✓		
Dispositivo-2018-03-22-1728-3279-Deteccion.avi	22-03-2018	☐	✓		
Dispositivo-2018-03-22-1728-3794-Deteccion.avi	22-03-2018	✓	✓		
Dispositivo-2018-03-22-1728-4368-Deteccion.avi	22-03-2018	☐	✓		

<< Anterior pagina 1 de 6 Siguiente >>

✖ Cerrar

FIGURE 7. Handling files in the cloud from SAVI in the analyst tier.

Configuration of operation parameters: Manages the parameter settings to each device connected to the video surveillance system in order that all the system elements have the same configuration.

We want to highlight that the functionality of the SAVI prototype corresponds to the protection, detection, and reply services of a third-generation video surveillance system on mobile devices [5].

V. EVALUATION OF THE PROPOSED ARCHITECTURE

The evaluation of the layered software architecture is based on two approaches: the first one makes a qualitative comparison between the architectures of related works and the proposed architecture. This comparison is described in section V-A. The second approach evaluates the SAVI prototype, that is, the resulting product from the proposed architecture. The results of the latter approach are presented in section 5-B.

A. FIRST EVALUATION APPROACH: QUALITATIVE COMPARISON

This section presents a qualitative comparison of the software architectures presented in the related works and the proposed architecture (see Table 1 and Table 2). The comparison is based on the identification of desirable requirements for third-generation video surveillance systems on mobile devices described in the specialized literature reported in [2], [3], [5], [8] and [10]. The descriptions of these requirements can be found in sections III-A and III - B.

TABLE 1. Features of architectures for MVDS systems part 1.

Work	Sen	IP	RTM	ECA	IIHS	SC	DAS	VDSN	S	CA
[3]	Y	Y	Y	Y	Y	N	Y	N	Y	N
[10]	Y	N	Y	N	Y	Y	N	N	Y	N
[6]	Y	N	Y	N	N	Y	N	N	Y	Y
[7]	N	Y	Y	N	N	N	N	N	Y	N
[8]	Y	N	Y	N	N	Y	Y	N	Y	N
[9]	Y	N	Y	Y	N	Y	N	Y	Y	Y
[11]	N	Y	Y	N	N	N	Y	N	N	Y
[12]	Y	Y	Y	N	Y	N	N	N	Y	Y
Our proposed architecture	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

TABLE 2. Features of architectures for MVDS systems part 2.

Work	L	SD	VDH	MTN	RAD	ADTM	GC	TSEI
[3]	N	N	N	N	N	N	N	Y
[10]	Y	N	Y	N	Y	Y	N	Y
[6]	Y	N	N	Y	Y	Y	N	Y
[7]	Y	N	N	N	Y	Y	N	N
[8]	Y	N	N	Y	Y	N	N	Y
[9]	Y	N	Y	N	Y	N	N	N
[11]	N	N	N	N	N	N	Y	N
[12]	Y	Y	N	N	Y	Y	Y	N
Our proposed architecture	Y	Y	Y	Y	Y	Y	Y	Y

Table 1 summarizes the comparative analysis of each related work with respect to the requirements or essential features that third-generation video surveillance systems must fulfill. Note that the architectures described in [3], [6]–[11] and [12] do not consider requirements such as surveillance calibration (SC), vigilant device status notification (VDSN), information processing (IP), environment change adaptation (ECA), integration of information from heterogeneous sources (IIHS), detection of abnormal situations (DAS) and control of alarms (CA); the latter requirement is directly related to the reply service.

Although the architectures reported in [3], [6]–[11] and [12] fulfill several features or requirements, they do not consider security device (SD), vigilant device health (VDH), minimization of traffic network traffic (MTN), abstraction of the data transmission medium (ADTM) neither group communication.

Note that the proposed architecture accomplishes the requirements that are related to protection, detection, reply services of MVDS systems and considers the features of mobile distributed systems.

B. SECOND APPROACH: PRODUCT-ORIENTED EVALUATION

This section describes the evaluation based on the quality of the SAVI prototype, the resulting product of the proposed architecture. This type of evaluation is also known in the literature as a prototype oriented assessment [29]. In this product-oriented evaluation, unit tests on services for each application were performed, as well as integration tests with both applications of the system. Furthermore, performance tests were carried out to measure aspects of battery consumption, RAM and CPU workload in both tiers. Unit, integration, and performance tests were performed in a room of 4.92 meters wide by 10.13 meters long with lighting provided by two 100-watt spotlights. In all the case tests, the vigilant tier was executed in 3 mobile devices. Tables 3, 4 and 5 describe these devices, respectively.

The analyst tier was executed on a laptop with the characteristics presented in Table 6.

Unit, integration, and performance tests are described in detail below.

TABLE 3. Description of the alcatel device.

Device:	*Smartphone Alcatel 5054s POP model with a year of use*
Characteristics:	Operative System: Android 5.1.1. Device: Smartphone Alcatel 5054S model. Processor: ARM Cortex-A7. Processor speed: 1100 MHz Device processor cores: 4. Device graphics processor: Qualcomm Adreno 304. Graphics processor speed: 400 MHz. Camera: rear-facing camera. Initial battery load: 100%. Device RAM: 1 GB.

TABLE 4. Description of the samsung device.

Device:	*Smartphone Samsung model Galaxy S5 released in 2014*
Characteristics:	*Operative System: Android 5.1.1.* *Device: Smartphone Samsung S5.* *Processor: Snapdragon 801* *Processor speed: 2500 MHz* *Device processor cores: 4.* *Device graphics processor: Qualcomm Adreno 501.* *Graphics processor speed: 508 MHz.* *Camera: rear-facing camera.* *Initial battery load: 100%.* *Device RAM: 2 GB.*

TABLE 5. Description of the lenovo device.

Device:	*Smartphone Lenovo K5 model with a year of use*
Characteristics:	*Operative System: Android 5.1.1.* *Device: Smartphone Alcatel 5054S model* *Processor: Snapdragon 415.* *Processor speed: 1500 MHz.* *Device processor cores: 4.* *Device graphics processor: ARM Mali-T860 MP2.* *Graphics processor speed: 667 MHz.* *Camera: rear-facing camera.* *Initial battery load: 100%.* *Device RAM: 2 GB.*

TABLE 6. Description of the computer used as the analyst tier.

Device:	*Laptop*
Characteristics:	Processor: Intel® Core™ I3-4030U CPU@ 1.90 GHz. Operative System: Windows 10, 64 bits RAM: 8GB Hard Disk: 500 GB

1) UNIT TESTS

The goal of unit tests is to ensure the correct operation of each service of the SAVI prototype. The vigilant tier tests were executed on the Alcatel device, described in Table 3. Likewise, the laptop described in Table 4 was used for the analyst tier. The execution of all tests followed the standard format compose of the next elements:

- **Module:** the name of the tested module.
- **Test:** an identifier for the test.
- **Purpose:** indicates the expected functionality of the tested module.
- **Requirements:** Conditions that must be ready to execute the module.

- **Entry:** Input data for the test.
- **Result:** Indicates whether the test archive the expected purpose.

A set of errors were found and fixed during the tests in the vigilant tier. Table 7 shows the type and number of errors found per module. A total of five iterations were performed for each test.

TABLE 7. Summary of errors found in the vigilant tier.

Module	*Found errors*	*Solved errors*	*Pending errors*	*Iterations*
Authenticate	2	2	0	5
Access to cameras	1	1	0	5
Turn off alarms	3	3	0	5
Set schedules	2	2	0	5
Analyze information	1	1	0	5
Store information locally	1	1	0	5
Store information in the cloud	0	0	0	5

Table 8 shows the type and number of errors found per module during the tests to the analyst tier. Also, a total of five iterations were performed.

TABLE 8. Summary of errors found in the analyst tier.

Module	*Found errors*	*Solved errors*	*Pending errors*	*Iterations*
Authenticate	1	1	0	5
Access to cameras	0	0	0	5
Turn off alarms	2	2	0	5
Set schedules	4	4	0	5
Analyze information	2	2	0	5
Store information locally	1	1	0	5
Store information in the cloud	0	0	0	5

2) INTEGRATION TESTS

Integration tests were performed to ensure the correct operation of services during the interaction between the vigilant tier and the analyst tier. Integration tests follow the format presented for unit tests; they were recorded and then fixed the found errors. The results of the integration tests are summarized in Table 9.

TABLE 9. Summary of integration tests.

Module	*Found errors*	*Solved errors*	*Pending errors*	*Iterations*
Authenticate	0	0	0	5
Access to cameras	0	0	0	5
Turn off alarms	2	2	0	5
Set schedules	1	1	0	5
Information analysis	1	1	0	5
Store information locally	1	1	0	5
Store information in the cloud	0	0	0	5

As a result of unit and integration tests, the operation of all services in the proposed architecture was corrected.

3) PERFORMANCE TESTS

The features of the scenarios considered during the performance tests were the following:

- 2, 4 and 6 people (P) simultaneously in the monitored area by the vigilant tier.
- 0, 2 and 4 background applications (Apps) such as phone, SMS, clock, calendar, schedule or Gmail running in the vigilant tier
- 2, 4 and 6 meters (M) away from the vigilant tier with respect to the monitored area

The performance tests were divided into two phases. In the first phase, three vigilant tiers were running on the mobile devices described in Tables 3, 4 and 5. The purpose of this test was to evaluate the CPU workload, RAM and battery, when performing tasks such as area monitoring, motion detection, video recording, delivery of captured videos to the analyst tier, detection of people, delivery of frames, and notification of battery levels or analysis of information. In the second phase, the server performance was evaluated (analyst tier) in terms of CPU workload and RAM usage while this was making notifications to the mobile devices that integrate the video surveillance network. Each test lasted about 30 minutes. In these tests, we consider unreliable and asynchronous communication channels. The message transmission delay considered was between 0 and 450 ms and the message lost rate considered at communication channels was between 0-5 percent, which reflects a real network load. The obtained results are shown below.

a: PERFORMANCE TEST FOR THE VIGILANT TIER

Fig. 8. shows the results obtained during the execution of the first phase of the performance tests related with RAM consumption. The observations for each device are the following:

- The Alcatel device maintained RAM consumption less than 128 MB even in the scenario that involved 4 parallel running applications with the vigilant tier and simultaneous detection of 6 people in the monitored area
- The Samsung device consumed less than 65 MB despite the distance and the number of people detected

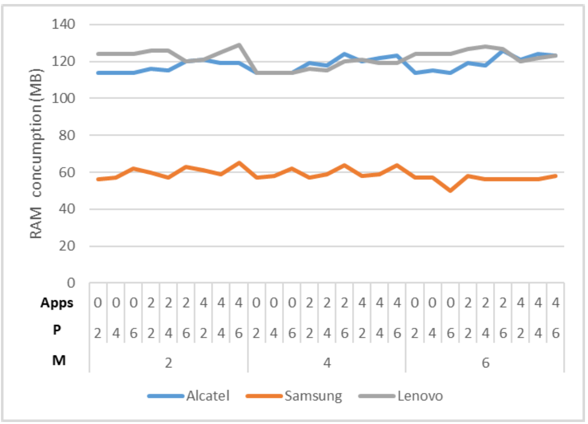


FIGURE 8. RAM consumption of vigilant tiers.

- The Lenovo mobile device has a consumption of less than 130 MB despite increasing the size of the monitored area, the distance of the mobile device and the number of people detected

The results of the CPU workload by the vigilant tier on mobile devices are illustrated in Fig. 9. The observations for each device are the following:

- The Alcatel device had a higher CPU consumption, as there were more people in the monitored area. However, these peaks remain below 72% of its CPU capacity.
- The Samsung device consumed less than 76% despite changing the number of people and the distances between the monitored area and this device.
- The Lenovo device maintained CPU consumption below 88% in its maximum peaks by varying the number of people in the monitored perimeter.

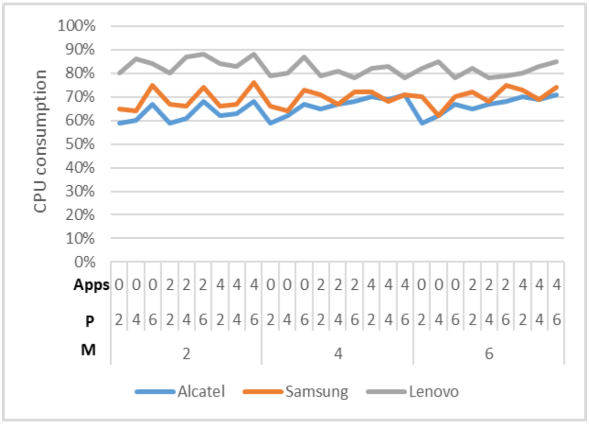


FIGURE 9. CPU consumption of mobile devices during the first phase of the performance tests.

In the second testing phase, the battery consumption during the execution of the vigilant tier on the three devices was below 15% as is showed in Fig. 10. Table 10 presents the average consumption of performance. The results presented in this section indicate that the performance of RAM, CPU workload and battery consumption were acceptable even in complex scenarios [29].

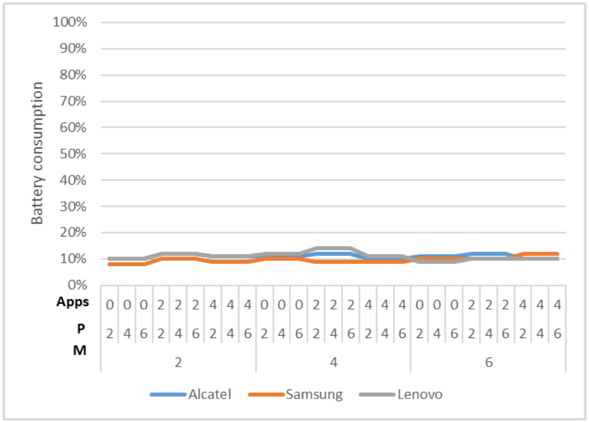


FIGURE 10. Battery consumption of mobile devices during the first phase of the performance tests.

TABLE 10. Average consumption of resources during the first testing phase.

Device	RAM	CPU	Battery
Alcatel	11.55 %	65 %	11 %
Samsung	02.90 %	70 %	10 %
Lenovo	05.95 %	82%	11 %

b: PERFORMANCE TEST FOR THE ANALYST TIER

The results of the performance test for the analyst tier was obtained considering a simultaneous interaction with the three vigilant tiers running on the previously described devices. The test lasted 45 minutes approximately; there were six people in the monitored area. The objective of this test was to measure and evaluate the consumption of resources that the analyst tier had when this was running on a server mounted on the laptop described in Table 6, the interaction was done with three vigilant tiers. Fig. 11 shows the results of CPU workload and RAM consumption; the results were below 85% and 65%, respectively. It is worth to mention that during the execution of test basic Windows 10 processes were coexisting. The video reception from the monitored area was also part of the test, the Google Chrome browser 66.0.3359.181 version was used (Official build, 64 bits); speakers were also used.

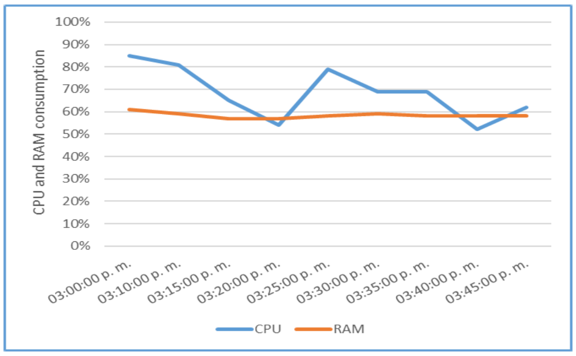


FIGURE 11. CPU and RAM use during the performance test for the analyst tier.

VI. CONCLUSIONS AND FUTURE WORK

This work introduces a layered software architecture that models protection, detection, and reply services in third-generation video surveillance on mobile devices. Two tiers integrate the architecture: vigilant tier and analyst tier. The vigilant tier was developed to run on devices with limited capabilities regarding processing and storage such as low and mid-range mobile devices. On the other hand, the analyst tier was designed considering devices with higher processing and storage capacities (such as personal computers or servers). Therefore, workload and processing information is higher in the analyst tier than in the vigilant tier. Each tier is composed of a set of layers where the lower layers provide services to the upper layers. The vigilant tier is integrated by the layers for communication, sensing, the status of the device, event handling, system manipulation, alerts and presentation, while the analyst tier includes layers for communication, sensing, processing of information, analysis of events, system manipulation, alerts and presentation. Besides, we have presented a prototype that implements the proposed architecture; the results of unit and integration tests show that the proposed architecture provides users and developers of technical guidelines to implement protection, detection and reply services. The results of the performance tests in the vigilant and analyst tiers running on three different mobile devices and a laptop maintained an acceptable consumption of resources even for complex scenarios. In comparison with related works, the proposed architecture includes the features of mobile distributed systems; this defines basic technical guidelines and a robust framework for developers in order to satisfy quality requirements of third-generation video surveillance systems. The implementation of computer vision methods and algorithms to identify people and action recognition in the monitored areas are planned as future work.

REFERENCES

[1] R.-C. Mihailescu, P. Davidsson, U. Eklund, and J. A. Persson, "A survey and taxonomy on intelligent surveillance from a system perspective," *Knowl. Eng. Rev.*, vol. 33, no. E4, pp. 1–25, 2018.
[2] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: A review," *IET J.*, vol. 152, no. 2, pp. 192–204, 2005.
[3] M. D. Ruiz-Lozano, "Un modelo para el desarrollo de sistemas de detección de situaciones de riesgo capaces de integrar información de fuentes heterogéneas. Aplicaciones," Ph.D. dissertation, Dept. Ciencias Computación Inteligencia Artif., Escuela Técnica Superior Ingenierías, Univ. de Granada, Granada, España, 2010, p. 328.
[4] P. Kumar, A. Mittal, and P. Kumar, "Study of robust and intelligent surveillance in visible and multi-modal framework," *Informatica*, vol. 32, no. 1, pp. 63–77, 2008.
[5] M. A. Maloof, *Machine Learning and Data Mining for Computer Security: Methods and Applications*. London, U.K.: Springer, 2006.
[6] D. J. F. Gutiérrez *et al.*, "Sistema de video vigilancia semántico basado en movimiento, aplicación a la seguridad y control de tráfico," Ph.D. dissertation, Dept. de Teoría de la Señal y Comunicaciones e Ingeniería Telemática, Univ. de Valladolid, Valladolid, España, 2013, p. 93.
[7] H. D. Park, O.-G. Min, and Y.-J. Lee, "Scalable architecture for an automated surveillance system using edge computing," *J. Supercomput.*, vol. 73, no. 3, pp. 926–939, 2017.
[8] I. H. Velasco, E. L. Domínguez and J. de la Calleja, "Propuesta de una arquitectura en capas para el desarrollo de sistemas distribuidos de videovigilancia," *Adv. Inf. Technol.*, vol. 57, no. 1, pp. 107–114, 2012.
[9] R. Vezzani and R. Cucchiara, "Event driven software architecture for multi-camera and distributed surveillance research systems," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, Jun. 2010, pp. 1–8.
[10] E. Cermeño, A. Pérez, and J. A. Sigüenza, "Intelligent video surveillance beyond robust background modeling," *Expert Syst. Appl.*, vol. 91, pp. 138–149, Jan. 2017.
[11] A. Alshammari and D. B. Rawat, "Intelligent multi-camera video surveillance system for smart city applications," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 317–323.
[12] D. Nagothu, R. Xu, S. Y. Nikouei, and Y. Chen, "A microservice-enabled architecture for smart surveillance using blockchain technology," in *Proc. IEEE Int. Smart Cities Conf. (ISC2)*, Kansas City, MO, USA, Sep. 2018, pp. 1–4.
[13] M. A. M. Acosta, E. L. Dominguez, G. G. Castro, S. E. P. Hernandez, and M. A. M. Nieto, "Two-level software architecture for context-aware mobile distributed systems," *IEEE Latin Amer. Trans.*, vol. 13, no. 4, pp. 1205–1210, Apr. 2015.
[14] Google. *ViewModel Overview|Android Developers*. Accessed: Jun. 5, 2018. [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel
[15] Google. (2018). *Dagger*. Accessed: Jun. 5, 2018. [Online]. Available: https://google.github.io/dagger/
[16] Google. *Design—Material Design*. Accessed: Jun. 5, 2018. [Online]. Available: https://material.io/design/
[17] ORACLE. *Java SE|Oracle Technology Network|Oracle*. Accessed: Jun. 5, 2018. http://www.oracle.com/technetwork/java/javase/Poverview/index.html
[18] Google. (2018). *Manage APIs in the Cloud Platform Console—Ayuda de Cloud Platform Console*. Accessed: Jun. 5, 2018. [Online]. Available: https://support.google.com/cloud/answer/6326510
[19] Google. *Room Persistence Library|Android Developers*. Accessed: Jun. 5, 2018. https://developer.android.com/topic/libraries/architecture/room
[20] SQlite. (2000). *About SQLite*. Accessed: Jun. 5, 2018. [Online]. Available: https://www.sqlite.org/about.html
[21] Google. (2018). *GitHub—Google/Gson: A Java Serialization/Deserialization Library to Convert Java Objects Into JSON and Back*. Accessed: Jun. 5, 2018. [Online]. Available: https://github.com/google/gson
[22] Y. F. Romero and Y. D. González, "Patrón modelo-vista-controlador," *Rev. Telem@tica*, vol. 11, no. 1, pp. 47–57, 2012.
[23] The PostgreSQL Global Development Group. *PostgreSQL: Documentation*. Accessed: Jun. 5, 2018. [Online]. Available: https://www.postgresql.org/docs/
[24] Google. (2018). *AngularJS—Superheroic JavaScript MVW Framework*. Accessed: Jun. 5, 2018. [Online]. Available: https://angularjs.org/
[25] OpenCV Team. *Android—OpenCV Library*. Accessed: Jun. 5, 2018. [Online]. Available: https://opencv.org/platforms/android/
[26] H. Reb. *Your Relational Data. Objectively—Hibernate ORM*. Accessed: Jun. 5, 2018. [Online]. Available: http://hibernate.org/orm/
[27] Pivotal. (2018). *Web on Servlet Stack*. Accessed: Jun. 5, 2018. [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html
[28] Infobip. *Gateway SMS*. Accessed: Jun. 5, 2018. [Online]. Available: https://www.infobip.com/es/glosario/gateway-sms
[29] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. New York, NY, USA: Reading, MA, USA: Addison-Wesley, 2004.

YAIR VIVEROS MARTÍNEZ is currently pursuing the master's degree in applied computing with the National Laboratory on Advanced Informatics (LANIA). His current research interests are focused on software engineering and software development.

EDUARDO LÓPEZ DOMÍNGUEZ received the Ph.D. degree from the National Institute of Astrophysics, Optics and Electronics (INAOE), Mexico, in 2010. He is currently a Researcher with the Department of Computer Science, National Laboratory of Applied Informatics (LANIA), Veracruz, Mexico. Since 2004, he has been researching in the field of mobile distributed systems, partial order algorithms, and multimedia synchronization.

YESENIA HERNÁNDEZ VELÁZQUEZ received the M.Sc. degree from the Benemérita Universidad Autónoma de Puebla (BUAP), Mexico, in 2011. She is currently a Researcher with the Department of Computer Science, Laboratorio Nacional de Informática Avanzada (LANIA), Veracruz, Mexico. Since 2009, she has been researching in the field of mobile learning systems.

SAÚL DOMÍNGUEZ ISIDRO received the Ph.D. degree in the artificial intelligence program from the University of Veracruz, Mexico, in 2017. He is currently a Researcher with the Department of Computer Science, Laboratorio Nacional de Informática Avanzada (LANIA), Veracruz, Mexico. Since 2010, he has been researching in the field of soft-computing.

MARÍA AUXILIO MEDINA NIETO is currently an Associate Professor of computer science with the Universidad Politécnica de Puebla (UPPuebla). Her current research topics are knowledge representation based on ontologies, semantic web, information and communications technologies (TICs), and social network analysis. She has the candidate status of the National Researchers System (SNI).

JORGE DE LA CALLEJA received the M.Sc. and Ph.D. degrees from the National Institute of Astrophysics, Optics and Electronics (INAOE), Mexico. He is currently a full-time Professor with the Computer Science Department, Universidad Politécnica de Puebla (UPPuebla), Mexico, since March 2008. His research interests include machine learning, computer vision and data mining, with applications in medicine, education, and complex systems.