TECHNICAL SCIENCES AND TECHNOLOGIES
UDC 004.4
DOI: 10.25140/2411-5363-2017-4(10)-101-109

Ruslan Zarovsky, Andrii Radchenko

ARCHITECTURE OF SOFTWARE FOR VIDEO SURVEILLANCE SYSTEMS WITH DIFFERENT TYPES OF CAMERAS

Urgency of the research. *Usually, the software that performs NVR functions on a normal PC is suitable only for certain types of cameras. Accordingly, the use of cameras from many manufacturers in the video surveillance system leads to use a large number of different software. This creates inconvenience to the user because for performing necessary functions (viewing, recording video, etc.) on different cameras it is necessary to run various software. Therefore, there is a need of creation software that would support different types of cameras.*

Target setting. *Non-optimal implementation of software architecture that supports devices of different manufacturers can lead to difficulty in understanding of source code, non-optimal use of network resources and so on. Thus, there is a problem of proper construction of the software architecture in order to eliminate these problems.*

Actual scientific researches and issues analysis. *The analysis of publications allows revealing the general tendencies of building video surveillance architectures, among which decreasing networking and storage costs. Reduction of network costs implies the use of special measures to minimize the total size of transmitted media data. This can be achieved through a video surveillance system architecture that eliminates the retransmission of the same information and in general minimizes the exchange of information in the IP network of video surveillance. So, in publications describes the architecture of a video surveillance system, but not software architecture for such systems.*

Uninvestigated parts of general matters defining. *Now there is no open software architecture that support the IP cameras from different manufactures.*

The research objective. *The objective of this paper is to describe the architecture of software that supports IP cameras and NVRs from leading Chinese manufacturers, such as Hikvision, Dahua, UniView, Aevision, as well as devices that operate on universal protocol Onvif.*

The statement of basic materials. *The architecture that works with different types of cameras should be designed accordingly. First of all it is necessary to build architecture at the level of logical components and then at the level of functional components. Software architecture at the level of logical components consists of Screen, VideoPlayer, VideoSchedule, CameraView, ModulesContainer and VideoSender components. Software architecture at the level of functional components consists of Screen, VideoPlayer, VideoSchedule, CameraView, ModulesContainer, VideoSender, FrameSourcer, FrameSaviour and Logginner components.*

Conclusions. *The proposed architecture allows using many types of cameras in single software, which is much more convenient than using many programs for many types of cameras. It minimize network load by using only one video stream from one channel, allows to connect all the channels of devices of supported manufacturers and to use all necessary functions for video surveillance systems of supported IP cameras. It does not lead to the redundancy of the source code or its great complexity. Thus, software is not difficult to maintain and add new functionality.*

Key words: *software; software architecture; video surveillance; IP camera.*
Fig.: 3. Bibl.: 14.

Introduction. In the modern world, video surveillance systems are becoming widespread, the role and importance of which is difficult to overestimate. Such systems consist of a set of hardware and software that include:

– IP surveillance cameras [1];

– video display devices (monitors, video walls and etc.);

– video recorders (DVR, NVR [2]) and/or intelligent software that performs a similar function as NVR, but on a normal PC.

Usually, the software that performs NVR functions on a normal PC is suitable only for certain types of cameras. Accordingly, the use of cameras from many manufacturers in the video surveillance system leads to use a large number of different software. This creates inconvenience to the user because for performing necessary functions (viewing, recording video, etc.) on different cameras it is necessary to run various software. Therefore, there is a need of creation software that would support different types of cameras.

Non-optimal implementation of architecture of software that supports devices of different manufacturers can lead to:

– difficulty in understanding of source code, that lead to a high cost of its modification and addition of new functions;

– redundancy of source code;

– non-optimal use of network resources, that is expressed in the receipt of multiple video streams from a single video channel;

– non-optimal use of PC resources, that is expressed in the multiple decoding of one video stream or multiple decoding of video streams from one channel;

– limitation of video channels count that can be connected from NVRs by software.

Thus, there is a problem of proper construction of the software architecture in order to eliminate the above-mentioned problems.

Analysis of recent research and publications. There are too few research and publications on this matter that are in open access since the development of such software is carried out by a large corporate sector (which regard information about it's software as a trade secret) or small companies (which do not spend time on the development of documentation and its publication).

In [3] describes the architectures of video surveillance systems, compares them, points out advantages and disadvantages. Also mentioned the software functions that reduce the load on computers and improve the functioning of video surveillance systems: cluster organization of servers, restart in case of failures, support of various video streams, using hardware decoders, support multitasking, optimizing video streams by using a computer as a gateway.

In [4] the reasons of wasting network resources in video surveillance systems are named, as well as software functions that reduce the network load and reduce the total cost of video surveillance systems. Among such functions it is possible to select the following: the use of multicast video streaming instead of unicast, the receipt of video streams of different resolutions, the automatic determination of the quality of the desired video stream and the caching of video stream in the event of its transfer from the server.

In [5] five different system architectures of video surveillance are described, its advantages and shortcomings are mentioned.

In [6] describes the design and optimization of a wireless video surveillance system, specifies the criteria for selecting cameras, hardware and software for such a system, describes the architecture of the wireless surveillance system, as an example shows the physical location of the cameras, describes the software that was implemented for such a system, describes experiments to determine the parameters of cameras for optimizing the load on the network, hardware and software.

In [7] describes designing, development, integration and delivery of Intel technology-based digital security and surveillance systems. Technical information and design support including recommended system components, technical requirements and specifications is mentioned.

The analysis of publications allows revealing the general tendencies, among which: decreasing networking and storage costs of surveillance system. Reduction of network costs implies the use of special measures to minimize the total size of transmitted media data. This can be achieved through a video surveillance system architecture that eliminates the retransmission of the same information and in general minimizes the exchange of information in the IP network of video surveillance.

So, in publications describes the architecture of a video surveillance system, but not software architecture for such systems.

The goal of the article. The objective of this paper is to describe the architecture of software that supports IP cameras and NVRs from leading Chinese manufacturers, such as Hikvision [8], Dahua [9], UniView [10], Aevision [11], as well as devices that operate on universal protocol Onvif [12].

Basic concepts for the building of software architecture. The software for video surveillance systems must provide the following functions:

– simultaneous viewing many video streams from cameras or NVRs in real time;

– obtaining data of various events from devices (motion detection, alarm, etc);

– displaying events data;

– recording a video stream on a hard drive on the user's request;

– recording a video stream on a schedule and events like motion detection and alarm;

– saving images from a video stream;

– playback saved video streams;

– transferring a video stream and event's data from connected devices to some client's software.

Software architecture must also support intelligent modules, such as license plate recognition module, face detection module, persons counting module, module for counting abandoned objects, etc for solving specific practical problems without modification of software source code.

Each manufacturer provides unique libraries, which allow interacting with theirs devices. These libraries include a set of functions that can be used by video surveillance system software. The exception is *onvif* protocol, which is the result of an international organization of Open Network Video Interface Forum (ONVIF) work for creating a standardized protocol for interaction with devices of different manufacturers. For interaction with devices that use this protocol, the library similar to the libraries of manufacturers must be developed.

The main functions provided by the manufacturer's libraries, which are needed for creating intelligent surveillance system software with the requirements described above, are the following:

– authorization/deauthorization on the device;
– receiving and displaying a video stream;
– receiving a video stream without displaying it;
– recording a video stream;
– playback of stored video stream;
– saving images from a video stream;
– receiving data (encoded frames) from a video stream through the callback function;
– displaying received data of a video stream from the corresponding callback function;
– obtaining events from devices via callback functions;
– obtaining decoded frames via callback functions;
– obtaining individual frames from video stream;
– displaying data on a top of a video stream (for recognition of license plate's, highlighting objects, etc).

Therefore, all the functions, except the sending of video stream, are present in the libraries provided by manufacturers.

The total sequence of actions for interacting with devices taking into account a predetermined functional is as follows (see Fig. 1, optional actions are shown in dotted lines):

– authorization on the device;
– event's data retrieving if necessary;
– start receiving video stream;
– performing appropriate action with a video stream (display it, save images, recording to hard drive);
– stop receiving video stream;
– logout on the device.

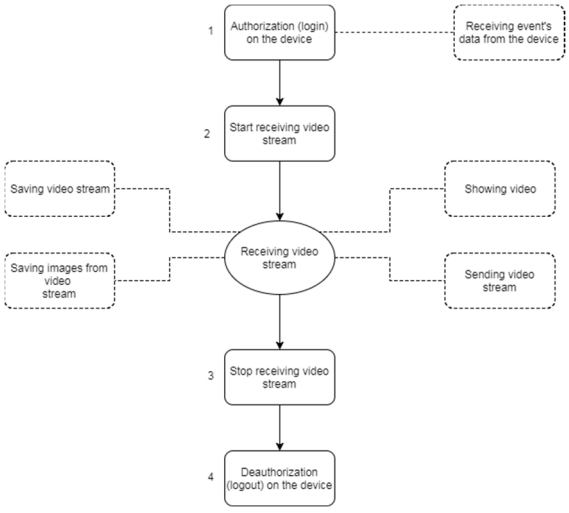

Fig 1. The sequence of actions for interaction with devices

Each device in a video surveillance system has a one (IP camera) or more (NVR) video channels. Each video channel has at least two video streams – primary and secondary. The primary video stream has a high resolution and excellent picture quality. The secondary one – low resolution and worse, compared to the primary stream, picture quality.

Logical and functional components. Now, software is rarely developed from scratch. As a rule, for performing necessary functions existing libraries are used in order to reduce development time and the final cost of the application.

Typically, the libraries that are used in the application do not affect its architecture, however in this case the researches showed that the libraries of the manufactures work differently, and that it is impossible to construct effective architecture of the software without regard to their work.

Therefore, the development of software architecture for video surveillance systems consists of two stages. At the first stage, the so-called logical components of the architecture are allocated, which do not take into account the work of manufacturer's libraries, with the goal of splitting the application into separate modules that perform necessary functions. At the second stage, the so-called functional components are allocated, which take into account the work of manufacturer's libraries and form the final architecture of the application.

The software architecture at the level of logical components.

In Windows to display a video stream from specific device it is needed to pass the window object handle to a corresponding library. Since there must be the ability to display up to 64 video streams and the ability to display events from corresponding IP cameras, one of the components of the architecture must be a component "Screen" that will show the video stream and information about events. In addition, this component should be responsible for user interaction with the software (start recording, stop recording, save the image from the video stream, etc).

One of the components of the architecture must be a component that is responsible for the recording of video streams according to a schedule or event. This component must work on separate thread for each camera, which has a set up schedule. These threads will enable or disable recording, depending on the schedule settings, the current time and events on the respective device. The component with the described functionality will be called "VideoSchedule".

To transfer the video stream to other software it is necessary to allocate a separate component. As "VideoSchedule", this component must work on a separate thread for each camera, from which it is necessary to transfer the video stream. This component will be called "VideoSender".

For the functioning of intelligent modules, a component which will upload and store them in a special list is required. This component will be called "ModulesContainer".

The above components should not interact directly with manufacturer's libraries for performing the appropriate actions, since it will lead to the redundant source code. To eliminate this redundancy the architectural pattern "mediator" can be used [13]. The essence of such pattern is in the introduction of an additional component, which will be located between components and libraries of manufacturers. Such component will be called "CameraView". This component must call the appropriate library function to perform a certain action. For example, to display a video stream the authorization function and video receive function with window handle as parameter must be invoked, for storing video – the authorization function, the function of receiving a video stream and the function of preserving a relevant video stream. Also, this component should interact with "ModulesContainer" to create module's objects for each video stream, and should call necessary functions in order to receive the video frame and draw data on the displayed stream by module's objects. Actually "CameraView" should encapsulate various realizations of SDK's functions for obtaining the required data for the functioning of the modules.

The last necessary logical component is the component responsible for playing of stored video streams. It will called "VideoPlayer". To display the stored video on the screen, it can be used the "Screen" component. The interaction between "VideoPlayer" and libraries will

not lead to the redundancy of the source code because except this one there are no more components that interact with the functions for playing stored video streams.

The software architecture at the level of the logical components shows in Fig. 2. The arrows show the direction of the interaction between the individual components. The arrow between the "manufacturer's libraries" and "VideoSender" means data transfer to the component by the libraries via calling installed callback functions.

Shown in Figure 2 architecture has certain drawbacks.

Firstly, library functions of all manufacturers are not able to parallelize the incoming video stream from the device to reduce network load. Because of this, the number of video streams of one channel retrieving simultaneously from the device can reach four: one stream for playing and for recording on the user demand, two video streams for event recording with prerecord and one for sending a video stream to other software. Implementation of prerecording requires two video streams because libraries do not support circumcision of the stored video stream. That is why prerecording with a one video stream becomes impossible. Four video streams from a device quadruple the load on the network. When using a primary video stream with a bit rate of 4 Mbit/s it would be 12 Mbit/s on the network. When at least 10 cameras are connected, the software work will require a gigabit network. Moreover, each intelligent module requires a separate video stream. So, if software, for example, have 5 modules, then a video stream with a bit rate of 4 Mbit/s would require at least 20 Mbit/s network speed. Thus, such architecture creates a large exceed load on the network.

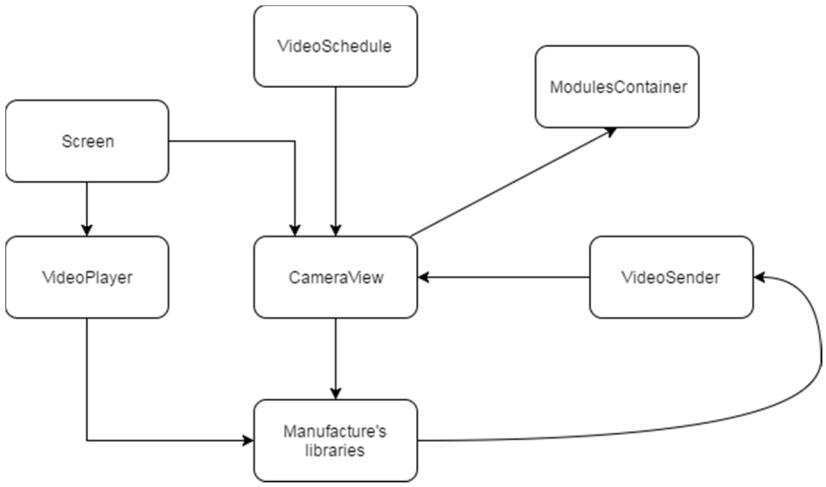

Fig. 2. The software architecture at the level of the logical components

Secondly, it was found that although the library's API is similar, its functions performing differently. For example, Dahua manufacturer libraries require authorization every time when it is necessary to receive a video stream. Because of this, it is impossible to login on the device once and receive video streams from different device channels. Libraries of other manufacturers do not have the authorization limit and it allows receiving a video stream from multiple channels with one authorization. It was also found that Aevision's devices do not support more than 16 per device authorizations from one IP address. This means that if implement the architecture that is shown in Figure 2, the maximum number of channels from which a video stream can be received from Aevision's device are 16. This limits the use of such software architecture with Aevision's devices.

The software architecture at the level of the functional components. For the above reasons it is necessary to create architecture at the level of functional component. Such architecture must allow:

– use one video stream for any purpose from a single video channel;
– receive video streams from all device channels;
– simplify the implementation of "CameraView" component.

Any interaction with device starts with the authorization process and ends with the process of unauthorization. Since manufacturer's libraries implement the authorization process differently, it is necessary to encapsulate it from the rest of the architectures to ensure its extensibility and to obtain all the streams available on the device. The component responsible for the authorization encapsulation will be called "Logginner". It must return the handle of the appropriate authorization on the device. Thus, for Dahua devices this component, every time when referring to it, must login to the device and return the new handle of authorization. For other types of devices it login on specific device only once and store the appropriate handle which is available at any time.

After authorization a lot of functions for work with device are available, including setting of callback functions for receiving data of events from the devices. In fact, it is needed only once to set such function for appropriate device for receiving all events. From the fact that for all vendors except Dahua authorization should be carried out only once and that the callback function for receiving events must be set only once follows that the easiest way to implement the receipt of all events is to implement callback functions in the component that is responsible for the authorization on device. Implementation of an additional component for obtaining information about the events will lead to the needing of implementation the interaction with the "Loginner" that would complicate the system architecture.

One of the requirements of the architecture is the requirement that only one stream per channel must be receiving from the device in order to avoid an excessive load on the network. For this, it is necessary to create a component – the source of the video data that will receive the video stream's data and transmit it to all the necessary components of the software. This component will be called "FrameSourcer". Components that require a video stream from a particular device must "subscribe" to the appropriate video stream and pass to this component the callback function that will be called every time when new data from the respective device has arrive. If a component is "signed" on the video stream, which still does not come from the device, then "FrameSourcer" asks "Loginner" component for authorization descriptor, calls the specific library function for receiving a video stream and passes to it its own callback function that will be called every time when library gets a new bit of data. This "FrameSources's" callback function must call the callback functions of those components that have subscribed to the corresponding video stream. Also it makes sense to pass to this component the relevant callback functions which are intended to receive events from the device. "FrameSources" should not work with these functions but must pass it to "Loginner" component. This will lead to encapsulation of "Loginner" component and its use only by "FrameSourcer". Thus, other components of the architecture will not be aware of the presence of any component for obtaining data from a device except "FrameSourcer". This will facilitate the interaction of various components of the architecture.

Receiving the video stream by only one component and it's paralleling lead to the disappearance of excessive load on the network. However, this solution has a disadvantage. While the data is being processed by one function, the others are waiting for completion of it. Therefore, if the callback function of any component will process income data for too long then the software will freeze. Because of this, all the callback functions should be implemented so that the data are processed as quickly as possible.

For recording a video stream to the file the library's functions can be used. However, in such case the device starts transmit the video stream over a network regardless of whether it already comes from the device or not. To eliminate redundancy of video streams the "FrameSourcer" component must be used. It was found that the saved video stream via the library functions of such manufacturers as Dahua, Hikvision and Aevision is the data which come in a callback function that receives the video stream. As described above, this function is set in the component "FrameSourcer" and calls all of the functions that have been "signed" for the corresponding video

stream. Therefore, for saving the video stream of these manufacturers it is necessary to save the data that was transferred by "FrameSourcer" to the corresponding callback function.

Since the video stream saving is a separate feature, it makes sense to create a separate component for it. This component will be called "FrameSaviour". The objectives of this component are obtaining data from "FrameSourcer" and saving it to a file. In addition, with the objective of reducing the load on the hard drive of this component, it makes sense to use an intermediate buffer which must store data before writing it on the disk. The size of this buffer must be larger than the size of the data coming, but not very large to avoid ram over usage (because all 64 channels can be recorded). It was found that the maximum size of the incoming data at a resolution 2048x1536 is about 200 kb (coded I-Frame [14]). So the size of the buffer has been selected 2 MB taking into account bigger bitrate and bigger resolution. In the worst case – when recording 64 video streams at the user's request and 64-event streams with prerecord – it will use 384 MB of RAM.

Because of "FrameSourcer" the functionality of the "CameraView" component must be changed. Now, this component should not access the libraries directly but must access a "FrameSourcer" component for receiving video stream in order to eliminate the excess load on the network. As for playing the received video data, the manufacture's libraries have functions for showing it. In addition, theirs should be used. In addition, this component should also be responsible for saving the image in a separate file from the video stream. Besides, it should take the frames through the setting callback functions and transfer theirs into modules for further processing.

Received events from devices must be captured and processed. Since "FrameSourcer" component encapsulates "Loginner" component, which actually received events from devices and send it to other components, it is necessary refer to "FrameSourcer" component for obtaining information about events.

Is this architecture there is no sense to implement the interaction between the "VideoSender" components and "CameraView" because in such case "CameraView" will act as an intermediary and will not do any necessary functions. So, it makes sense to implement the interaction between "VideoSender" component and "FrameSourcer" component directly.

The software architecture at the level of the functional components are shown in Figure 3.

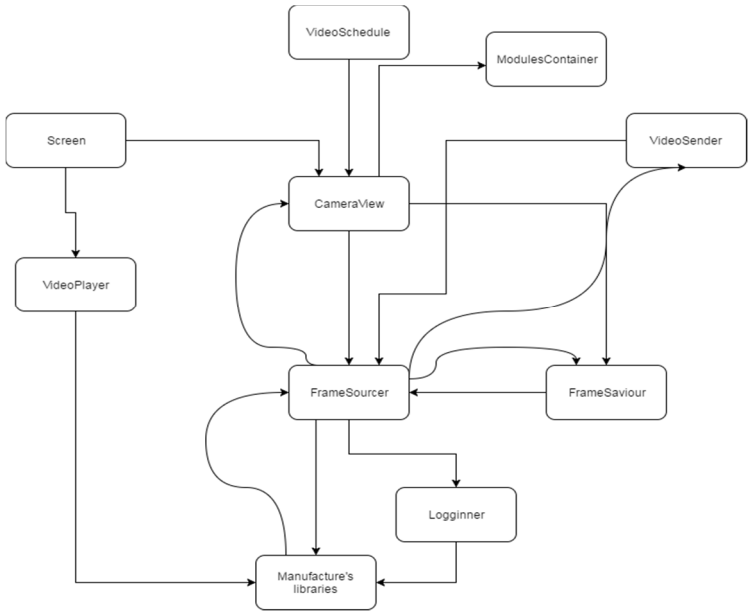

Fig. 3. The software architecture at the level of the functional components

Conclusions. The proposed architecture allows using many types of cameras in single software, which is much more convenient than using many programs for many types of cameras. It minimize network load by using only one video stream from one channel, allows to connect all

the channels of devices of supported manufacturers and to use all necessary functions for video surveillance systems of supported IP cameras. It does not lead to the redundancy of the source code or its great complexity. Thus, software is not difficult to maintain and add new functionality.

References

1. *What is IP camera?* Retrieved from http://www.proximasp.ru/ip-camera.html.
2. *What is NVR?* Retrieved from http://inprog.kz/news/что-такое-nvr/.
3. *Some aspects of the design of IP-surveillance systems.* Retrieved from http://www.algoritm.org/arch/arch.php?id=73&a=1716.
4. *Decreasing Networking and Storage Costs of Your IP Video Surveillance System.* Retrieved from https://www.securitymagazine.com/ext/resources/whitepapers/Genetec-Bandwidth-Management-White-Paper.pdf.
5. *Architecture of video surveillance systems based on IP networks.* Retrieved from http://www.dipolnet.com/architecture_of_video_surveillance_systems_based_on_ip_networks_bib701.htm.
6. *Design and Optimization of the VideoWeb Wireless Camera Network.* Retrieved from https://jivp-eurasipjournals.springeropen.com/articles/10.1155/2010/865803.
7. *Building Digital Security & Surveillance Systems Based on Intel Technology.* Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/presentation/dss-systems-intel-technology-guide.pdf.
8. *HikVision.* Retrieved from http://www.hikvision.com.
9. *Dahua.* Retrieved from http://www.dahuasecurity.com.
10. *Uniview.* Retrieved from http://en.uniview.com.
11. *Aevision.* Retrieved from http://www.aevision.com.cn.
12. *Onvif protocol.* Retrieved from http://www.onvif.org.
13. *Pattern "Mediator".* Retrieved from http://cpp-reference.ru/patterns/behavioral-patterns/mediator.
14. *Overview of the H.264/AVC Video Coding Standard.* Retrieved from http://ip.hhi.de/imagecom_G1/assets/pdfs/csvt_overview_0305.pdf.

УДК 004.4

Руслан Заровський, Андрій Радченко

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМ ВІДЕОСПОСТЕРЕЖЕННЯ З РІЗНИМИ ТИПАМИ КАМЕР

Актуальність теми дослідження. *Зазвичай програмне забезпечення, яке виконує функції NVR на звичайному ПК, підходить тільки для певних типів камер. Відповідно, використання камер багатьох виробників у системі відеоспостереження призводить до використання великої кількості різних програм. Це створює незручності для користувача, тому що для виконання необхідних функцій (перегляду, запису відео та ін.) на різних камерах необхідно запускати різне програмне забезпечення. Тому існує потреба створення програмного забезпечення, яке підтримує різні типи камер.*

Постановка проблеми. *Неоптимальна реалізація архітектури програмного забезпечення, яке підтримує пристрої різних виробників, може призвести до ускладнення розуміння вихідного коду, не оптимального використання мережевих ресурсів тощо. Таким чином, існує проблема побудови відповідної архітектури програмного забезпечення для усунення цих проблем.*

Аналіз останніх досліджень і публікацій. *Аналіз публікацій дозволив виявити загальні тенденції при побудові архітектур систем відеоспостереження, серед яких зменшення витрат на створення мережі та зберігання даних. Зменшення мережевих витрат передбачає застосування спеціальних заходів для мінімізації загального обсягу переданих мультимедійних даних. Це може бути досягнуто завдяки архітектурі системи відеоспостереження, яка усуває повторну передачу тієї ж інформації і загалом мінімізує обмін інформацією в мережі. У цілому в публікаціях описуються архітектури систем відеоспостереження, але не описується архітектура програмного забезпечення для таких систем.*

Виділення не вирішених раніше частин загальної проблеми. *Нині немає відкритої архітектури програмного забезпечення, яка підтримує камери різних виробників.*

Постановка завдання. *Мета цієї роботи описати архітектуру програмного забезпечення, що підтримує сумісність з камерами та NVR від провідних китайських виробників, таких як Hikvision, Dahua, UniView, Aevision, а також пристроями, що працюють за універсальним протоколом Onvif.*

Виклад основного матеріалу. *Архітектура, яка працює з різними типами камер, повинна бути відповідно розроблена. Насамперед необхідно побудувати архітектуру на рівні логічних компонентів, а потім на рівні функціональних компонентів. Архітектура програмного забезпечення на рівні логічних компонентів складається з компонентів Screen, VideoPlayer, VideoSchedule, CameraView, ModulesContainer та VideoSender. Архітектура програмного забезпечення на рівні функціональних компонентів складається з компонентів Screen, VideoPlayer, VideoSchedule, CameraView, ModulesContainer, VideoSender, FrameSourcer, FrameSaviour та Logginner.*

Висновки. *Запропонована архітектура дозволяє використовувати багато типів камер в одному програмному забезпеченні, що набагато зручніше, ніж використання багатьох програм для багатьох типів камер. Така архітектура мінімізує завантаження мережі завдяки використанню лише одного відеопотоку з одного каналу, дозволяє*

підключати всі канали пристроїв підтримуваних виробників і використовувати всі необхідні функції для систем відеоспостереження підтримуваних IP-камер. Вона не створює надмірності вихідного коду або його великої складності. Таким чином, програмне забезпечення не важко підтримувати і додавати нові функціональні можливості.

***Ключові слова**: програмне забезпечення; архітектура ПО; відеоспостереження; IP камера.*

Рис.: 3. Бібл.: 14.

УДК 004.4

Руслан Заровский, Андрей Радченко

АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СИСТЕМ ВИДЕОНАБЛЮДЕНИЯ С РАЗНЫМИ ТИПАМИ КАМЕР

В статье описана универсальная архитектура программного обеспечения для систем видеонаблюдения, в которых используются IP камеры ведущих производителей и которая максимально раскрывает функционал IP камер при минимизации нагрузки на сетевую инфраструктуру. Данная архитектура рассмотрена как на уровне логических, так и функциональных компонент. Описаны ограничения относительно программных библиотек, которые поставляются с IP камерами, и процесс взаимодействия с ними в ходе реализации предложенной архитектуры.

***Ключевые слова:** программное обеспечение; архитектура ПО; видеонаблюдение; IP камера.*

Рис.: 3. Библ.: 14.

Zarovsky Ruslan – PhD in Technical Sciences, Associate Professor, Chernihiv National University of Technology (95 Shevchenka Str., 14027 Chernihiv, Ukraine).
Заровський Руслан Владиславович – кандидат технічних наук, доцент, Чернігівський національний технологічний університет (вул. Шевченка 95, м. Чернігів, 14027, Україна).
Заровский Руслан Владиславович – кандидат технических наук, доцент, Черниговский национальный технологический университет (ул. Шевченко 95, г. Чернигов, 14027, Украина).
E-mail: rolandzar@ukr.net
ORCID: http://orcid.org/0000-0001-5598-1879
ResearcherID: R-2937-2016

Radchenko Andrii Oleksiyovich – PhD student, Chernihiv National University of Technology (95 Shevchenka Str., Chernihiv, 14027, Ukraine).
Радченко Андрій Олексійович – аспірант, Чернігівський національний технологічний університет (вул. Шевченка 95, м. Чернігів, 14027, Україна).
Радченко Андрей Алексеевич – аспирант, Черниговский национальный технологический университет (ул. Шевченко 95, г. Чернигов, 14027, Украина).
E-mail: teor292@gmail.com
ORCID: http://orcid.org/0000-0002-5019-8364
ResearcherID: R-2879-2016

Zarovsky, R., Radchenko, A. (2017). Architecture of software for video surveillance systems with different types of cameras. *Technical sciences and technologies*, no. 4 (10), pp. 101-109.