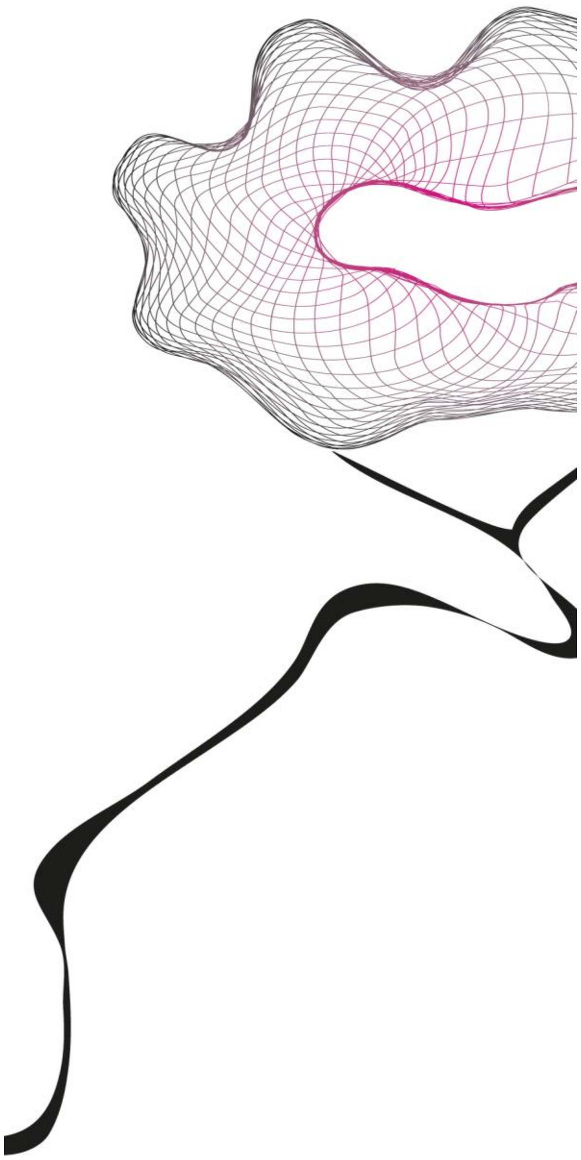
MASTER THESIS



MIDDLEWARE FOR INTERNET OF THINGS

Shirin Zarghami

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS
AND COMPUTER SCIENCE
SOFTWARE ENGINEERING

EXAMINATION COMMITTEE
dr. Luís Ferreira Pires
dr. Maya Daneva
ir. A. Dercksen
ir. drs. T. Garthoff

UNIVERSITY OF TWENTE.

November 2013

Abstract

The Future Internet enables us to have an immediate access to information about the physical world and its objects. As such, Internet of Things (IoT) has been introduced to integrate the virtual world of information and the real world of devices. Internet of Things covers the infrastructure, which can be hardware, software and services, to support the networking of physical objects. IoT aims to provide a simple interaction between the physical world and the virtual world, by integrating a large numbers of real-world physical devices (or things) into the Internet.

IoT has increasingly gained attention in industry to interact with different types of devices. This popularity cause a demand to use IoT vision for different types of device. While each type of devices can support its own communication protocol and required data to provide data for each interaction. This heterogeneous device interaction cause difficulties to interact with the devices to gather information from the environment. The solution that has introduced in the literatures is defining a middleware layer between the devices and the user of the IoT- based system.

In this research, we investigate on developing a middleware for an IoT-based system like video Monitoring System (VMS), to facilitate configuration and deployment for non-expert end-users. A VMS is responsible to provide full video coverage to monitor an area for an end-user, such as a guard. The configuration and deployment can be facilitated by providing a homogeneous Graphical User Interface to interact with different types of camera in a uniform way.

A VMS must support the technical details of different types of cameras. However, these variations should be hidden from non-expert end-users. Thus, we extract a model from the required features to configure different types of cameras. In this project, we developed a VMS that consist of a Middleware for video Monitoring System (MVMS) and applications, which run on top of the middleware. Our VMS let non-experts end-users configure cameras through communicating with third-party camera service providers which is responsible to apply end-users configuration on cameras.

To evaluate our VMS to achieve the ease of configuration and deployment for non-expert end-users, we developed a prototype and interview with the practitioners in a company which has developed VMS.

Acknowledgment

I believe writing this acknowledgement was the most difficult part of my master's thesis since it was quite hard to write my feelings about the people I appreciate. However, we have a saying in Persian, which is "the words that come from your heart will be fully perceived by hearts of others for sure". I hope it makes true for my words too since they are really my true feelings.

First of all, I would like to thank Luis who spent quite time to change my problem solving approach. Luis, many thanks for all of your criticisms that sometimes really made me disappointed but without your comments and advices, I would have never known as much as I know today.

Maya, I'd truly appreciate you for all your help, support, and kindness. Although, I had your support more in the final part of my master's thesis, it helped me a lot to finalize it indeed with your encouragements and positive feedback.

I also would like to thank my adorable family, who always support me in my whole life. My wonderful parents, who really supported me both financially and emotionally during my master's study abroad. Without their support, I could not have had the opportunity to even start my studies. Shahin, my dear brother, you had an important role in my progress during my master's studies. Shahin, I'd like to thank you for walking with me, and not instead of me. You helped me to face different challenges during my study, while you respected my independency. My dearest sister, Soude, thanks for all of your kindness. Although you are too busy, you never forgot me that meant a lot to me. My brother, Shahrokh, I really appreciate your support and considerations.

Furthermore, I'd like to thank all my friends. Alireza and Zahra, many thanks for your support and help to find useful references in my research. Arefeh, thanks for your favor to review my thesis report. Martijn, thanks for listening patiently to my words and making me calm when I was tired or angry. Mohammad, thanks for sharing your experience with me in order to improve my report.

Last but not Least, I'm thankful to my colleagues at Nedap, especially Albert, Tim and Erik. Albert, thanks for respecting my curiosity about the company's system and motivating me to work hard. You and Tim always answered to my questions patiently. Erik, I'd like to thanks you for sharing your experiences with me. I learned a lot about software architecture from you.

Table of Contents

Chapter 1 Introduction

The structure of this chapter is the following: Section 1.1 defines the motivation for this work. Section 1.2 shows the research objectives and questions. Section 1.3 presents the steps required to achieve our objectives. Section 1.4 shows the structure of the report.

1.1 Background

The Future Internet goal is to provide an infrastructure to have an immediate access to information about the physical world and its objects. Physical objects can be applicable to different application domains, such as e-health, warehouse management, etc. Each application domain may have different types of physical devices. Each physical device can have its own specifications, which is required to use in order to interact with it. To achieve the future Internet goal, a layered vision is required that can facilitate data access. Internet of Things (IoT) is a vision that aims to integrate the virtual world of information to the real world of devices through a layered architecture.

The term 'Internet of Things' consists of two words, namely *Internet* and *Things*. *Internet* refers to the global network infrastructure with scalable, configurable capabilities based on interoperable and standard communication protocols. *Things* are physical objects or devices, or virtual objects, devices or information, which have identities, physical attributes and virtual personalities, and use intelligent interfaces [1]. For instance, a virtual object can represent an abstract unit of sensor nodes that contains metadata to identify and discover its corresponding sensor nodes. Therefore, IoT refers to the *things* that can provide information from the physical environment through the Internet.

Middleware is as an interface between the hardware layer and the application layer, which is responsible for interacting with devices and information management [2]. The role of a middleware is to present a unified programming model to interact with devices. A middleware is in charge of masking the heterogeneity and distribution problems that we face when interacting with devices [3].

1.2 Motivation

IoT-based system is in charge of providing knowledge from an environment to an non-expert user. IoT-based system can be used in different environments, so it needs to be able to address many heterogeneous devices. Thus, a major concern within developing an IoT-based system is how to handle the interaction with the heterogeneous devices for non-expert users. This concern can be addressed by a middleware layer between devices and non-expert users. This layer is responsible to hide the diversity of devices from the user perspective, and provides access transparency to the devices for the end users.

The idea of creating abstractions of devices been addressed in the literature. The middleware we found in the literature can provide satisfaction by facilitating the interaction with devices, but they do not support low-level device configuration [4].

1.3 Objective and Research Questions

In this research, we work toward an architecture for a middleware for IoT-based systems, based on [5], which provides a simple and flexible interface to interact and configure different devices. This middleware allows users to completely control physical devices by masking their heterogeneity. We work on the Video Monitoring System (VMS), to develop a middleware with an uniform interface to interact with different types of camera devices.

A video Monitoring System (VMS) is in charge of providing full video coverage to monitor an area for an end-user, such as a guard. Since different types of camera can be connected to a VMS, an end-user needs to know how to configure each of them. This can cause some difficulties for end-user to configure the cameras. To facilitate camera configuration, we need to have an abstract layer on top of the cameras. This layer provide a unique way for end-users to configure any types of cameras.

The *main* research objective of this thesis is:

To develop a middleware for an IoT-based system like VMS, to facilitate configuration and deployment for non-expert end-users.

The term "*facilitate configuration and deployment for non-expert end-users*" have been defined by Henricksen, K. et. al [6] as "The distributed hardware and software components of a context deployment aware system must be easily deployed and configured to meet user and environmental requirements, potentially by non-experts"

To achieve this goal, we will define a generic model of the data required to configure and interact with different camera devices. Toward this and in this research, the following research questions will be addressed:

Q1. *What is the role of middleware in an IoT-based system*?

Q2. *What are the main functional components of a middleware in an IoT- based system*?

Q3. *How to facilitate the interaction of non-expert users with different types of devices?*

Q4. *How to verify ease of this interactions with different types of devices ?*

In this work we answer these questions for the special case of Video Monitoring System (VMS)

1.4 Empirical Research Approach

To achieve the main objective of this research and answer the research questions, the following research process has been taken (Figure 1.1):

1. Study the literature about IoT-based system definition and challenges.

2. Interviews practitioners of a company that develops IoT-based systems to identify their requirements on the middleware for these systems.

3. Design and implement a middleware for VMS, which accomplish with the both functional and non-functional requirements that have been identified.

4. Test the middleware of a case study in which a prototype application has been built as support a usage scenario.

5. Interview with the practitioners in the company, and analyzing its results. In order to evaluate if the middleware can meet the defined objective.

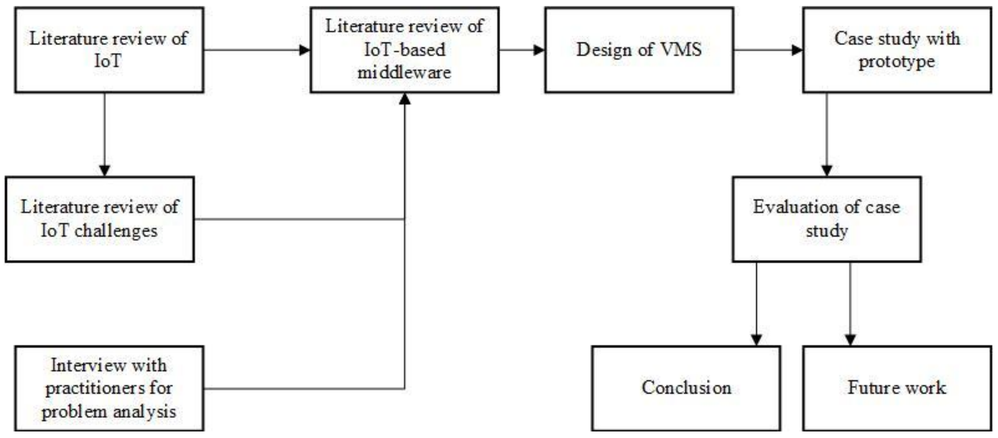

Literature review of IoT
Literature review of IoT-based middleware
Design of VMS
Case study with prototype
Literature review of IoT challenges
Evaluation of case study
Interview with practitioners for problem analysis
Conclusion
Future work

Figure 1.1 Research approach

The research approach has been inspired by the design science method of Hevner [7]

1.5 Structure

The remainder of this thesis is structured as follows:

Chapter 2 gives the background of our work. We explain IoT vision and discuss the functionalities that a middleware for an IoT-based system should support.

Chapter 3 reviews some middleware for IoT- based systems and discusses their features.

Chapter 4 reports of both functional and non-functional requirements that our middleware should address. We extract these requirements based on reviewing the literature and the result of interviews, which we performed with practitioners in a commercial company

Chapter 5 proposes middleware architecture to support most of the requirements identified in Chapter 4.

Chapter 6 reports implementations of a prototype of a middleware for VMS.

Chapter 7 evaluates our middleware with respect to ease of configuration and deployment for non-expert end-users.

Chapter 8 provides answer to the research questions of this thesis, the key conclusions and the recommendations for the further research.

Chapter 2 Background for this research

In this chapter, we explain the IoT definition and the challenges to develop an IoT-based system that are independent from the application domain. One of the main challenges to develop an IoT-based system is developing a middleware between the user of the system and heterogeneous devices in the system in a homogeneous way. Therefore, we identify the required functional components to develop a middleware for an IoT system. We called all the system that fit into IoT definition as an IoT-based system.

This chapter is structured as follows: Section 2.1 introduces IoT by explaining the IoT layered architecture. Section 2.1 briefly explains the developing of a IoT-based systems. Section 2.3 explains the required functional components to develop a middleware for an IoT-based system.

2.1. IoT definition

In this section, we explain some of the IoT definitions. Also, we explain the layered architecture for IoT.

Internet of Things (IoT) has increasingly gained attention in industry to interact with different types of devices. IoT can have influence on industry and society by integrating physical devices into information networks [8]. IoT impacts can be on different perspectives, namely for private and business users. From the perspective of a private user, IoT has effect on both working and personal fields, such as smart homes and offices, e-health and assisted living. From the aspect of a business user the impacts would be in fields such as automation and industrial manufacturing, logistics, business process management, intelligent transportation of people and goods [9]

IoT integrates physical things into information networks. IoT covers the overall infrastructure, including software, hardware and services, which is used to support these information networks. The integrated physical things can exchange data about the physical properties and information that they sense in their environment. To identify devices, we can use identification technologies like for example RFID, which allow each device be uniquely identified [10].

International Telecommunication Union (ITU)[1] defines IoT as "*A global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies*"[11]

[1] http://www.itu.int

[2] http://www.iot-a.eu/

Internet of Things-Architecture [2] (IoT-A) defines it as "*The idea of a globally interconnected continuum of devices, objects and things in general emerged with the RFID technology, and this concept has considerably been extended to the current vision that envisages a plethora of heterogeneous objects interacting with the physical environment*."[12]

IoT has a layered architecture designed to answer the demands of various industries, enterprises and society. Fig. 2.1 shows a generic layered architecture for IoT that consist of five layers, which are discussed, in the following:[13]

- Edge Technology layer

  This is a hardware layer that consists of embedded systems, RFID tags, sensor networks and all of the other sensors in different forms. This hardware layer can perform several functions, such as collecting information from a system or an environment, processing information and supporting communication.

- Access Gateway layer

  This layer is concerned with data handling, and is responsible for publishing and subscribing the services that are provided by the *Things*, message routing, and hovelling the communication between platforms.

- Middleware layer

  This layer has some critical functionalities, such as aggregating and filtering the received data from the hardware devices, performing information discovery and providing access control to the devices for applications.

- Application layer

  This layer is responsible for delivering various application services. These services are provided through the middleware layer to different applications and users in IoT-based systems. The application services can be used in different industries such as, logistics, retail, healthcare, etc.

[2] http://www.iot-a.eu/

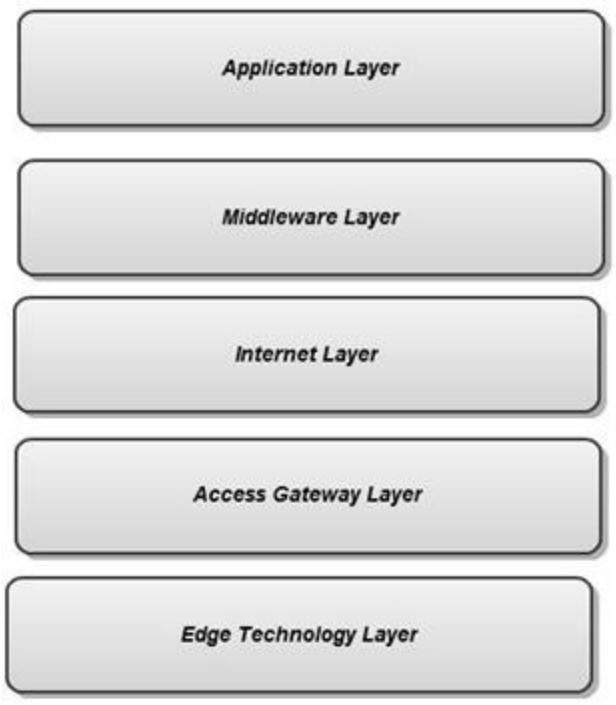


Figure 2-1 Layered architecture of the Internet of Things (IoT).

2.2 Challenges

IoT-based systems can be used for different purposes and areas, so that, we have to face different challenges. In this section, we explain some of the challenges that need to be considered in research activities [10]:

- Edge technology
  At the hardware level, more research efforts are required to develop the technology of embedded devices, sensors, actuators and (passive and active) identification, since an IoT-based system must be able to gather sufficient information about the real world by employing a large variety of devices and environments. Thus, there more work is still required to connect heterogeneous devices and deploy them in IoT applications, and to provide support for new devices.

- Networking technologies

  In IoT, *things* are connected through different kinds of networks, i.e. mobile, wired and wireless network. These networks support bi-directional communication at different levels among the real world objects, applications and services that are employed by the IoT applications. This highly distributed structure should provide interconnection with low energy consumption, while distributed data can cause privacy issues.

- Middleware system

  In IoT, we have heterogeneous devices and networks. Their heterogeneity can potentially increase with new technologies. To facilitate the use of these devices by IoT applications, we should shield their heterogeneity. Therefore, we need to develop a secure, scalable and semantically enriched service-oriented middleware to cope with the heterogeneity of devices.

- Platform services

  They support a superior management of all involved devices in an integrated way, ensuring scalability, high availability, and the safe and secure execution of the requested functionality from devices.

In continue, we focus on the middleware challenges, because we are looking for the functionalities that a middleware can provide for the application layer in the IoT-based systems.

2.3 IoT_based Middleware

Bandyopadhyay, S. et. al. have studied the middleware systems that have been applied in IoT-based systems [1]. They classify the required functionality of middleware to manage interaction with a variety of devices in four functional components, namely (1) interface protocols, (2) device abstraction, (3) central control, context detection & management, and (4) application abstraction (shown in Fig. 2.2). In the following, we explain these components in details.

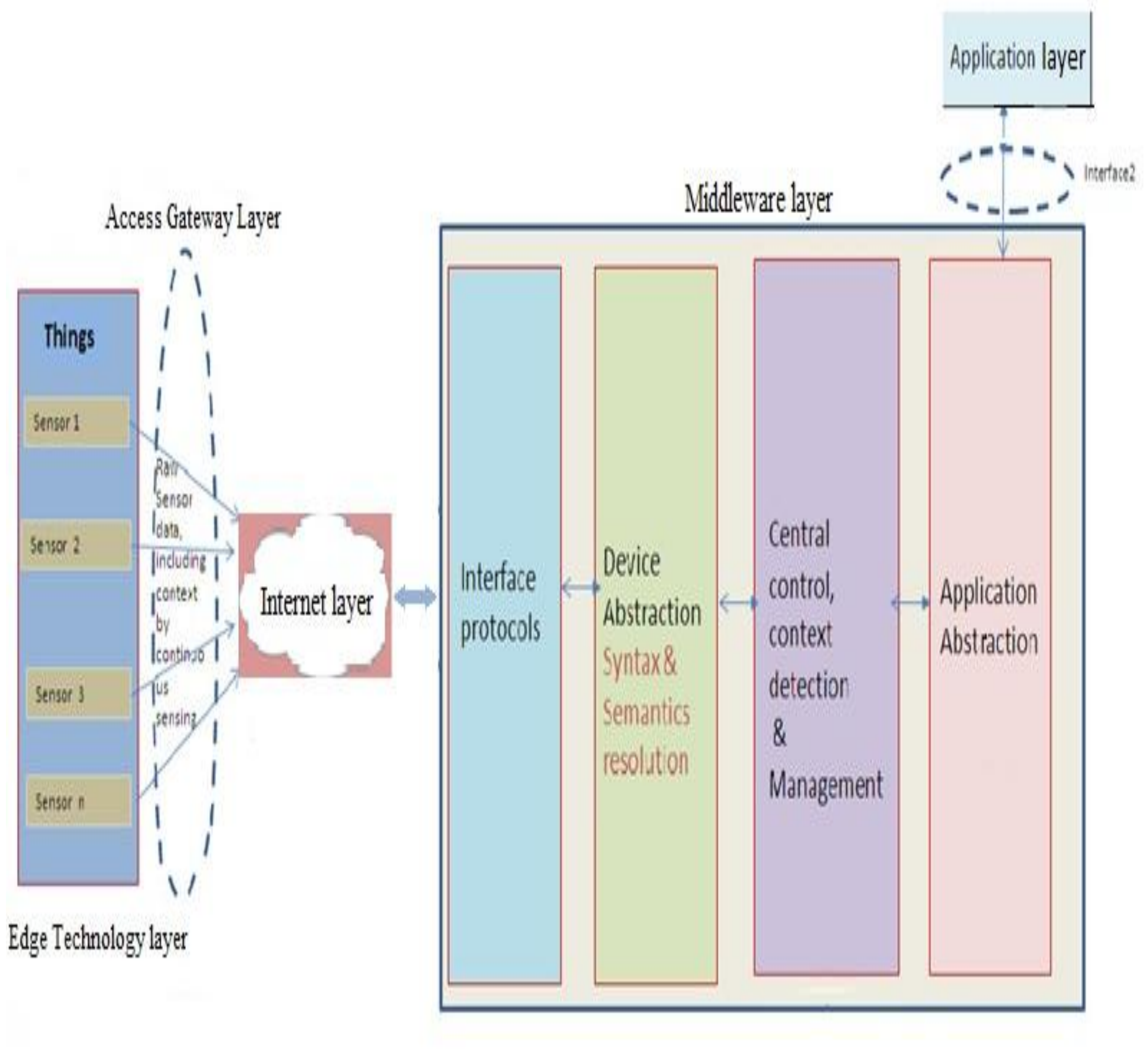


Figure 2.2 Functional components of a middleware for IoT-based systems

2.3.1 Interface protocols

This component is in charge of providing *technical interoperability*. Interoperability in the context of *Interface protocols* means: *the ability of two systems to interoperate by using the same communication protocols*. According to ETSI (European Telecommunications Standards Institute) [14] *technical interoperability* is defined as the *association of hardware or software components, systems and platforms that enable machine-to-machine communication to take place. This kind of interoperability is often centered on (communication) protocols and the infrastructure needed for those protocols to operate*[14].

The Interface Protocol component defines protocols for exchanging information among different networks that may work based on different communication protocols, in order to allow technical interoperability. This component is responsible for handling basic connectivity in the physical and data link, network, transport, and sometimes the application layer of the TCP/IP stack.

To cope with the heterogeneity of devices, we can use a wrapper for each device to translate the protocol supported by the device to a common protocol. This wrapper can be placed either on the device side or the middleware side. If we want to have direct interaction with devices, we should place the wrapper in the middleware side. Devices usually have limited capability of computational process, so this would be a reason to implement wrapper on the middleware side. In contrast, in case of indirect interaction with devices we can develop an intermediary wrapper between the middleware and the devices. The interface protocol component is responsible to allow the middleware to support both direct and indirect interactions.

2.3.2 Device Abstraction (DA)

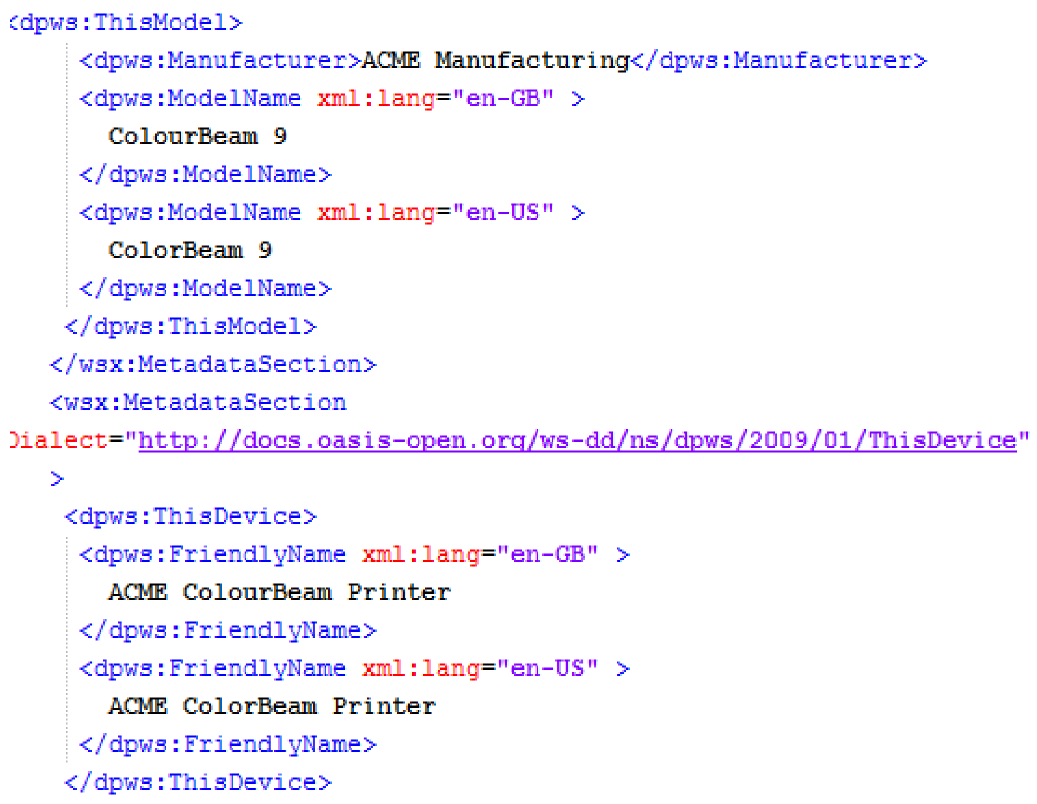
This component is responsible for providing an abstract format to facilitate the interaction of the application components with devices. This abstraction provides *syntactic* and *semantic* interoperability, which are defined by ETSI [14] as follows:

- *Syntactic interoperability* is associated with data formats. The messages transferred by communication protocols must have a well-defined syntax and encoding format, which can be represented by using high-level transfer syntaxes such as, HTML and XML.

- *Semantic interoperability* is usually associated with the meaning of the content of message which is understandable for human. Thus, interoperability on this level means that there is a common understanding among people on the meaning of the content (information) being exchanged among them. However, since DA does not communicate directly with human, semantic interoperability in the context of DA is in charge of providing this common understanding for applications.

  Semantic interoperability relies on semantic models which tends to be domain specific. For example, one way to provide semantic interoperability in Service Oriented (SOA) [15] based middleware is by using Devices Profile for Web Services (DPWS) [16] . In this context, each device type refers to a distinguished service type

  DA component provide two general functionalities: (1) to ask devices to perform some functionality and (2) to define and configure devices DPWS uses the XML format that is shown in Code 2.1.



```
<dpws:ThisModel>
    <dpws:Manufacturer>ACME Manufacturing</dpws:Manufacturer>
    <dpws:ModelName xml:lang="en-GB" >
      ColourBeam 9
    </dpws:ModelName>
    <dpws:ModelName xml:lang="en-US" >
      ColorBeam 9
    </dpws:ModelName>
   </dpws:ThisModel>
  </wsx:MetadataSection>
  <wsx:MetadataSection
Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisDevice"
  >
   <dpws:ThisDevice>
    <dpws:FriendlyName xml:lang="en-GB" >
      ACME ColourBeam Printer
    </dpws:FriendlyName>
    <dpws:FriendlyName xml:lang="en-US" >
      ACME ColorBeam Printer
    </dpws:FriendlyName>
   </dpws:ThisDevice>
```

Code 2.1 XML code to define a device

We explain the tags used in the XML of Code 2.1 as follows:

- `dpws:ThisModel/ dpws:Manufacturer`
  It defines the name of the device manufacturer.

- `dpws:ThisModel/ dpws:ManufacturerUrl`
  It defines the web site of the manufacturer of the device.

- `dpws:ThisModel/ dpws:ModelName`
  It defines the user-friendly name of this model of device as assigned by the manufacturer.

- `dpws:ThisModel/ dpws:ModelNumber`
  It defines the model number of this model of device.

- `dpws:ThisModel/ dpws:ModelUrl`
  It defines a URL of a web page where this model of device is described.

- `dpws:ThisModel/ dpws:PresentationUrl`
  It defines the URL of a presentation resource for this device. It correspond to the URI address from which the metadata of the resource can be retrieved. If Presentation URL is specified, the device can have the resource in multiple formats, but it must at least provide an HTML page.

2.3.3 Central control, Context detection & Management (CCM)

Context characterizes the situation of an entity, which can be a place, a person or an object that is relevant to the user, applications and their interactions [1]. The CCM functional component is responsible to support context-aware computation that is a computational style that take to account the context of the entities that interact with the system. A middleware for IoT-based systems must be context-aware to work in smart environments [1]. Smart environments refer to a *physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network* [17]. Context-awareness includes two functionalities:

1) Context detection, which consists of collecting data from resources, and selecting the information that can have an impact on the computation.

2) Context processing, to use the gathered information to perform a task or make a decision.

2.3.4 Application Abstraction

This functional component provides an interface for both high-level applications and end-users to interact with devices. For instance, this interface can be a RESTful interface or can be implemented with some query-based language.

A. Malatras. et. al. propose a SOA-based middleware to perform data management and data monitoring services [18]. This middleware uses a RESTful interface to facilitate the interaction of high-level applications with sensors, which can communicate with Wireless Sensor Network (WSN) through the HTTP operations: (1) GET to issue a query on an existing resource, (2) DELETE to remove an existing resource, (3) POST to create a new resource, and (4) PUT to update an existing resource. For instance, a client application gets the result of a domain task by sending a GET request through the URL: '`http ://{ hostname}/REST/ {version}/DomaintaskResult/id`'. In this URI that reference to an 'id' leads to a unique result.

In the TinyDB middleware [19], an end-user can use a query language to interact with devices. For instance, a user can send the following query to get the id of a sensor (nodeid) nearby and temperatures that are sensed by this sensor during the past 10 seconds before executing the query:

```
SELECT nodeid, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

2.4 Conclusion

In this chapter, we defined IoT and identified common IoT layers. Furthermore, we discussed a reference middleware architecture for IoT-based systems. This architecture has been proposed by Bandyopadhyay, S. et. al based on a study on the existing middleware frameworks for IoT-based systems [1].

Chapter 3 Review of IoT-based Middleware

The term *Internet of Things (IoT)* refers to a wide set of applications and research areas such as, distributed computing and knowledge management. Much of the IoT research we found in the literature is related to the field of *pervasive computing*. World Wide Web Consortium (W3C)[3] defines *pervasive computing* as a *vision about our future computing life style. In this vision, computer systems will be seamlessly integrated into our everyday life, anticipating our needs and providing relevant services and information for us in an anytime-and-any-where fashion. Pervasive Computing is all about making the human's life easier by exploiting the available computing technologies* [20]. This motivates scientists to develop new technologies for the everyday people's lives.

Since the vision of IoT is almost similar to Pervasive computing, we reviewed the literatures that address both IoT and Pervasive systems. To select the middleware to discuss we had the following approaches:

1) We looked at typical IoT middleware such as ISMP [21], ASPIRE [22] GSN [23]. We selected HYDRA [24] to review, because it is the most popular and well-documented middleware in comparer to the mentioned middleware. Also, based on the study of Bandyopadhyay, S. et. al [3], those middleware does not support the discussed functional components in Chapter 3, while HYDRA supports them.
2) We looked at typical Pervasive system middleware such as [25], [26], [27]. We selected the following middleware to review:
   - AURA [28] because this middleware focus on elaboration and manipulation of the gathered data from devices. we aim to provide ease of configuration and deployment for end user and developer. Since to meet this goal, we require to facilitate the gathered data manipulation, we reviewed AURA that was the only middleware among the reviewed paper that considered the data manipulation.

   - TinyDB [19] focus on gathering data from devices. Since in an IoT-based system we need to gather data from environment through different devices, we reviewed TinyDB that is a popular middleware.

   - WiseMID [29] is the only middleware among the reviewed middleware that is specific for energy saving purpose. As saving the energy of devices is important issue, we reviewed this middleware.

In this chapter, we the mentioned middleware in more detail. Also, we analyze the discussed middleware with respect to the functional components mentioned in Chapter 2.

[3] http://www.w3.org/

3.1 AURA

One of our objectives in this assignment is to find the possibility of improving the ease of configurations and deployment in an IoT-based system. AURA [28] aims to provide high-level APIs to interact with each device, which can facilitate device configurations for end-users. AURA middleware's aim is relevant to our objectives, so we review its architecture.

AURA is a middleware that supports interacting with complex devices (e.g. digital cameras, PDAs, etc.), and their integration. AURA defines a proxy, called *personal AURA* that enables users to use a device independent of their physical locations. Figure 3.1 shows the main components of the AURA framework architecture and their interactions.

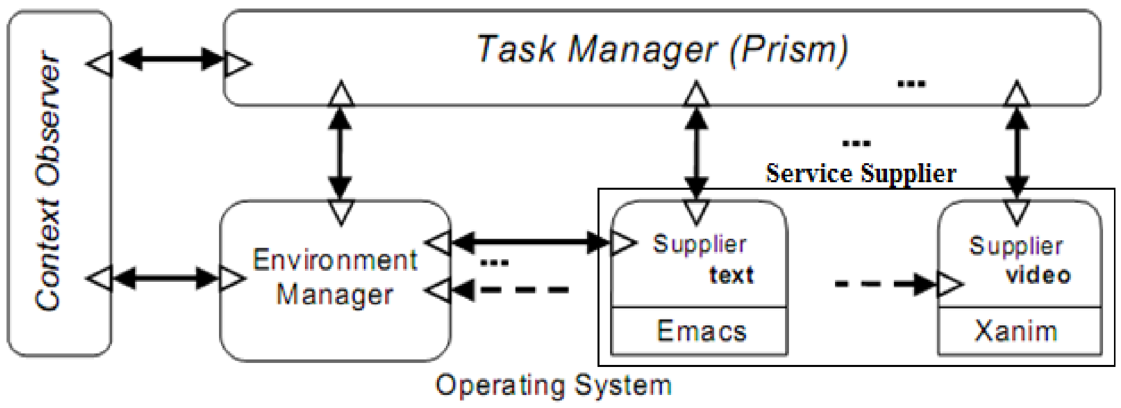


Figure 3.1 The components of the AURA framework architecture and their interactions.

We explain the four main components of AURA, in the following:

1. Task Manager (Prism)

   This component aims to provide the minimum distraction for end-users in case any change happens in the system environment, such as changing the end-user's location or the operating system. This component provides a platform-independent description for end-users' tasks, such as *play video* and *edits text*, which is an abstract service. The service abstraction allows an end-user to request for execution of a task in the same way in different platforms. For instance, to provide the *edit text* task for an end-user in the UNIX environment, AURA can use Emacs while, in Windows environment AURA can use Microsoft Word.

2. Service Supplier (SS)

   This component implements the services by composing task(s) to answer to an end-user's request. This component is similar to a wrapper that maps the abstract service descriptions to application-specific settings. For example, *text editor* can map the editing request from user to *Notepad*, Emacs or Microsoft word.

3. Context Observer (CO)

   This component gathers information about the physical context and accordingly triggers an event for the *Environment Mana*ger and *Task Manager*. The information is about end-users' location, activity, authentication, etc. *Context observer* can support different degrees of complexity according to devices of different types deployed in different environments. If a device has the capability of sensing more features, the CO component can become more complex.

4. Environment Manager (EM)

   This component is a gateway to the environment. It knows which available suppliers provide which services, and where services can be deployed. Also, in case of asking a file by end-user, the EM component supports different ways to access to the file, such as foe example using FTP. To facilitate file access for users, this component encapsulates the detailed information about accessing a file in a distributed environment.

By changing the location of the end-user of a Task Manager, the deployment of the supplier in the new location can be changed. For instance, imagine a user stops working on a file with a text editor on a desktop computer and wants to work on the file through an Ipad, the system is responsible to handle this changes. AURA uses four connectors to hide details of distribution and heterogeneity of service suppliers, as the following:

1. Connector between *Prism* and an arbitrary Supplier.

2. Connector between the Context Observer and the Environment Manager.

3. Connector between Prism and the Environment Manager.

4. Connector between the Context Observer and Prism.

Each of these connectors uses a special protocol based on the component types to which they connect. Each of them may have many implementations to support specific low-level interaction mechanism.

3.2 HYDRA

Hydra [24] is a well-known middleware framework for IoT-based system This middleware covers almost all the functional components discussed in Chapter 3. To provide the ease of deployment and configuration, we are looking for a Service Oriented Architecture that interacts with devices in a loosely couple way. The reason is, a loosely couple IoT-based system can support better system maintainability and extendibility in case of handling changes in the type an number of devices. As Hydra is a SOA-based middleware, and supports many required functionalities to support an IoT-based system, we consider it as our related work,

This project was developed for three application domains, namely building automation, healthcare, and agriculture scenarios [30]. Hydra middleware is intelligent software that is placed between applications and the operating system to handle various tasks in a cost-efficient way. This middleware provides a web service interface to interact with any physical devices, actuators, sensors or subsystems, irrespective of their network interface technologies, e.g. Bluetooth, RF, ZigBee, RFID, WiFi, etc.

This middleware has been designed to facilitate the interaction with devices by abstracting from the detailed information about these devices and their networks. Hydra considers each device as a service, and uses ontology languages, e.g. OWL, OWL-s and SAWSDL, to define semantic descriptions of these devices. Moreover, it provides an intelligent service layer that allows end-users to interact with these devices without dealing with the communication technology that is supported by the devices.

Figure 3.2 shows the components of Hydra architecture and the components that Hydra middleware communicates with.

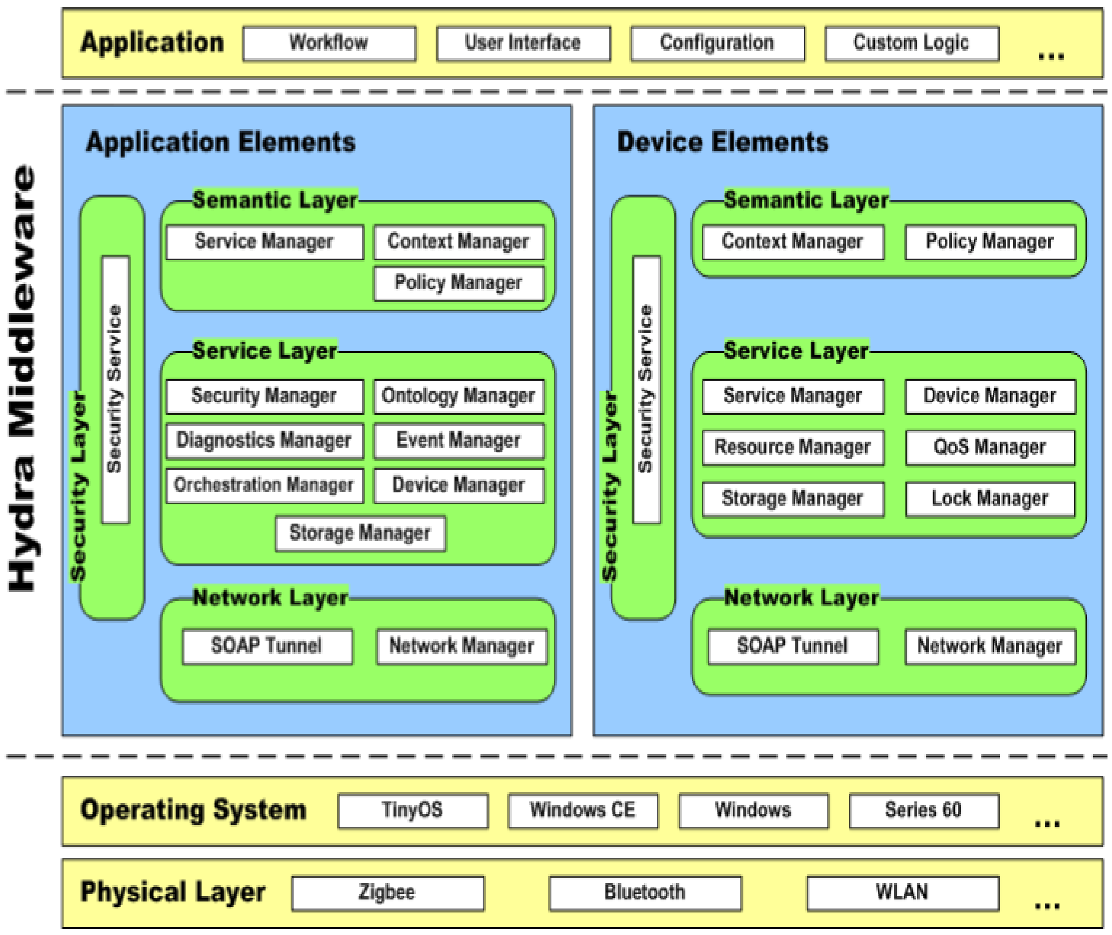


Figure 3.2 Components inside and outside of Hydra middleware.

3.3. TinyDB

TinyDB[19] middleware was the first project to propose the idea of abstracting from devices. TinyDB allows end-users to interact with devices without knowing about the details of the devices specification, such as the communication protocols that are supported by these devices. Since we are looking for a way to abstract from details of devices to facilitate interactions with them, this topic can be relevant to our work.

TinyDB provides a Domain Specific Language (DSL) for end-users to interact with devices. Its DSL is a query language that supports selection, join, projection, and aggregation to work with an embedded sensing environment. This DSL allows an end-user to get information about the time, place, type and method of sampling in an embedded sensing environment. TinyDB supports the following types of queries:

- Monitoring Queries

  It asks the value of one or more attributes periodically and continuously, such as, e.g., reporting the temperature of a warehouse every hour.

- Network Health Queries

  It provides information about the network itself. For instance, selecting neighboring nodes, with a battery lifetime larger than a threshold.

- Exploratory Query

  It shows the status of a specific node or a set of nodes at a specific time, such as, e.g., selecting the temperature of the sensor with same specific id.

- Actuation Query

  This kind of query can be used to ask for a physical action. For instance, an end-user of a system wants to turn off a fan in a room when the temperature of the room is lower than a threshold. A query to perform this has the following format:

```
SELECT nodeid,temp  FROM sensors  WHERE temp < thresholds
 OUTPUT ACTION power-off (node-id)
  SAMPLE PERIOD 12s
```

The OUTPUT ACTION clause defines the external command that is invoked in response to the asked query.

3.4 WISeMid

WISeMid [29] is an energy-aware middleware for integrating wireless sensor networks and Internet. In an IoT-based system, saving energy in interaction among devices is important, because they usually have limited power suppliers. Also, IoT-based system is IP-based communication. Therefore, as the WISeMid middleware considers both IP-based communication and energy awareness factors, we review this middleware.

3.4.1 WISeMid power saving mechanisms

WISeMid focus on integrating the Internet and WSNs at the application level. This middleware proposed several power saving mechanisms, as the following:

- Aggregation service

  The goal of the data aggregation service is to aggregate correlated or redundant data, and reduce the overall size of transferred data in a network. In this way, we can decrease the network traffic and save energy by decreasing the interactions with devices. In this service, a user sends one request and receives an answer based on an aggregation of the last values of the requested devices. In this way the number of transaction decreases and energy can be saved.

- Reply Storage Timeout

  This service stops sending the same messages with the same parameters. For instance, if the sensed data of a device is fixed for a specific period of time, we can send only one request to the considered device, and then use the reply message to answer to all of the equivalent request messages arriving during that specific period. Therefore, WISeMid prevents the system from getting information from sensors when the data is still up-to-date.

- Atomic Type Conversion

  This service decreases the size of messages in an IoT-based system. For instance, if we define an integer data type for a field that gets a numeric value like ‘1’ in this case, we many bytes are unnecessarily used. To save bytes, the argument format can convert from Integer to Short. Thus, WISeMid removes unnecessary bytes from messages.

- Invocation asynchrony patterns

  This service provides four patterns to handle the end-user requests in an asynchronous way. These patterns prevent the system waste time with blocking, when requests can be handled in an asynchronous way. These patterns are as the following:

1) Fire and forget

This pattern supports one-way operations, which have no return values or exception errors. This pattern cannot report any errors to the end-user when an error occurs either when sending the invocation to the remote service, or during the execution of the remote invocation.

2) Sync with Server

This pattern is used when we want to be sure that the request has been received by the server, even if a request has no exception or returned value. In this case, the service invokes a service provider, and then waits for an acknowledgment message from the service provider. We can use this pattern in case a service should be invoked before other services.

3) Poll object

This pattern is based on request and response operations. It checks if an asynchronous response has arrived, and if so, it receives the return value.

4) Result callback

This pattern can trigger an event in end-user side whenever the requested result becomes available.

3.4.2 WISeMid architecture

WISeMid uses a Interface Definition Language (IDL) [29], to describe a service in this middleware. IDL is a unified language to describe a service irrespective of where (Internet or WSN) or what implementation language is used. The IDL contains a module (package) that is as a container for specifying service interfaces. Each service interface includes name and the operation that can by supported by the service. Each operation contains input/output parameters types and may raise exceptions. Its format is the following:

```
Package{
        Service interface name{
        Operations{
                Input/ output parameters type
exceptions
                        }
        }
}
```

Figure 3.3 shows WISeMid architecture, which has three layers, as the following:

1) Common services layer

   This layer contains general services, which are not for particular or a specific application domain. It includes the following services:

   - Aggregation of sensors data.
   - Grouping to define clusters in a WSN.
   - Naming to store the required information to access a service.

2) Distribution layer

   This layer defines the required components to use a service. For instance, Requestor is a component that makes a remote invocation with parameters, such as, e.g. remote service location, service name and arguments on the client side; WISeMid uses WISeMid Inter-ORB Protocol (WIOP) to perform Request/Response interactions. We explain WIOP in Section 3.4.3.

3) Infrastructure layer

   This layer consists of the Server Request Handler and the Client Request Handler. These handlers provide network communication to interact with devices.

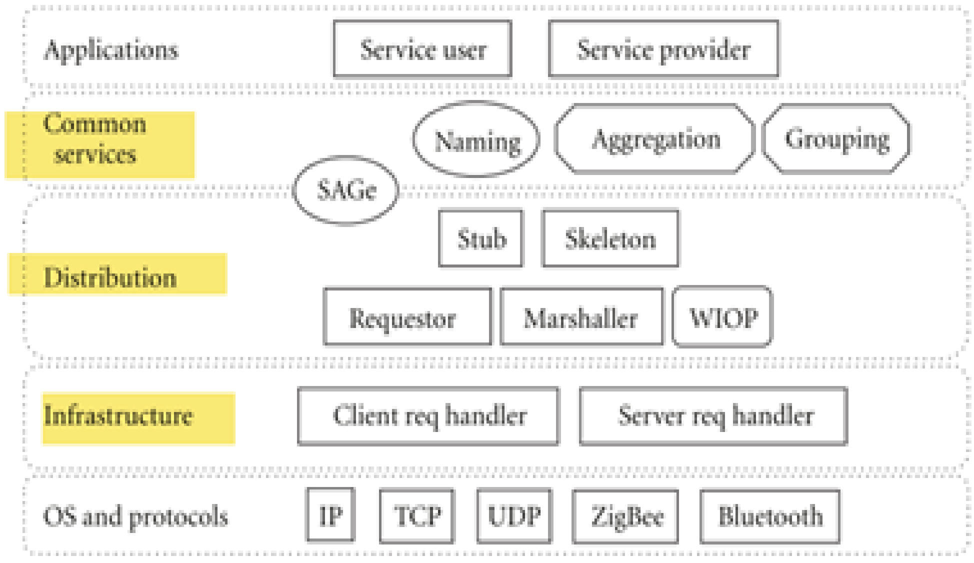


Figure 3.3 shows WISeMid architecture

3.4.3 WIOP protocol

The WIOP protocol defines a format for request or response messages between clients and servers. Each message consists of a header and a body part. There are two versions of WIOP:

1) WIOPi supports communication through Internet.

2) WIOPs support communication in a Wireless Sensor Network (WSN).

Figure 3.4 shows WIOP header has three fields. The *msgtype* field indicates whether a message is a Request or a Response.

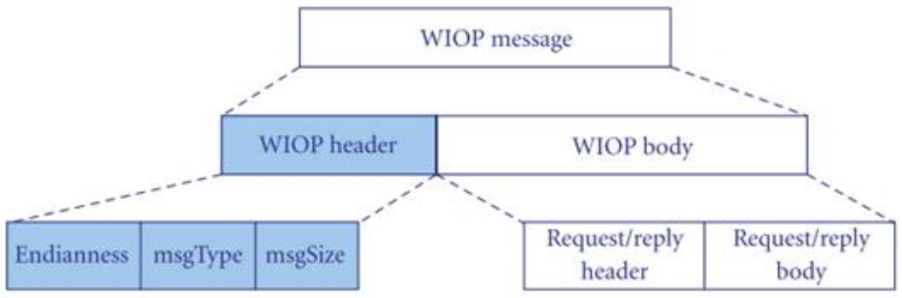


Figure 3.4 WIOP message headers

WIOP body can consist of Request or Reply messages with their own body and header.

- **Request message body**

Figure 3.5 shows the WIOP request message body. The fields in the red are used only in the WIOPs version, and the rest of the fields are common between both WIOP versions.

The *Resp* field indicates whether a request expects a response or not. By defining five operations, we can have access or use a service and register a service in the WISeMID Naming service. The operations defined in the *opr* field are listed, bellow:

1) *Bind* to register a service by its name and associate it with a name.
2) *Lookup* to return the reference associated with a service name.
3) *Rebind* to change the reference that is associated with a service.
4) *Unbind* to unregister a service name.
5) *List* to register all the registered services.

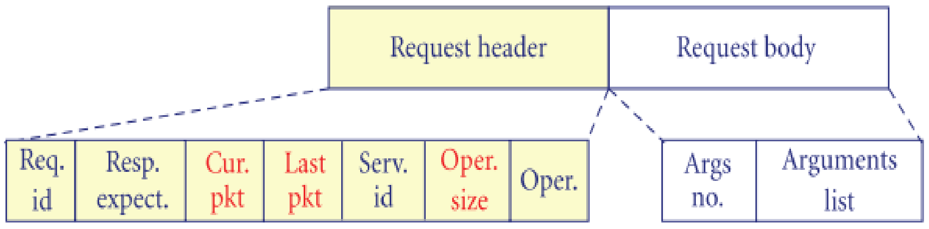


Figure 3.5 WIOP Request message body

- **Reply message body**

Figure 3.6 shows the reply message body, in which the fields in red are for WIOPs version and the rest are for both versions. To indicate the reply address refers to which request, we can use the *Req.id* field. *Reply status* indicates if any exceptions have happened.

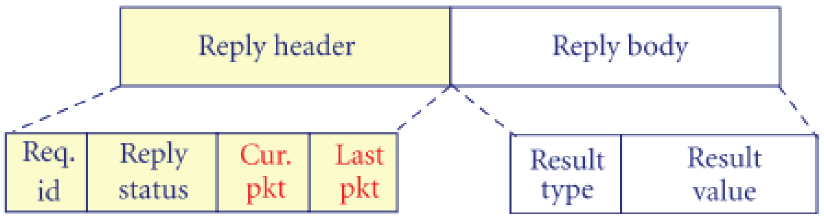
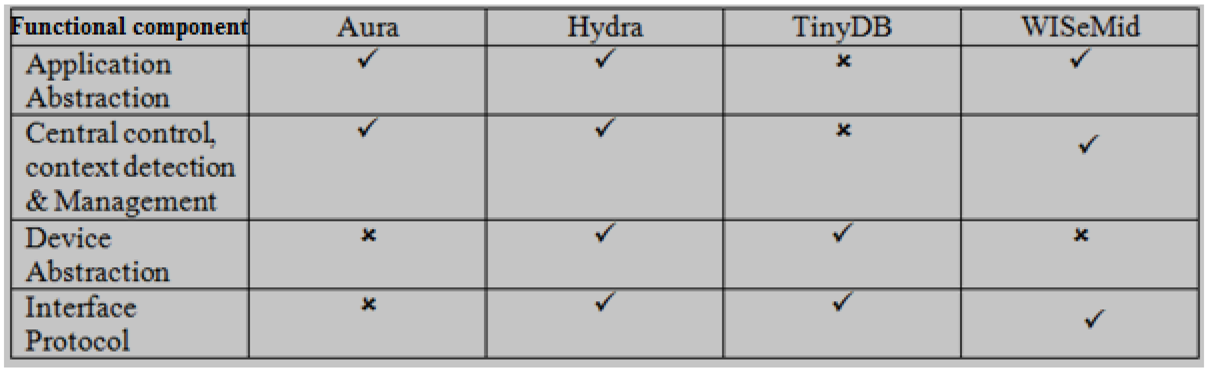


Figure 3.6 WIOP reply message

3.5 Comparative Analysis of the Middleware

In this section, we compare and contrast the four reviewed middleware with each other. Table 3.1 shows the summery of this comparison based on the functional components, which we discussed in Chapter 3.



Functional component	Aura	Hydra	TinyDB	WISeMid
Application Abstraction	✓	✓	✗	✓
Central control, context detection & Management	✓	✓	✗	✓
Device Abstraction	✗	✓	✓	✗
Interface Protocol	✗	✓	✓	✓

Table 3.1 Functional components supported by each middleware

AURA can change its configuration automatically when user tasks or the environment changes. Furthermore, AURA has been designed to provide a platform-independent description of the user’s tasks that allows the user to use different applications in different locations without changing the configuration. However, the technical details to interact with physical devices are out of the scope of the AURA project. Thus, AURA should not support the ease of deployment in terms of abstracting the format of required data to interact with devices.

Hydra makes an abstraction over devices so an end-user does not need to know the detailed information to configure the devices. Moreover, Hydra ease the deployment process by providing an interface to interact with the considered devices as service providers at deployment time. For devices that do not have a sufficient computing power to be a service provider, Hydra uses a proxy that allows these devices to interact with the Hydra middleware through the IP protocol.

Hydra interacts with devices through a SOAP-based API. Yaza. D .et. al [31] believe that RESTful architecture performs better in wireless sensor nodes with limited resources.

Also, based on the research of Guinard. D.et. al [32] RESTful architecture is more intuitive, flexible, and lightweight in compare with the SOAP-based web services. Since in an IoT-based system we interact with many devices with limited computational process capability, we think developing a middleware by using RESTful web service may be more suitable than SOAP-based web service.

TinyDB is defined to be used together with TinyOS, which is a software suite. It is designed to facilitate the access to the lowest level of hardware in an energy efficient way. TinyDB only supports TinyOS-based devices. Therefore, service deploying in TinyDB depends on the operating system that is supported by the required devices. The end-user needs to know the device specifications before working with devices in TinyDB.

WISeMid focuses on integrating the Internet and wireless sensor network at service level by providing transparency of access. Location and technology. By providing these transparencies, this middleware can provide ease of deploying, because we do not need to have the detail information such as address of sensors to deploy a service.

3.6 Conclusion

In this chapter, we reviewed four middleware for IoT-based systems. To satisfy application requirements and provide ease of configuration and deployment for an IoT-based system, middleware requires having a uniform way to communicate with different service providers (e.g. devices). Furthermore, middleware should support device abstraction to provide semantic interoperability between the system parts. In the following we discuss the ease of configuration and deployment of the reviewed middleware.

Chapter 4 Video Monitoring System Requirements

In this chapter, we explain the functional and non-functional requirements of the Video Monitoring System (VMS). We defined these requirements based on the literature and interviews with practitioners. This chapter is structured as follows: Section 4.1 briefly defines a VMS. Section 4.2 describes the methodology used to identify the requirements of the VMS which we have designed and implemented. Section 4.3 explains the VMS functional requirements. Section 4.4 discusses the VMS non-functional requirements.

4.1 VMS High-level Architecture

A VMS communicates through third-party camera service providers with cameras to provide video streams for monitoring a location. A VMS must support the technical details of different types of cameras. However, these variations should be hidden from non-expert end-users. Thus these users should be able to interact with different types of camera in a homogeneous way. Therefore, we developed a VMS that consist of a Middleware for video Monitoring System (MVMS) and applications, which run on top of the middleware to interact with three external entities of the VMS, namely third-party camera service provider, admin-user and guard. VMS is in charge of facilitating the interaction between guards and different types of camera devices. MVMS provides a set of APIs to hide the technical details to interact with different cameras.

We developed a VMS as an IoT-based system because an IoT-based system integrates physical things, such as cameras, into an information network through the Internet to exchange the information that they sense in their environment. VMS handles cameras, which provide video streams from different locations. These cameras are normally connected to the VMS through the Internet.

Our VMS has to interact with third-party camera service providers, which are responsible to interact directly with cameras to retrieve their required information. These camera service providers receive the technical information to configure the cameras, but they hide the detailed information about *how* to communicate with different types of camera. Thus, camera service providers give some general technical details to configure the cameras from the end-users to handle streaming.

Furthermore, our VMS has to interact with two sorts of users, who can issue queries to manipulate the data provided by the cameras or configure them, (I) The user who can only ask to monitor a video-service, called an *guard*. Video-service refers to one or more camera (s) which provides a full video coverage from the required area for the guard. (II) The user who can configure the cameras through a high level service, called an *admin-*

user. Figure 4.1 shows the boundary of the VMS by presenting the VMS internal and its operational environment.

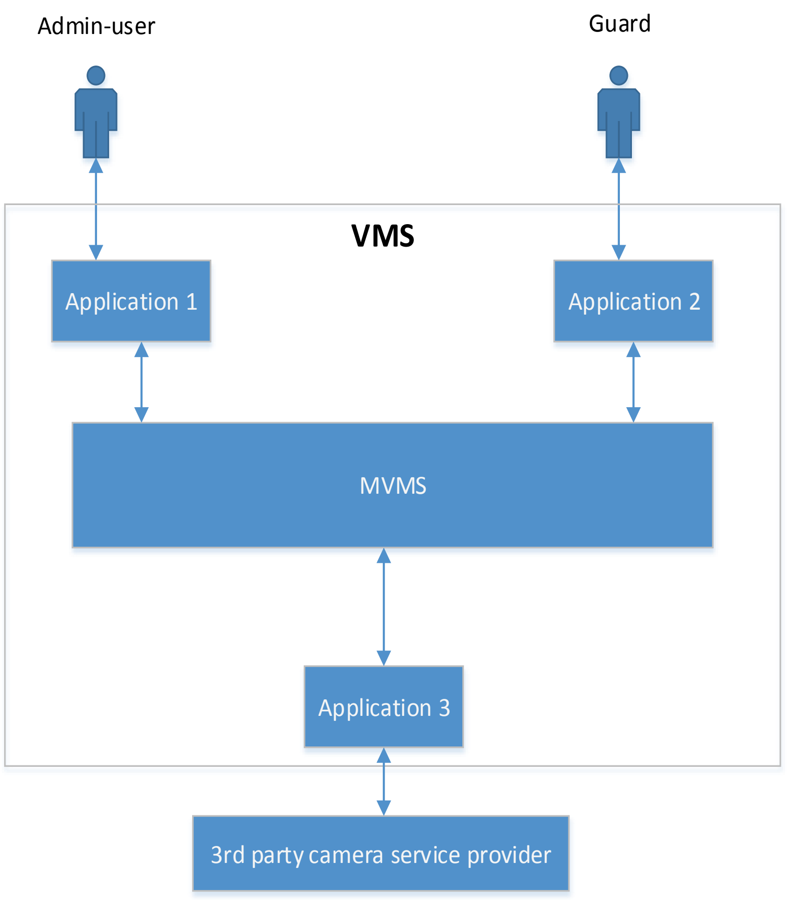


Figure 4.1 High-level architecture of a VMS

The VMS non-functional requirements affected the design of our MVMS. In this chapter, we discuss the requirements which were identified by interviewing of practitioners (mainly functional requirements), and by consulting the literature on IoT-based systems (mainly non-functional requirements).

4.2 Requirements Capturing Approach

Our approach to capture the VMS requirements consists of two parts:

1) Practitioners interviews

To find out the requirements of the admin-user and guard, we had interviews with a number of technical staff in a commercial company (Nedap[4]), who answered our

[4] http://www.nedap.com/

questionnaires on behalf of the admin-users and guards in the system. The company develops a security management platform in order to provide security in different domains, such as airports, companies. One of the services of this platform is *current VMS*, which aims to support security by providing video streams to monitor different locations. Since the security management platform has been used by many companies, it is fair to assume that the technical staff of the company who have developed *current VMS*, have sufficient knowledge about the VMS requirements. Therefore, interviews with the technical staff at Nedap should yield a clear understanding of the VMS functional requirements.

For capturing both functional and non-functional requirements of third-party camera service providers, we interviewed the developers of an interface to handle the interactions with the third-party camera service providers. Furthermore, we reviewed the RESTful API that is provided by a third-party camera service provider to facilitate the interactions with its cameras.

During the types of interviews, we also asked open-ended questions regarding the non-functional requirements, such as, for example, the acceptable application response time. After conducting the interviews, we first described a use case diagram that represents the required VMS functionalities. Then, we provided a sequence diagram for this use case to show how the application service providers, the third-party camera service provider, and the VMS have to interact in order to deploy a video service.

2) Reviewing the related literature

In addition to the our interviews, we reviewed the related work ([13], [33], [34], [1]) on IoT-based middleware to identify more non-functional requirements. Section 4.4 describes some of the non-functional requirements, which fall in the scope of our middleware in an IoT-based system.

4.3 Functional requirements

The VMS functional requirements were identified by interviewing the practitioners in a company, and with respect to our VMS scenarios. We defined a use case and a sequence diagram to represent the VMS functional requirements. VMS has three external entities who communicate with the system: (1) admin-user, (2) guard and (3) third-party camera service provider.

The main functional requirement of the system is to provide the video monitoring service. For this purpose, Figure 4.2 shows the use cases (1) monitoring a video service, (2) configuring the camera, (3) deploying a camera configuration, (3) reporting about the resources (4) configuring the video service. To provide each functionality, several external entities should interact with each other. The functionalities and the involved external entities are:

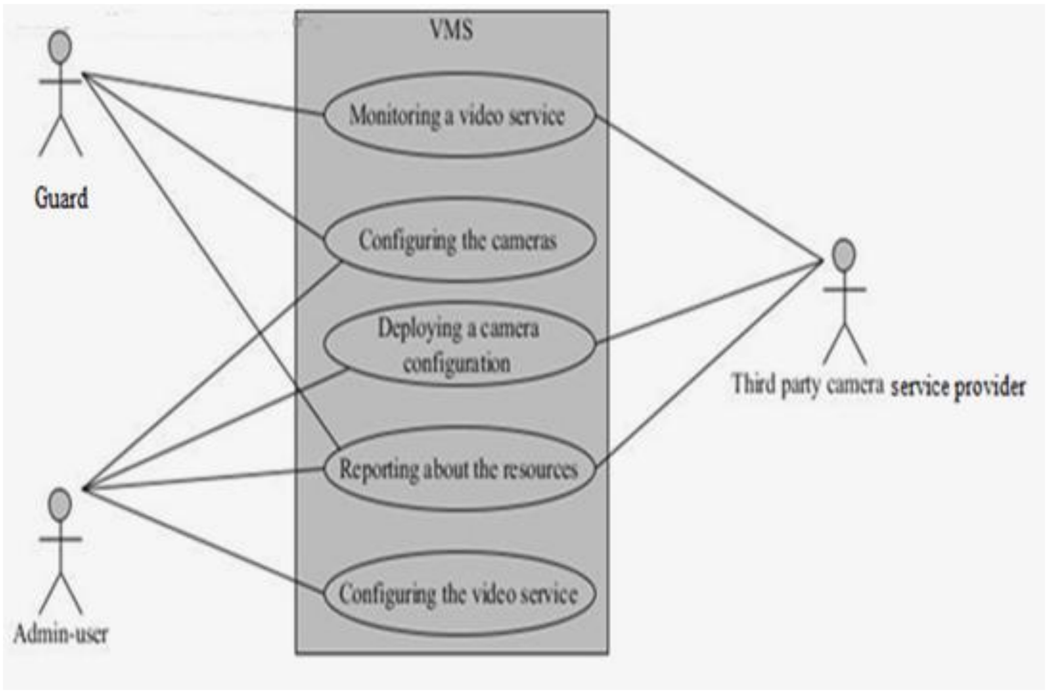

VMS
Guard
Monitoring a video service
Configuring the cameras
Deploying a camera configuration
Reporting about the resources
Configuring the video service
Admin-user
Third party camera service provider

Figure 4.2 VMS functionalities and involved external entities

1. Monitoring a video service

   This use case defines as a service that can be asked by guard. A guard sends a monitoring request to monitor a video service. VMS extracts the address of the third-party camera service providers that manage the cameras in the video service. Then, VMS sends the monitoring request and the address of the guard to the considered third-party camera service providers. Finally, the third-party camera service provider starts up the requested video stream to the guard. Later, the guard can send a request to the third-party camera service provider through the VMS to stop the video stream.

2. Configuring the camera

   This use case addresses the required camera configurations. Our VMS supports the capability of saving more than one configuration for each camera. Only one of these configurations can be applied on the cameras. Each configuration address the required camera configuration fields to prepare a camera for probing. In fact, each set of configuration identifies a state of a camera configuration that is desirable for a guard.

3. Deploying a camera configuration

   An admin-user can ask VMS to deploy the defined settings through a GUI. VMS extracts the camera configuration from VMS data-base and then sends the extracted data to the third-party camera service provider to apply the new configuration on the camera.

4. Reporting

   VMS is in charge of delivering reports with information on the state of cameras or third-party camera service providers. Both admin-users and guards are able to ask for these reports, which are provided by third-party camera service providers. However, to decrease the number of interactions with third-party camera service providers, VMS should have the capability of cashing the reported information. In this way, if VMS receives the same request more than once in a specific period of time and the reported information is still up to date, VMS can respond without an additional interaction with the third-party camera service provider.

5. Configuring video service
   This use case addresses the required configurations that are required to set a collection of one or many camera(s) that can provide the full video stream coverage for guards to monitor different places.

   Figure 4.3 shows how the external entities interact with VMS according to the five use cases described above.

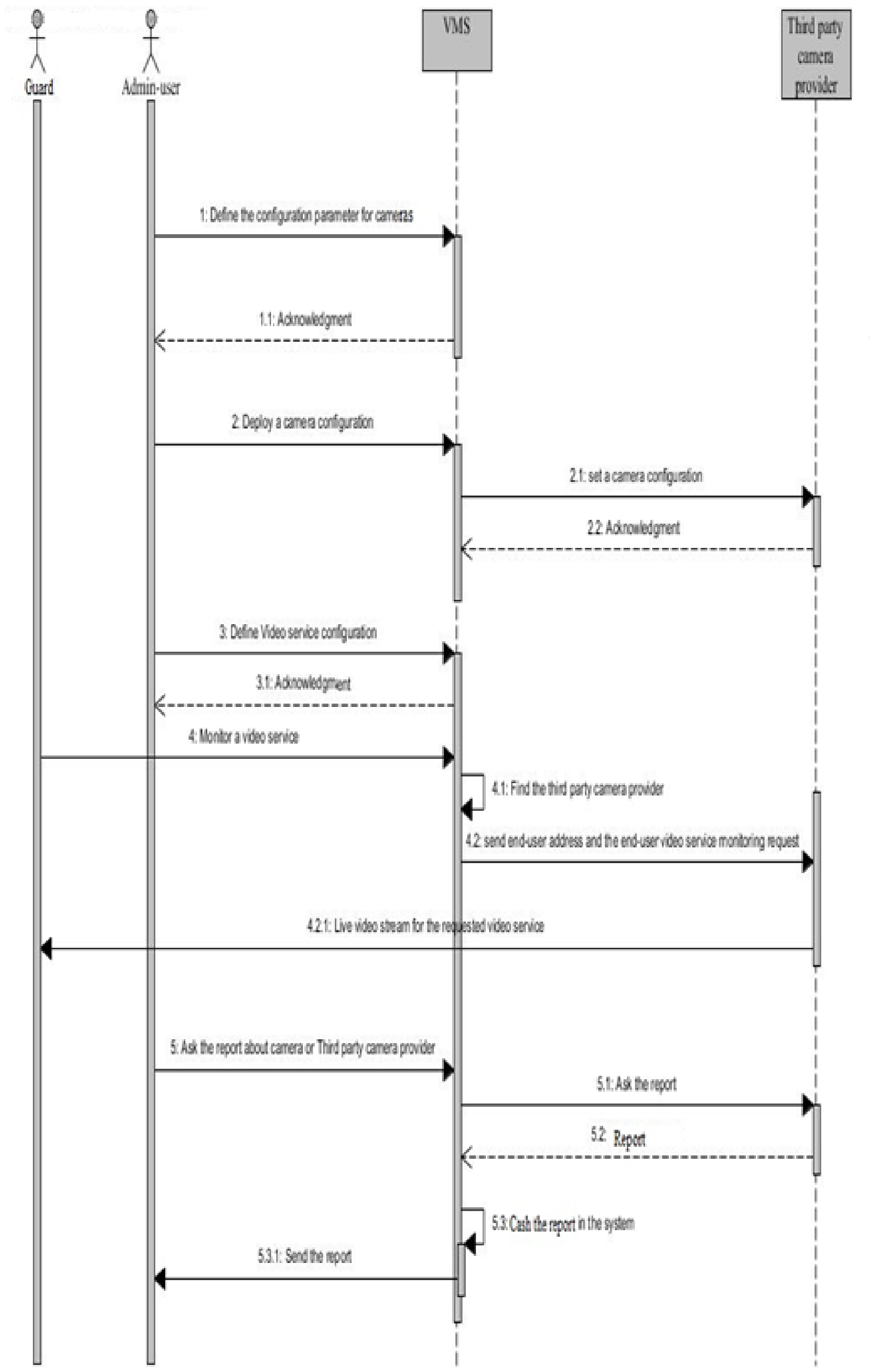

Guard
Admin-user
VMS
Third party camera provider
1: Define the configuration parameter for cameras
1.1: Acknowledgment
2: Deploy a camera configuration
2.1: set a camera configuration
2.2: Acknowledgment
3: Define Video service configuration
3.1: Acknowledgment
4: Monitor a video service
4.1: Find the third party camera provider
4.2: send end-user address and the end-user video service monitoring request
4.2.1: Live video stream for the requested video service
5: Ask the report about camera or Third party camera provider
5.1: Ask the report
5.2: Report
5.3: Cash the report in the system
5.3.1: Send the report

Figure 4.3 Interactions with external entities to provide a Video monitoring application

4.4 Non-functional requirements

The VMS non-functional requirements that we have identified in our interviews have also been addressed in the literature. We identified the following non-functional requirements for the VMS:

.

1. Ease of configuration for admin-user [10] [2]

   VMS should be able to connect to different types of cameras, which each can support specific communication protocols and standards. However, to facilitate the configuration of different types of camera, VMS should support a single GUI interface for admin-users to work with different cameras.

2. Facilitate monitoring[2]

   VMS should facilitate monitoring by providing both configuration and location transparency for guard. For example, the guard can refer to a location that has to be monitored by using its video service name. For instance, to monitor *Hall A*, which is covered by camera 1 and camera 2, guard only needs to send the name of the location (*Hall A*). VMS is in charge of finding the address and the configuration of the required cameras, which either has been set by an admin-user or already, has a default value in the system.

3. Supporting new types of cameras [13]

   VMS has to be able to support different types of camera in different domains. Thus, VMS has to be extendable with minimum changes, and it also needs to allow new cameras to be added to the system.

4. Scalability [10], [13], [2], [35]

   The security management platform has to be capable of supporting different numbers of camera. The company VMS as a main part of the security platform is in charge of supporting a large number (more than 1000) of cameras. To support a large scale system, VMS has to be able to integrate with the third-party camera service provider to distribute part of the necessary process.

5. Security and privacy

The major security problem of IoT is related to authentication and data integration [34]. To do the authentication, we need data exchange between authentication servers and devices. This makes a problem when an application use passive RFID tags in an IoT-based system, because a passive RFID does not have the capability of handling many communications with an authentication server. This problem has not been solved yet [36].

4.5 Conclusion

In this chapter, we identified both functional and non-functional requirements that should be considered in the design and implementation of a VMS. In order to identify these requirements, we conducted interviews with the technical staff of a commercial company (Nedap). We interviewed the technical staff of the research and development department in order to have a general understanding about both the current and next generation of VMS. We also had interviews with the software architects and developers of the current VMS to collect more information of the system.

Furthermore, we reviewed the literature on middleware for IoT-based systems to identify those requirements that a VMS should be able to support.

In our project, we are going to answer to two of these non-functional requirements: (1) providing ease of configuration for admin-user; (2) providing ease of monitoring for a guard who is as an end-user in VMS.

Chapter 5 Proposed architecture

In this chapter, first we explain an overview of the Video Monitoring System (VMS). After that, we explain the required parts of VMS in more detail.

5.1 Overview of the system

In this section, we explain resources of a VMS and provide an overview about its main parts.

In order to provide ease of configuration and deployment for both guards and admin-users, VMS provides APIs to facilitate client application's interaction with camera devices. Admin-users can select and configure the required camera devices to monitor a video service, for example, a specific location without considering the detailed camera specifications (e.g., supported communication protocols). VMS is also responsible to provide location transparency for guards and help them to do video monitoring without knowing the camera device's locations.

In a VMS, we have three resources which are accessed by both guards and admin-users of the system as follows:

1) Camera service: This resource is configured by admin-users to set the sampling data rate of a camera with respect to its specification.

2) Third-party camera service provider: This resource is provided by third-party camera service providers and can be configured by admin-users. This resource manipulates information about the cameras service provider such as, different reports about the cameras of a specific camera service provider or the available cameras in a third-party service provider. A third-party camera service provider manages one or more camera device(s) by applying the camera service specifications.

3) Video service: This resource refers to one or more camera service(s) which are used by third-party camera service providers to monitor, for example, a specific location. This resource is configured by admin-users and can be accessed by guards. A video service represents are shown in windows on the screen with several video streaming sub-windows.

   Figure 5.1 shows these resources used by a video monitoring application for a guard. For instance, the guard asks to monitor a shopping area. The admin-user defines the shop monitoring video service and the required configuration for the camera services in VMS. Then, VMS sends the camera service request to the considered third-party camera service providers. Finally the third-parties send the requested video stream to the guard.

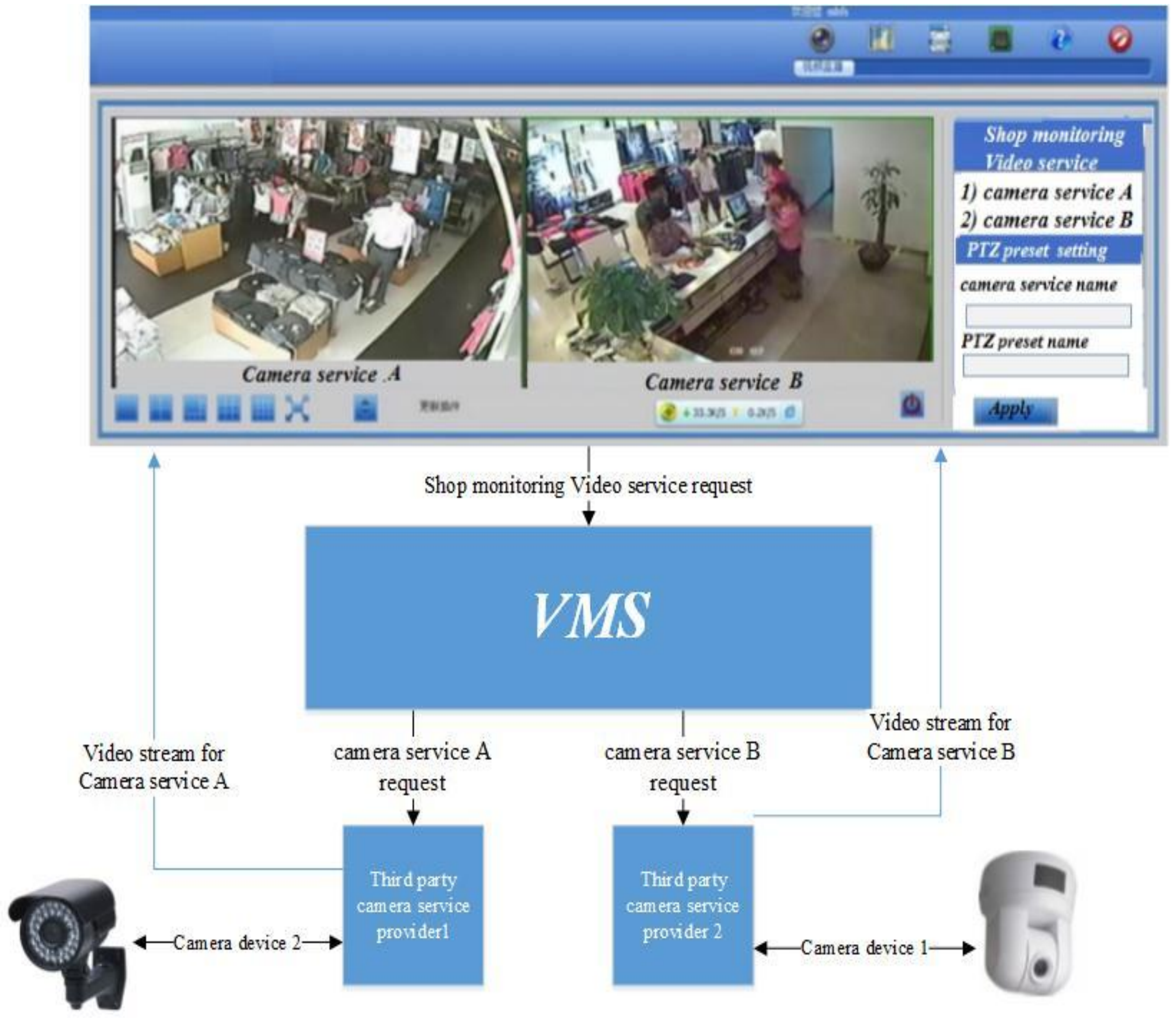


Figure 5.1 Example of a video service monitoring.

Figure 5.2 is the overview of the VMS system to show the main components which are required to interact with the three aforementioned resources.

VMS consists of two main parts: (1) Client Application that receives the end-users request through a GUI. Then, it sends the request in an appropriate format to third-party camera service providers. (2) MVMS that consists of three main components to answer the client application request.

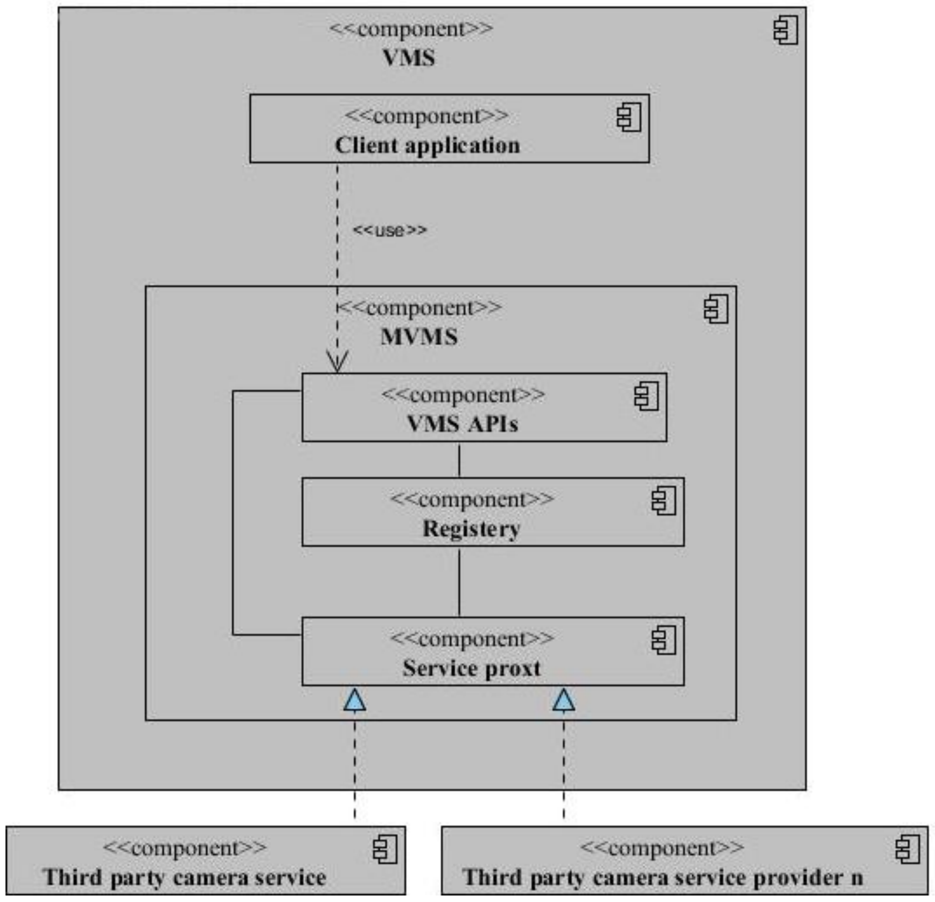


Figure 5.2 Abstract view of VMS

We will explain the MVMS components, in the following:

- APIs: We listed the possible operations that a client application can expect from VMS. We extracted these operation based on the requirements of a company that uses VMS .
  The expected operations from MVMS can be categorized in four main groups of APIs, namely Configuring, Deploying, Reporting and Monitoring. These APIs are placed between the client application and third-party camera service providers application. Thus, these APIs are seen as a software intermediary component that enables the client applications to interact with different third-parties camera devices. These API will be elaborated more in section 5.2

- Registry: This component is responsible to store the data about the three resources in the VMS. Registry provides the required data for APIs (i.e., the configuration data for a camera) to do the requested operations by the client application. Furthermore, to have a list of supported operations by the VMS, this component stores the supported operations, their input and output data. we will explain it in more detail in section 5.3.

- Service proxy: A VMS is responsible to interact with different Third-party camera service providers. Therefore, a service proxy is required that is composed of several adaptors to shield the heterogeneity of the camera service providers. Each adaptor is responsible to make the interaction with a third-party camera service provider in an acceptable format for both VMS APIs and that third-party camera service provider. we will explain it in more detail in section 5.3

As figure 5.2 shows, MVMS communicates with deferent third-party camera service providers. Each third-party camera service provider is in charge of (1) deploying the configuration to the camera devices, and (2) sending back a video stream to the requesting guard application which is out of the scope of this project. In this project, we only work on the first responsibility of third-party camera service provider.

5.2 APIs

As figure 5.3 shows VMS supports four APIs namely monitoring, configuring, deploying and reporting. In the following, we will discuss them in more detail:

- The *configuring* API allows admin-users to save the resource configuration data in the *registry*. An admin-user based on the needs of a guard should be able to change the camera configuration of a video service. Therefore, the repository can save different configuration presets for each camera service. In this case, an admin-user can use the configuration presets name that have already saved in the registry. Thus, there are no needs to redefine all the configuration details. Furthermore, This API allows admin-users to define the required camera services in video services.
- The *Deploying API* handles deploying the requested configuration. Thus, to perform this API, an admin-user gives the camera service configuration name, which has already saved in the VMS. Then, VMS finds the related configuration information that has been saved in them in the registry, and sends them to the service proxy to send to the consider third-party camera service provider.
- The *reporting* API allows admin-users or guards to have access to the possible reports that VMS can provide, for instance, a report about a list of the available cameras that are available by a specific third-party camera service provider.
- The *Monitoring* API gets the name of the video service, which a guard would like to monitor as an input. Then, VMS shows the list of the required cameras service and their PTZ presets. After that MVMS extracts the required data to interact with the third-party camera service provider. Finally, the extracted data is sent to the

service proxy, which is able to send it to the consider third-party camera service provider. This API also allows an admin-user to set different PTZ preset for each camera service, but only one of them can be deployed at the time.

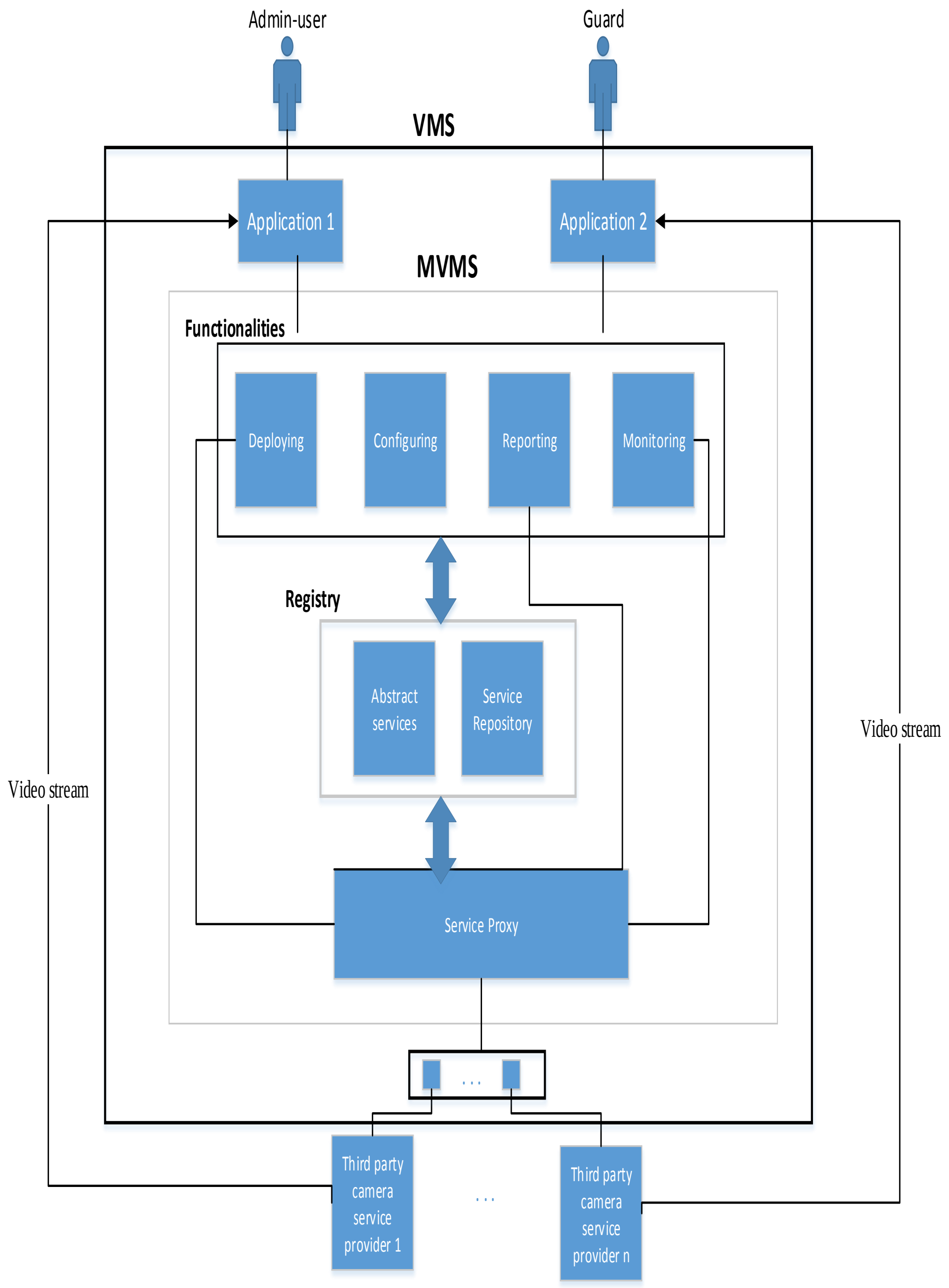


Figure 5.3 Overview of the VMS system including the applications and MVMS

5.3 Registry

Registry stores the required data to perform the MVMS APIs. This part is divided into two sub-parts: (1*) resource configuration* that contains the required specification to configure and define the resources that the both end-users would like to interact with (2) *service repository* that contains services that are required to perform the APIs. We will discuss each sub-part in more detail, in the following.

5.3.1 Resource configuration

Resource configuration is responsible to provide the required data to define and configure the three resources. Each *Video session* configuration refers to the required camera(s) to provide full camera coverage to monitor, for instance, a location. Each *camera* configuration defines the required data to configure *a camera service* to provide a desirable video service for the guard. The *third-party camera service provider* can be configured by admin-user or the third-party camera service provider. These resources will be explained more in the section 6.2 Resource definitions.

The data about the resources is mostly defined by admin-users. We can have different types of specifications in order to configure the resources. For instance, a resource like camera service can be defined on different types of camera devices. However, different types of camera devices can have their own specifications to be configured. An admin-user only needs to set some specifications that are required to perform MVMS APIs. These specifications can be common in most of the cameras devices. Therefore, to facilitate configuration and definition of resources in VMS, we can extract a structural model from the required specifications to define camera service resources.

To extract this structural model (as shown in Table 5.1) we reviewed three references: (1) Pervasive System (PS) group[5]of Twente university research that has proposed a generic WSN data model to interact with sensors in a WSN (2) The RESTful API that is going to be used at Nedap to integrate the current video monitoring system with the third-party camera service providers, (3) The ONVIF [6]group specifications [37] that defines camera attributes to develop a middleware framework between a client and different IP-based camera devices.

[5] http://ps.ewi.utwente.nl/

[6] http://www.onvif.org

Service Name	Attribute Name	Description
Video session	Id	A unique identification key for each video service.
	Location	The location name that can be monitored by a video service
Camera	Camera name	
	Camera ID	
	Camera descriptions	
	Camera playback control URL	
	Camera codec	
	Camera streaming protocol	A camera can support HTTP or RTP protocols for streaming
	Camera Pan- Tilt- Zoom	In case of supporting PTZ, these fields are required
	Camera brightness rate	
	Camera type	
	Camera Time zone	
	Camera memory (status, size, read time..)	In case of having built in memory in the camera these fields should be defined.
	Camera power battery (maximum capacity, recharging rate)	In case of using battery as the power supplier, these fields should be defined..
	Camera power harvesting (ID, description, energy source, storage capacity)	In case of using power supplier except of battery these fields should be defined.
Third-party camera service provider	camera service provider name	
	camera service provider ID	
	Camera service provider descriptions	
	Camera service provider URL	
	Camera codec	
	Camera service provider can show recording	
	Camera service provider can get recording details	
	Bandwidth	
	Camera service provider streaming protocol	A camera can support HTTP or RTP protocols for streaming
	Camera Time zone	
	Camera service provider can record	

Table 5.1. List of the specifications of resources in VMS

We made a data model out of these specifications as shown in Figure 5.4 As it shows each *Video service* can refer to one or more *camera service(s)*. Also each *camera* service can belong to one or more *video service*(s). Each *camera* can be supported by only one *camera providers*. While, a camera provider can support more than one *camera*.

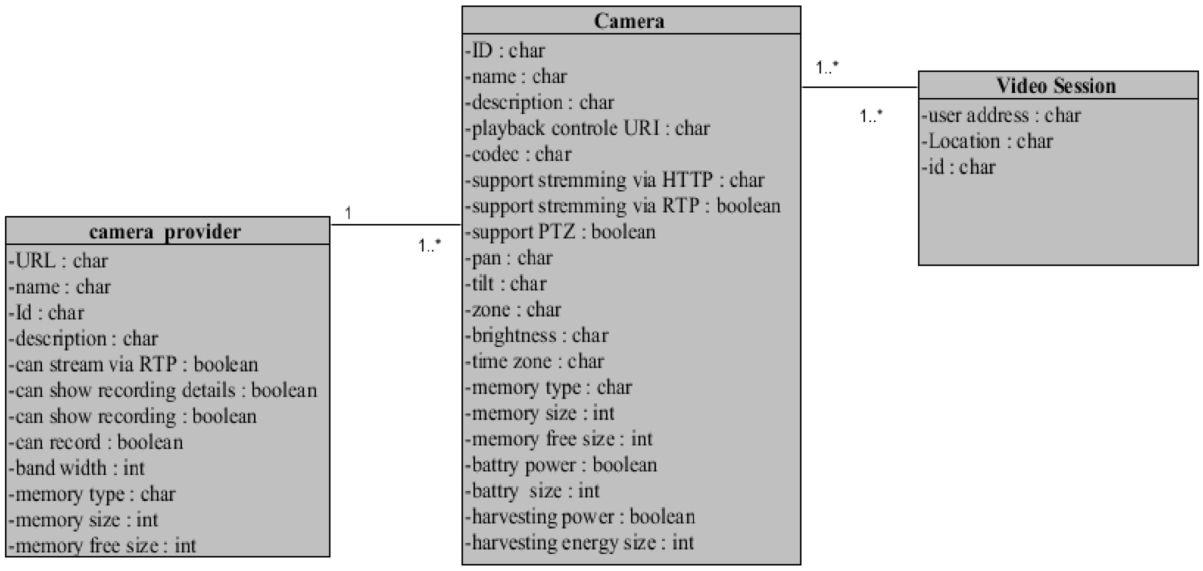
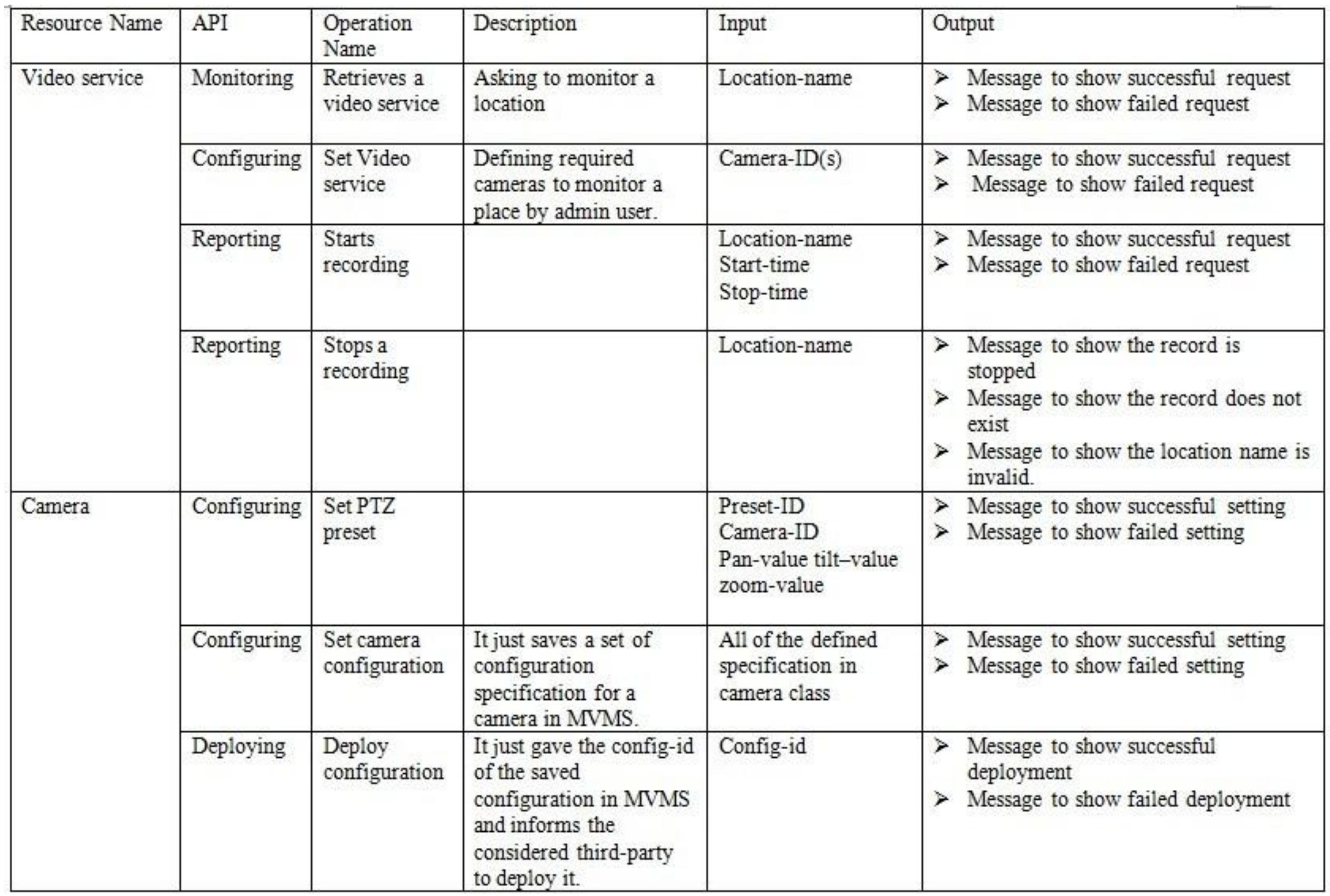


Figure 5.4 Data model to define resources in VMS

It should be mentioned that we try to make a generic data model that contains most of the required specifications that resources should have in different video monitoring systems. It means, we can have some video monitoring systems that use parts of this data model based on the end-users' purpose. For instance, we do not need to use the battery or harvesting power specification when the energy consumption is not an important issue in the system. Thus, to make a data model for a video monitoring system, we can instantiate parts of this specification.

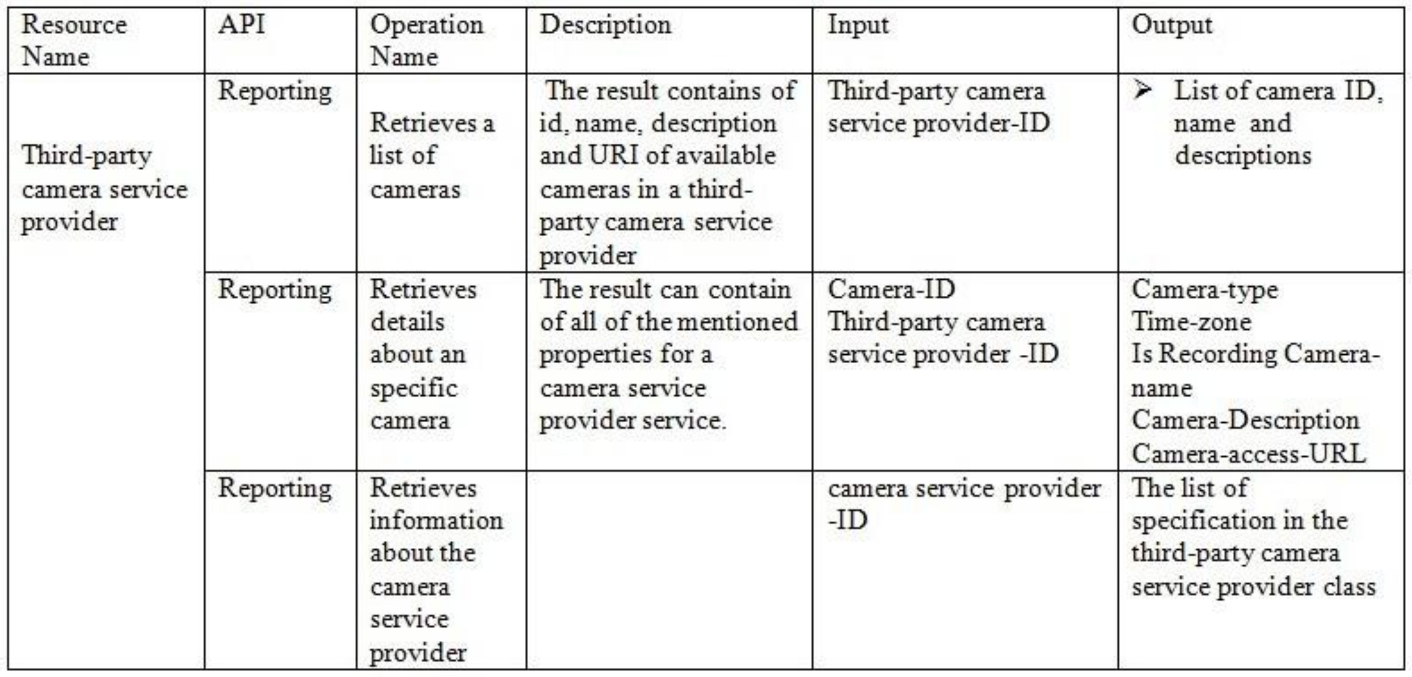
5.3.2 Services repository

Service repository refers to the sub-functionalities that can be provided by a third-party camera service provider. Each of the deploying, configuring, reporting and monitoring APIs is a composition of one or more of these sub-functionalities, which are shown in Table 5.3:



Resource Name	API	Operation Name	Description	Input	Output
Video service	Monitoring	Retrieves a video service	Asking to monitor a location	Location-name	➢ Message to show successful request ➢ Message to show failed request
	Configuring	Set Video service	Defining required cameras to monitor a place by admin user.	Camera-ID(s)	➢ Message to show successful request ➢ Message to show failed request
	Reporting	Starts recording		Location-name Start-time Stop-time	➢ Message to show successful request ➢ Message to show failed request
	Reporting	Stops a recording		Location-name	➢ Message to show the record is stopped ➢ Message to show the record does not exist ➢ Message to show the location name is invalid.
Camera	Configuring	Set PTZ preset		Preset-ID Camera-ID Pan-value tilt–value zoom-value	➢ Message to show successful setting ➢ Message to show failed setting
	Configuring	Set camera configuration	It just saves a set of configuration specification for a camera in MVMS.	All of the defined specification in camera class	➢ Message to show successful setting ➢ Message to show failed setting
	Deploying	Deploy configuration	It just gave the config-id of the saved configuration in MVMS and informs the considered third-party to deploy it.	Config-id	➢ Message to show successful deployment ➢ Message to show failed deployment

Table 5.3 Sub-functionalities of the APIs: Part 1



Resource Name	API	Operation Name	Description	Input	Output
Third-party camera service provider	Reporting	Retrieves a list of cameras	The result contains of id, name, description and URI of available cameras in a third-party camera service provider	Third-party camera service provider-ID	➢ List of camera ID, name and descriptions
	Reporting	Retrieves details about an specific camera	The result can contain of all of the mentioned properties for a camera service provider service.	Camera-ID Third-party camera service provider -ID	Camera-type Time-zone Is Recording Camera-name Camera-Description Camera-access-URL
	Reporting	Retrieves information about the camera service provider		camera service provider -ID	The list of specification in the third-party camera service provider class

Table 5.3 Sub-functionalities of the APIs: Part 2

5.3 Service proxy

Service proxy is in charge of making query in an appropriate format, which can be supported by the third-party camera service provider, to interact with the third-parties. For instance, a VMS can work with two third-party camera service providers that support different communication protocols (e.g., one of them SOAP-based communication and the other one supports RESTful). Therefore, Service proxy needs to have different components to adopt the required data to send to the third-party camera service provider in an acceptable format that is supported by them.

Furthermore, the service *proxy* is responsible to change the third-party camera service providers' response in an understandable format for MVMS. In the MVMS the video streaming data is sent directly to guards and only the data for reporting API is sent through VMS.

5.4 Conclusion

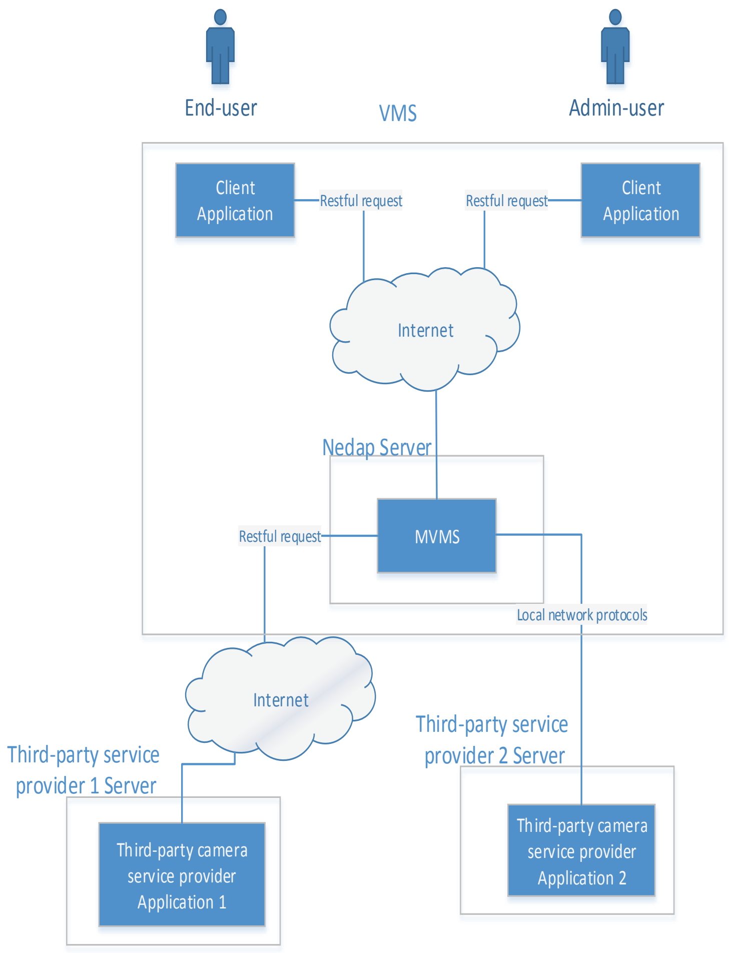
In this chapter, we defined a general overview of the VMS parts. Generally, we can summarize VMS in three parts (1) APIs (2) repository that stores the required data to perform the functionalities (3) service proxy that handles the interaction with third-parties camera service providers.

Chapter 6 Implementation

In this chapter, we explain the implementation of our prototype. Section 5.1 explains the deployment architecture of internal and external components of VMS. Section 5.2 explains the client application and the way that the APIs have been used in the client applications. Section 5.3 shows the database schema of the Registry. Finally, section 6.4, explains the implementation of service proxy complement.

6.1 Deployment

Figure 6.1 shows how internal and external components of a VMS have been deployed and the communication protocols which are used by them. As the Figure shows, client applications send a RESTful request to the server that MVMS is hosted on. Then, MVMS interacts through the communication protocols that can be used in either LAN or Internet to interact with third-party service providers. Finally, third-party can send the required information to answer the requesting query to the client application or MVMS.



End-user
VMS
Admin-user
Client Application
Restful request
Restful request
Client Application
Internet
Nedap Server
Restful request
MVMS
Local network protocols
Internet
Third-party service provider 2 Server
Third-party service provider 1 Server
Third-party camera service provider Application 1
Third-party camera service provider Application 2

Figure 6.1 the deployment architecture

A Camera service provider is located inside Nedap Company that deploys the VMS. The third-party camera service provider uses some LAN protocol such as AMQP (Advance message queuing protocols), which seems more light weight than HTTP, to interact with MVMS.

6.2 Client Application

We used web service to implement the interaction between the client applications and MVMS. Web service can be implemented based on two architectures: (1) SOAP-based and (2) RESTful. we decided to use REST for two reasons: (1) VMS does not have complex operations and (2) REST is easy to use for clients [32]

The client application supports two access levels, namely (1) end-user and (2) admin-user. In our prototype the end-user client application uses monitoring API and also move the camera in the form of changing its PTZ preset. The admin-user client application uses configuring and deploying APIs to configure the cameras for endd-users. We explain two types of client application as follows:

End-user client application:

As the monitoring-form (see Figure 6.2) shows, first an end-user enters a name of a video service. MVMS by receiving the monitoring request shows the list of the required cameras of the requested video service to the end-user. Also, MVMS extracts the name of the required third-party camera service providers and ask them to send the video stream to the requested user. Beside this, if end-user needs to change the one of the cameras PTZ, he/she can see the list of available PTZ preset for the camera and then apply one of the listed PTZ preset on them. By receiving the new PTZ preset MVMS updates the registry information and informs the considered third-party camera service provider about the changes.

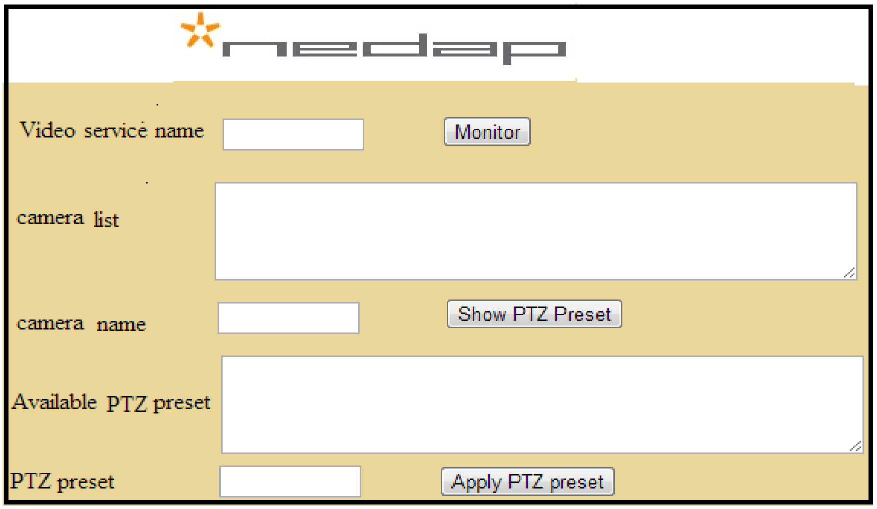


Figure 6.2 End-user monitoring form

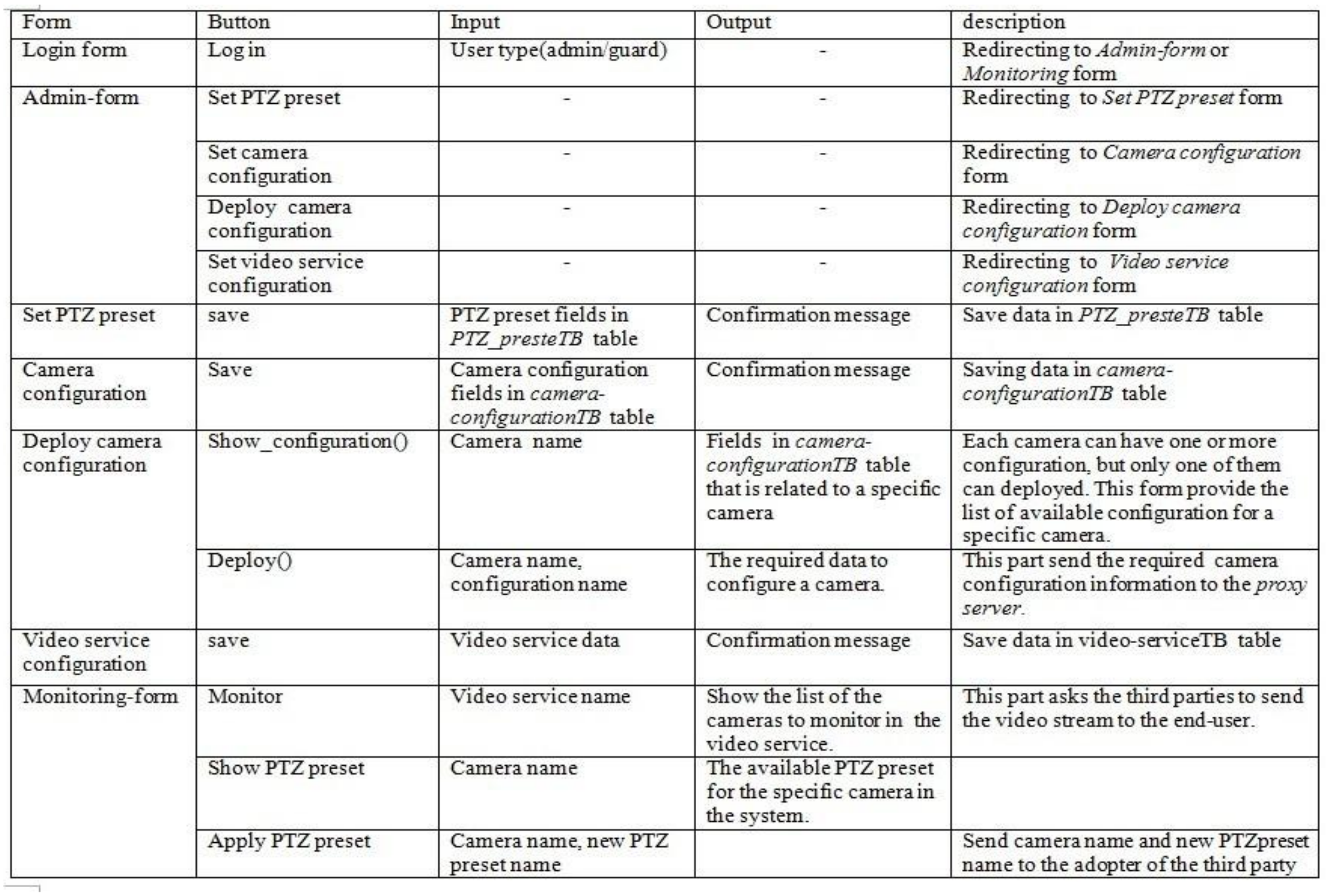
Admin-user client application:

In our prototype, an admin-user client application provides access to Configuring and Deploying APIs through admin-form. In order to set and apply the configurations of cameras and video services, admin user has the following forms:

- **Set PTZ preset form**: Admin-user defines the rates for Pan, Tilt and Zoom and save it in the system through this form as a PTZ preset. Later this PTZ preset is used by end-user or admin-user.
- **Set camera configuration form**: Admin-user defines one or more configuration attributes for a camera and records it in the system through this form. Each camera works based on only one configuration that has been deployed in the system. The capability of saving more than one configuration in the system let admin-user to make different sets of configuration attributes over the most requested configurations. In this case, if the configuration set has already saved in the system the admin-user does not need to redefine it.

- **Deploy camera configuration form**: An admin-user sends one of the required camera configuration, which has been saved in the MVMS, to the responsible third-party camera service provider through this form.

- **Set video service configuration form**: Each video service refers to one or more camera(s). Through this form, we make a group of cameras that can provide a full video stream to satisfy the end-user monitoring request.
  Table 6.1 show more detail about the client application’s forms



Form	Button	Input	Output	description
Login form	Log in	User type(admin/guard)	-	Redirecting to *Admin-form* or *Monitoring* form
Admin-form	Set PTZ preset	-	-	Redirecting to *Set PTZ preset* form
	Set camera configuration	-	-	Redirecting to *Camera configuration* form
	Deploy camera configuration	-	-	Redirecting to *Deploy camera configuration* form
	Set video service configuration	-	-	Redirecting to *Video service configuration* form
Set PTZ preset	save	PTZ preset fields in *PTZ_presteTB* table	Confirmation message	Save data in *PTZ_presteTB* table
Camera configuration	Save	Camera configuration fields in *camera-configurationTB* table	Confirmation message	Saving data in *camera-configurationTB* table
Deploy camera configuration	Show_configuration()	Camera name	Fields in *camera-configurationTB* table that is related to a specific camera	Each camera can have one or more configuration, but only one of them can deployed. This form provide the list of available configuration for a specific camera.
	Deploy()	Camera name, configuration name	The required data to configure a camera.	This part send the required camera configuration information to the *proxy server*.
Video service configuration	save	Video service data	Confirmation message	Save data in video-serviceTB table
Monitoring-form	Monitor	Video service name	Show the list of the cameras to monitor in the video service.	This part asks the third parties to send the video stream to the end-user.
	Show PTZ preset	Camera name	The available PTZ preset for the specific camera in the system.	
	Apply PTZ preset	Camera name, new PTZ preset name		Send camera name and new PTZpreset name to the adopter of the third party

Table 6.1 the summary of our prototype forms

6.3 Registry

Registry stores the data for two purposes (1) for service *repository* and (2) to record *resource configuration* information. Since, implementing the *resource configuration* part can show if the system meets the goal of the system, providing ease of configuration and deployment for non-expert users. We only implement the *resource configuration* part of the *registry*.

To provide the required information for the APIs, MVMS needs to have the resource *configuration* component that records the resources information in the database. As discussed in chapter 5, VMS has three resources namely Video *service*, *Camera service* and *Third-party camera service provider,* which is recorded in three separate tables. Each *camera service* can have one or more configuration(s) and PTZpreset(s), which can be recorded in two different tables. Since, it is possible that only the PTZ preset in a camera configuration be changed by an admin-user or end-user. We made a separate table to save the PTZ preset information. Furthermore, since a camera can be assigned to more than one video service and a video service can refers to more than one camera service, we have an intermediary table between *Video service* and *Camera service* namely serviceTB table.

Our assumption is the *third-party-camera-service-providerTB* and *cameraTB* are initialized by the third-party camera service providers. We try to make the required logics of the VMS by using Primary Key (PK) and Foreign Key (FK) relation between the tables. For instance, admin-user can save different PTZ preset in the system that be used in the camera configuration. Since each camera configuration can have only one PTZ preset the PTZ-preset-name is as the foreign key in camera-configuration-TB.

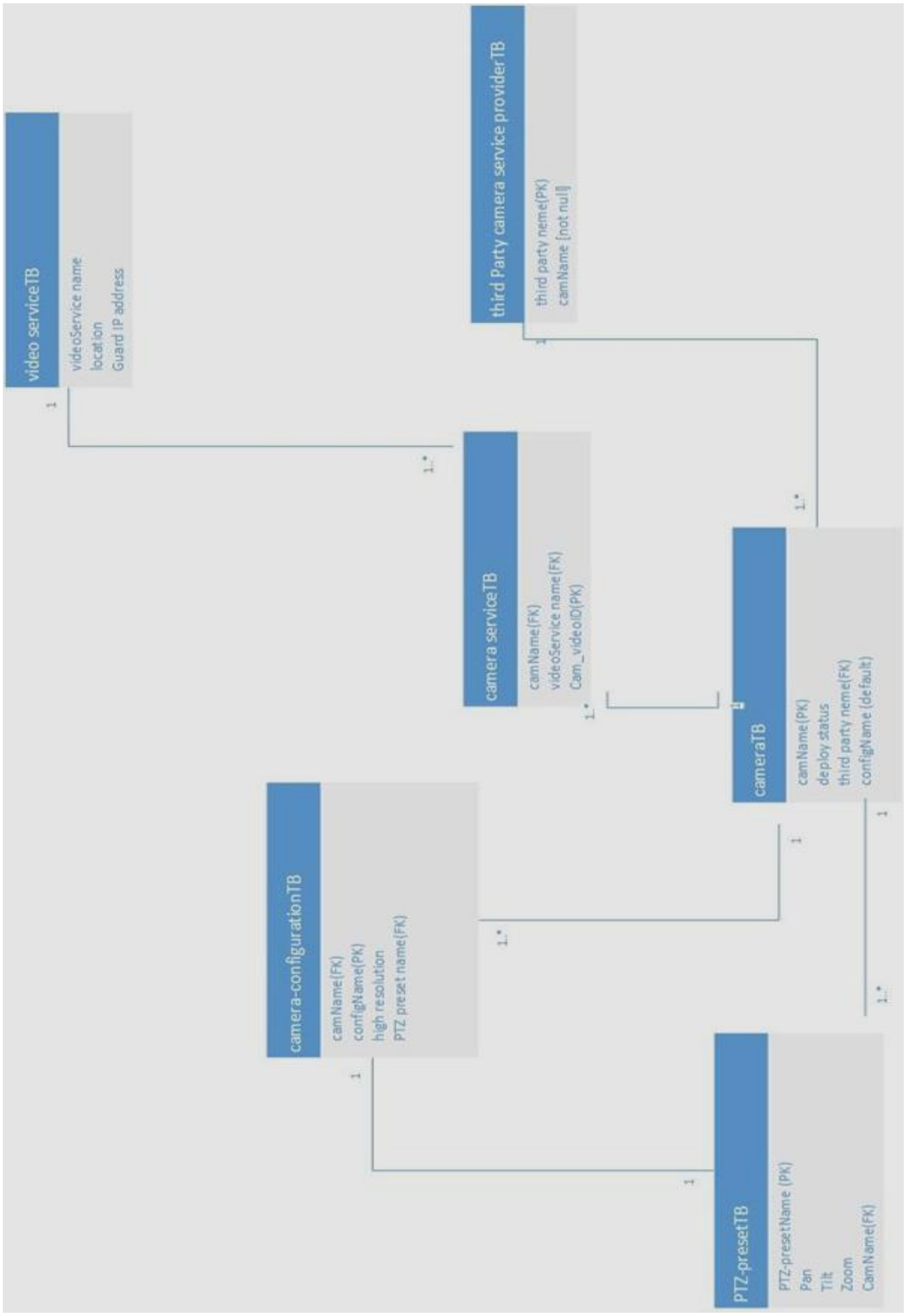


Figure 6.3 Data base schema

6.4 Service proxy

In order to provide the required interaction between MVMS and the third party camera service providers in appropriate formats, we need to call the service proxy component for three operations in the system (1) to deploy the configuration, (2) modify the PTZ preset value by end-user or admin-user, and (3) to answer the monitoring request by the end-user. Figure 6.4 shows how these operations implemented in our prototype.

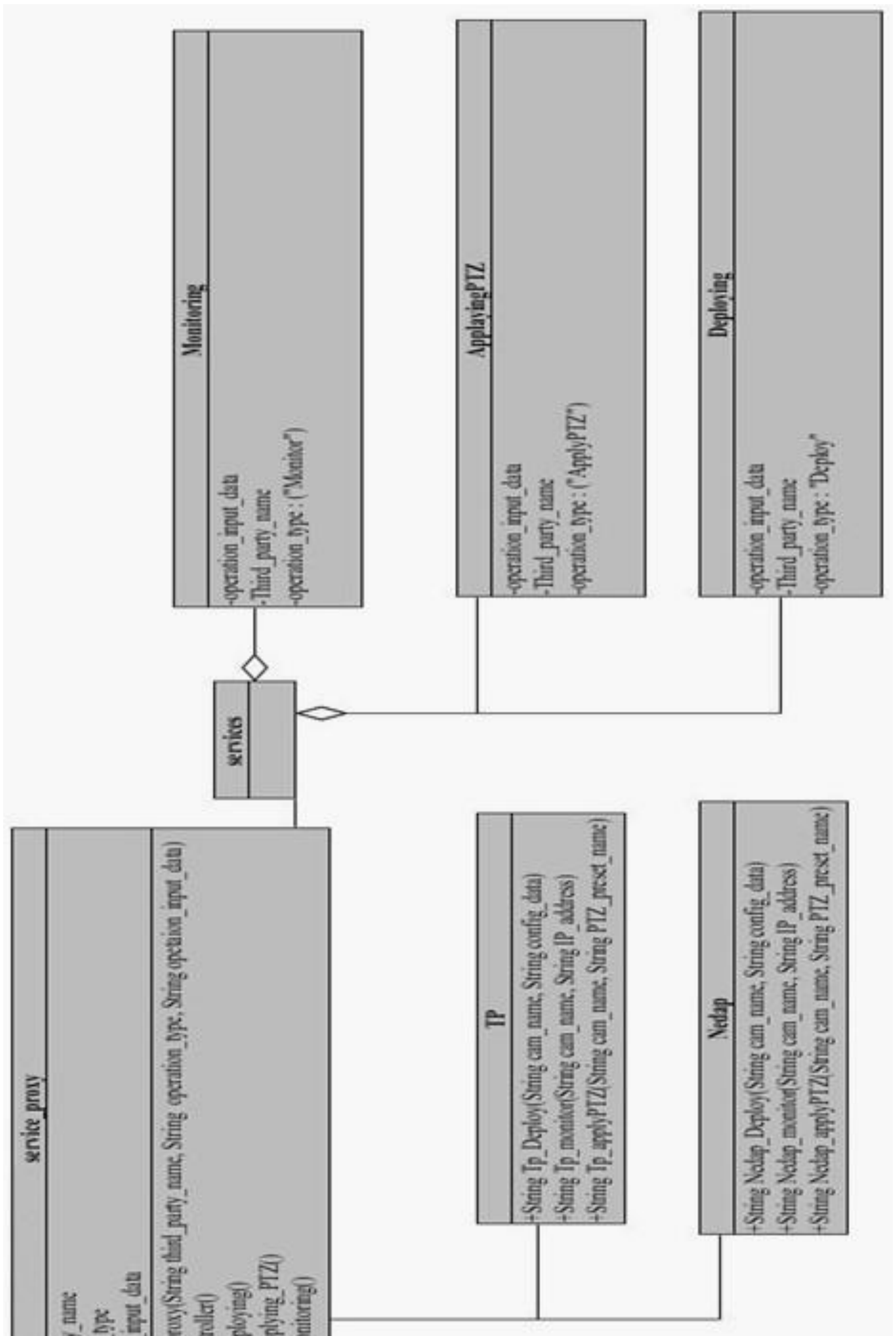

Monitoring
-operation_input_data
-Third_party_name
-operation_type : ("Monitor")
ApplayingPTZ
-operation_input_data
-Third_party_name
-operation_type : ("ApplyPTZ")
Deploying
-operation_input_data
-Third_party_name
-operation_type : "Deploy"
services
service_proxy
-third_party_name
-operation_type
-operation_input_data
+Service_proxy(String third_party_name, String operation_type, String operation_input_data)
+Void controller()
+String Deploying()
+String Applying_PTZ()
+String Monitoring()
TP
+String Tp_Deploy(String cam_name, String config_data)
+String Tp_monitor(String cam_name, String IP_address)
+String Tp_applyPTZ(String cam_name, String PTZ_preset_name)
Nedap
+String Nedap_Deploy(String cam_name, String config_data)
+String Nedap_monitor(String cam_name, String IP_address)
+String Nedap_applyPTZ(String cam_name, String PTZ_preset_name)

Figure 6.4 the relation between different components of VMS

In our prototype, to call the service-proxy we use a method that has the following signature:

```
Service_proxy(String Thirdparty_name, String  Operation_type, String opetaion_input_data);
```

Its parameters are explained as follows:

- Thirdparty_name refers to the name of the third party that is responsible to provide the information to answer the requested query.
- Operation_type can get one of these three values: (1) Deploy, (2) Monitor (3) Apply _PTZ. This parameter indicates to the service-proxy that which of the three fore mentioned operations needs to be called.
- opetaion_input_data has a string value that is coded that the required information to do the one of the requested operations operation in the xml format.

After service proxy received the opetaion_input_data, it has controller method to extract the required information form opetaion_input_data value which is shown in Code 6.1. It has a ParseXMLString(element,xmldata) method that extracts the data about the element from the xmldata. The out-put of this method is a string in XML format that contains the required information about the element

```
Void controller(){
        cam-name= ParseXMLString (cam_name, opetaion_input_data);

                switch (Operation type) {
              case 1:  Operation type = "Deploy";
                            config_data= ParseXMLString(config, opetaion_input_data);
                             Deploy(Third_party-name, cam-name, config_data);
                   break;
              case 2:  Operation type = "Monitor";
                            Guard_IP_address= ParseXMLString(IP, opetaion_input_data);
                             Monitor(Third_party-name, cam_name, Guard_IP_address);
                   break;
              case 3:  Operation type = "ApplyPTZ";
        `          PTZ_preset= ParseXMLString(PTZ, opetaion_input_data);
                            ApplyPTZ(Third_party-name, cam_name, PTZ_preset);
                           break;
        }
```

Code 6.1 extracting the required information based on the requested operation

After extracting the required information to do the operations, we have three methods to call specific third-party operation . Each method make an appropriate format of data based on the supported data by the third-party and makes the request over the third-party (see code 6.2).

```
Deploy(Third_party-name, cam-name, config_data){
        If(Third_party-name= "Nedap")
                Nedap_deploy(cam-name, config_data);
        Else
                TP_deploy(cam-name, config_data);
}

Monitor(Third_party-name, cam_name, Guard_IP_address){
     If(Third_party-name= "Nedap")
                Nedap_monitor (cam_name, Guard_IP_address);
        Else
                TP_ monitor (cam_name, Guard_IP_address);
}


ApplyPTZ (Third_party-name, cam_name, PTZ_preset){
     If(Third_party-name= "Nedap")
                Nedap_applyPTZ(cam_name, Guard_IP_address);
        Else
                TP_applyPTZ(cam_name, Guard_IP_address);
}
```

6.5 Conclusion

In this chapter, we explained one of the possible deployment of VMS. Furthermore, We explained the implementation of the discussed part in chapter 5 (Design). That consist of the client application and the used APIs, registry and service-proxy

Chapter 7 Preliminary Evaluation

As discussed in the previous chapter, we have developed a VMS prototype as a proof of concepts. Our goal is to develop a middleware for a VMS, to facilitate configuration and deployment of camera devices for non-expert end-users.

ISO 9241-11 defines usability as "*Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specific context of use*"

Also ISO standard defines satisfaction as *"the extent to which users are free from discomfort, and their attitude toward the use of the product"*[38].

In our case, based on our goal, we aim to help non-expert end-users configure camera devices and deploy their configurations. Thus, we can use the ISO 9241-11 standard guidelines to evaluate our prototype whether it provides the satisfaction aspect of the usability definition.

Generally speaking, usability evaluation methods should consider both objective and subjective aspects. Objective evaluation is related to measure the system performance for instance, by logging the response time in a system. Subjective evaluation measures the participant's attitude or opinion based on their perception of usability. The standard guidelines suggest to measure satisfaction, which is subjective evaluation, through interview with the system end-users. [39]

To evaluate our prototype, we used a subjective evaluation process. It was inspired by evaluation studies done by other researchers ([39], [40]) in research settings like ours. We made questionnaire and performed interview with the practitioners, who are developing VMS in the company (Nedap). In this chapter, we explain our evaluation process and its result.

7.1 Evaluation process

In this section, we explain the interview place and the interviewees. In addition, we explain our idea behind making the questionnaire's questions.

We performed our interview in a commercial company (Nedap), which develops VMS. VMS provides video stream information to monitor different places, such as airport, companies.

We asked three of the company practitioners who are project manager, developer and architecture of a VMS developed by Nedap. The requirements of VMS have been identified by performing several interviews with the customers of the company. Then, the VMS developers work on VMS based on the identified requirements. Therefore, the VMS developers know about the end-user requirements in the system and they can be considered as representative of both admin-user and end-users to answer our questionnaire.

We wanted to gather interviewees feedbacks as much as possible without limiting them to our questions, therefore we used open-end questionnaire [41]. Our questionnaire has five open-end questions; the first three questions are about the general opinion of the interviewees to compare the current version of VMS with our prototype from non-expert end-users perspectives. The last two questions emphasize more on our usability goal and to what extent our prototype can provide it.

As Figure 7.1 shows, in the last two questions, we asked about the required time to configure camera devices using our prototype. The reason is, our prototype has capability of saving more than one configuration presets for each camera. Each configuration preset consists of a set of required attributes to configure a camera. This capability can save admin-user time to changing the camera configuration. Because it prevents admin-user to redefine the camera configuration preset which is already saved in the VMS. In addition,, we directly ask the interviewees about the ease of configuration to evaluate our usability goal. We will explain in more details about the idea behind each question in section 7.2.

1. Which tasks can you do with the system now that you could not do before?
2. Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.
3. Does using the system make camera configuration easier ?In which way?
4. Does using the system save your time to apply new camera configuration changes? in which way?
5. Are there things that are more difficult now than before? In which task and in which way?

Figure 7.1: list of the questions identified in the questionnaire.

We interviewed with the practitioners separately for two reasons: (1) the interviewees are as the *representative* of VMS end-user and not the actual end-users. Thus, more discussion time was required to interpret their evaluation from the end-user perspective; (2) since our interviewees play different roles in the development process of a VMS, performing separate interviews gave us the opportunity to investigate the future possibilities to improve the system.

We performed all of our interviews in the same way. First, we explained the goal of our interview to the interviewees. Then, to be sure the candidates have sufficient knowledge about the current VMS, we discussed the current VMS system. After that, we explained and presented our prototype. Finally, we asked them to fill in the questionnaire form. For clarity, we present the form in Appendix A (page 81). The data collected by means of the form is presented in the Appendix B (page 82)

7.2 The experiment and its results

In this section, we explain the results of our interviews individually for each question. After analyzing the interview data, we engaged another practitioner, who never worked with the VMS, to interpret our questionnaire results.

In the following, we mention our goal behind the questions and the interviewee's results for each question:

➢ Question1: Which tasks can you do with the system now that you could not do before?

  The goal of this question is to find the new functionalities that our VMS provides for end-users. In this question, we do not mention our goal in the question directly, so that we can to investigate the interviewees feeling about the functionalities that are added to our VMS.

  Interviewee 1: Currently, Nedap has a VMS to manage its own cameras. To extend the company VMS to support other type of cameras, we want to integrate the company VMS with other third-party camera service providers. Our first interviewee believes that our VMS add a uniform way to communicate with the old camera devices for end-users. These cameras can be managed by different third-party camera service providers or Nedap VMS.

  Also, in our VMS, we have designed two access levels roles, admin-user and guard. The interviewee mentioned that the idea of having admin-user is useful. Admin-user is an end-user who is able to ask third-party camera service providers to apply camera configurations. Old VMS does not support applying camera configurations by end-users.

Interviewee 2: He has the same opinion as the first one. He emphasized that our VMS provides a uniform way to handle the company cameras and the third-party cameras.

Interviewee 3: The third interviewee also mentioned the same points.

- Question 2: Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.

  The goal of this question is to have more specific examples about the new functionalities that our VMS provides for end-users.

  Interviewee 1: He believes that it is pleasing to have a cleared and practical way to configure the cameras. In this way, end-users do not need to think differently to apply configuration for each type of cameras.

  Interviewee 2: He believes, since he is not an end-user of this system, answering to this question is difficult for him.

  Interviewee 3: He believes that the end-user experience largely depends on user interface. Since our VMS developed for testing purpose to show how the system works, it is difficult for him to answer to this question

- Question 3: Does using the system make camera configuration easier? In which way?

  To facilitate camera configuration in our VMS, a admin-user can save more than one camera configuration setting in the system for each camera. In this way, if the admin-user wants to change a camera configuration to a predefined configuration in the system, he/she doesn't need to redefine the camera configuration. This feature can save camera configuration time. The goal of this question was to ask indirectly about this feature. The interesting point was our interviewee referred to other features.

Interviewee 1: He believes that we need more generic action to configure different cameras in our VMS. Thus, end-user needs to spend less time to learn how to work with the system.

Interviewee 2: He believes that the user interface has the main effect on saving time for end-users. Since the user interface of our prototype is for testing purpose, he cannot answer to this question.

Interviewee 3: He thinks our VMS can save time for end-users. However, testing the system with actual end-users is required to have more accurate answer.

- Question 4: Does using the system save your time to apply new camera configuration changes? in which way?

  The goal of this question is to ask directly about our main goal in this project. Therefore, we explicitly asked about the ease of camera configurations that our VMS provides for end-user.

  Interviewee 1: He believes that our VMS provides ease of camera configuration for admin-user. The reason is, since all of the cameras are configured in an uniform way, an end-user does not need to know the camera configuration of different camera types.

  Interviewee 2: He mentioned the same point as the first interviewee.

  Interviewee 3: He believes that if our VMS is integrated with third-party camera service providers, this can make the configuration easier.

- Question 5: Are there things that are more difficult now than before? In which task and in which way?
  The goal of this question is to find whether our VMS causes any difficulties for end-users or not.

  Interviewee 1: He did not mention any difficulties.

  Interviewee 2: He believes that configuring video service in the early stage of the camera configuration can cause more complex model in comparison with old VMS.

Interviewee 3: He believes in our system, end-user might have difficulty to know the available cameras. He mentioned that we should have a mechanism to update the list of the available cameras and inform end-users about them. For instance, a mechanism can be broadcasting a message periodically to the third-parties to get the list of their available cameras.

Before making the conclusion we asked someone who has clear mind about the VMS, to interpret our questionnaires' results. Since, he has independent judgments to interpret our system, his interpretation help us to provide more reliability to evaluate our system. His interpretation is as follows:

"The surveyed software can provide easier deployment and configuration of cameras because it provides an uniform way to accomplish these tasks for several camera types. This can make these tasks less time consuming and less challenging for non-expert users.

A disadvantage is that the model is more complex meaning that grouping of cameras needs to be done in an earlier stage of configuration. Also, the user interface can be improved. Because in the prototype end-users are required to know camera names in advance. These need to be entered by the user itself. A better way would be to present the possible options in a drop-down menu."

7.3 Discussion:

In this section, we explain our conclusion for each question and how our interviewees' answers can be related to our design.

Based on the answers to first question, we conclude that in general, our VMS can provide ease of communication with the third-party camera service providers and also ease of camera configurations for end-users.

Our interviewees provide this answer, because our VMS has a *service proxy* component. This component is responsible to make the requested query from the third-party camera service providers in an appropriate format, which are supported by them. Therefore, this component facilitates communication with third-party camera service providers.

Based on the answers to the second question, we should consider two further tasks to have more precise evaluation: (1) improving the user interface of our prototype (2) interviewing with the actual end-users instead of their representatives.

Our interviewees provide this answer, because the client applications for our prototype are for testing purpose.

Based on the answers to third question, we are not sure if having the capability of saving more than one configuration for a camera can save the camera configuration time. To have more precise answer to this question, we should test the prototype with actual end-users.

The interviews answer to this question was surprising for us. Because, we expected they mention to the configuration capability that our system supports, while they identified other features.

Based on the answers to the fourth question, we conclude if an end-user needs to interact with different type of cameras, our VMS can facilitate the configuration.

Our interviewees provide this answer, because our VMS has two components, namely (1) Configuring (2) Deploying, to provide ease of configuration for admin-users.

Based on the answers to the fifth question, we should add a mechanism to update the available cameras for the end-user. Moreover, we found that although defining video service can facilitate the interaction of a guard with the system, it can cause some difficulties for admin-users. The reason is an admin-user needs to set the required cameras for a video service in the configuration phase.

To make the configuration easier for admin-user, VMS can use some intelligent mechanisms to suggest admin-user the list of the required cameras to monitor an area. We will explain it more in the future work (Section 8.2)

Each video service provides full video stream coverage to monitor an area. The required cameras for each video stream usually are fixed. So, instead of redefining the address of the cameras to monitor an area we can define Video service. In this case, admin-user defines the required camera to monitor an area once as a video service in the system, later guards can use the video service name in our VMS, instead of addressing the cameras.

7.4 Validity Threats

In this section we discuss about the expertise and number of the interviewees to see the possible validity threats in our evaluation.

We could interview the VMS researchers, who do research on the next generation of the VMS in the company, for evaluating our VMS. Since the researchers have not worked on the current VMS in the company, they would probably have difficulties to answer our questionnaire from end-user perspective. We interviewed with the practitioners who know about the required functionalities for VMS end- users. If we would have interviewed with the VMS researchers, they could not have been as the representative of the VMS end-users as the practitioners were.

If we would have interviewed with more people with the same profiles as our interviewees, we would not have expected to obtain different responses. The reason is, all of our interviewees have provided very similar answer to our questionnaire. Therefore interview with three practitioners is sufficient. [42]

If we would have performed perform the interview with the actual end-users, we could have got more precise answers for some of our question. An example is our second question in which, we are looking for the new functionalities that our VMS provides for end-users. Since our interviewees have not worked as an end-user with our prototype for a long time, they have difficulties to answer this question.

7.5 Conclusion

In this chapter, we explained the way that we performed our questionnaire. To evaluate the result of our questionnaires, first we explained the interviews answer for each question. To have more reliable evaluation, after reading the questionnaire's answers, we discussed with the interviewees again about the unclear parts. To have a more reliable evaluation, we asked someone who has an independent judgment about the result of our questionnaires' answers to interpret the results.

According to the result of our interview, we can conclude that the designed system can satisfy the goal of our project. However, we should work more on the user interface to have more accurate result. Improving the user interface is helpful to provide easy access to the implemented features and to facilitate camera configuration. Since our evaluation is based on the end-user perspective, it would be better that we test it in a real scenario and with actual end-users.

Chapter 8 Conclusion

This thesis proposed architecture for Video Monitoring System (VMS). The aim of our research was to develop a middleware for a VMS, to facilitate configuration and deployment of camera devices for non-expert end-users.

In the requirement election phase of our approach, we interviewed researchers and developers of a commercial company, which develops VMS. Since VMS can be considered as an IoT-based system, we also reviewed literatures about IoT-based system to find the challenges in this area as part of the requirement election phase.

Based on the identified requirements, we conclude that making a generic data model for the configuration of the cameras would help end-users to have a uniform way to configure different types of cameras. This feature facilitates working with different types of cameras for non-expert end-users. Accordingly, we have developed our VMS prototype which implements a generic data model for cameras. The prototype has been evaluated in the commercial company.

In this chapter, we discuss the lessons which we learned based on our research questions and objective. Furthermore, we discuss some of the challenges which we have faced during our research and can have the potential to be considered as the future research topics.

8.1 Contributions

The Goal of our VMS has been evaluated with respect to ISO 9241-11 [38] that defines the usability satisfaction and also provides some guideline to evaluate it.

In Section 1.3, we listed four research questions. In this section, we described how the research questions have been addressed in this thesis.

RQ1. *What is the role of middleware in an IoT-based system?*

In Chapter 2, we reviewed some of the challenges to implement an IoT-based system. One of them is to deal with the heterogeneity of devices, which is the responsibility of an IoT-base middleware.

An IoT-base middleware facilitates working with heterogeneous types of data messages and communication protocols, which are supported by different devices. We summarize dealing with the device heterogeneity in two phases: (1) data processing phase that extracts information from sampled data message, and (2) data gathering phase that manages data sampling from devices with different communication protocols.

RQ2. *What are the main functional components of a middleware in an IoT- based system?*
In Chapter 2, we discussed the study on the existing middleware in IoT-based systems, which has been done by Bandyopadhyay, S. et. al [1]. To manage middleware interactions with a variety of devices, Bandyopadhyay S. et. al define four functional components, namely (1) interface protocols, (2) device abstraction, (3) central control, context detection & management, and (4) application abstraction.

To find out more about the role of these functional components, we discussed four IoT-based middleware in Chapter 3.

RQ3. *How to facilitate the interaction of non-expert users with different types of devices?*

We limited our research scope to camera devices. Thus, in chapter 4, we discussed the VMS and its functional and non-functional requirements. We gathered these requirements based on interviewing with the practitioners in the company and reviewing some of the related literatures.

Based on the identified requirements, we made an data model abstraction for configuration attributes of different camera devices. This was explained in chapter 5. Using this data model, our middleware can provide a high level homogenous view of the cameras. This homogeneity hides the detailed information to interact with different types of cameras for end-users. Therefore, it facilitates the interaction of non-expert users with the camera devices.

RQ4. *How to verify ease of this interaction with different types of devices?*

To validate our VMS, first we made a VMS prototype, which was explained in chapter 6. After that, in Chapter 7, we evaluate our prototype using questionnaires method and performing interviews with practitioners of a commercial company. We used this method because our goal can be interpreted as satisfaction in the usability definition of ISO 9241-11 standard[38]. Based on this standard guidelines, we can use questionnaire method to evaluate our prototype [39].

8.2 Future Research

In this section, based on the challenges that we have faced in our research, we discuss the possible research topics as follows:

- ***Other field test***

We made a data model based on the required camera configuration attributes. This data model provides a uniform way for end-users to work with different types of cameras. IoT-based system should be able to deal with a large variety of sensors and actuators.

Extending our data model to support more sensors and actuators is considered as a future research.

- ***VMS works in an intelligent way***

Camera configuration and interaction with cameras can be facilitated by applying some intelligent mechanisms as follows:

- Applying an intelligent mechanism to suggest more efficient cameras to help admin-user to make a video service. The efficiency can be defined based on the features that can be supported by the video monitoring system or can improve its performance. For instance, if storing video stream on the camera is supported by a VMS, in case of having two cameras with the same video coverage; the intelligent VMS should be able to suggest admin-user for the camera that has more available memory to record the video stream. Another example can be using some intelligent mechanism based on the mathematic algorithms to provide full video coverage with the minimum number of used cameras. This can improve the VMS performance.

- Another intelligent mechanism is to support an automatic notification event for end-user or a security reaction in case of violating a security rule in the monitoring area. For instance, if a motion detector sensor in a place be fired the VMS be able to start recording video stream from the appropriate video service that monitor the reported location or send the video stream to a responsible user in the system.

- ***Evaluating the system in run time***

We evaluated our prototype based on providing ease of configuration and deployment from end-user perspective, so we tested our prototype at deployment time.

As a future work, we can test the VMS at run time to check the performance of the system from third-party camera service provider and developer of the system perspectives. In that case, we should add two parts to the current VMS prototype as follows:

1) In our prototype, we assume that cameraTB and camera-service-providerTB tables have been initialized by the third-party camera service providers. In the future, we should add an interface (i.e. a RESTful interface) to get information from the third-party camera service providers to initialize and update these two tables.
2) Our prototype does not deal with receiving the video stream in the client application from the third-party camera service provider. Because, it is the

responsibility of the third-party camera service provider to manage sending the video stream. Thus, by sending the request to the consider third-party camera service provider we assume the video service will be receive in the client application. As the future work we can test receiving the video service in the client application as well.

References

[1] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of Middleware for Internet Of Things," *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 2, no. 3, pp. 94–105, 2011.

[2] M. A. Chaqfeh, N. Mohamed, P. O. Box, and A. Ain, "Challenges in Middleware Solutions for the Internet of Things," *International conference on Collaboration Technologies and Systems (CTS)*, no. IEEE, pp. 21–26, 2012.

[3] G. Mulligan, "An Interaction Independence Middleware Framework," Virginia Polytechnic Institute and State University, 2009.

[4] G. R. A. Schreiber, Romolo Camplani, Marco Fortunato, Marco Marelli, "Design of PerLa, a Declarative Language and a Middleware Architecture for Pervasive Systems," Politecnici di milano, 2010.

[5] F. Schreiber, M. Camplani, Romolo Fortunato, M. Marelli, and G. Rota, "Perla: A language and middleware architecture for data management and integration in pervasive information systems," *Software Engineering, IEEE Transactions on*, vol. 38, pp. 478–496, 2012.

[6] K. Henricksen, J. Indulska, and S. McFadden, Ted Balasubramaniam, "Middleware for Distributed Context-Aware Systems," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, R. Meersman and Z. Tari, Eds. Springer Berlin Heidelberg, 2005, pp. 846–863.

[7] S. R. R. Hevner, Salvatore T. March, Jinsoo Park, "Design Science in Information Systems Research," *MIS Quarterly 28(1)*, pp. 75–105, 2004.

[8] E. Guinard, Dominique and Trifa, Vlad and Mattern, Friedemann and Wilde, "5 From the Internet of Things to the Web of Things : Resource Oriented Architecture and Best Practices," in *Architecting the Internet of Things*, F. Uckelmann, Dieter and Harrison, Mark and Michahelles, Ed. Springer Berlin Heidelberg, 2011, pp. 97–129.

[9] J. Sen Debasis Bandyopadhyay, "Internet of Things - Applications and Challenges in Technology and Standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[10] Commission staff working document, "Accompanying document to the communication from the commission to the European parliament, the council, the European economic and social committee and 2 the committee of the region's future networks and the internet early," p. 18, 2008.

[11] ITU-T Study Group, "New ITU standards define the Internet of Things and provide the blueprints for its development," *ITU*, 2012. [Online]. Available: http://www.itu.int/ITU-

T/newslog/New+ITU+Standards+Define+The+Internet+Of+Things+And+Provide+The+Blueprints+For+Its+Development.aspx.

[12] A. Bassi, “Introduction (IoT-A),” *IoT-A*, 2013. [Online]. Available: http://www.iot-a.eu/public.

[13] T. Teixeira, S. Hachem, and N. Georgantas, “Service Oriented Middleware for the Internet of Things: A Perspective),” in *Towards a Service-Based Internet*, vol. 257178, no. 257178, J. Abramowicz, Witold and Llorente, IgnacioM. and Surridge, Mike and Zisman, Andrea and Vayssière, Ed. Springer Berlin Heidelberg, 2011, pp. 220–229.

[14] H. van der Veer and A. Wiles, “Achieving Technical Interoperability the ETSI Approach,” *ETSI White Paper*, no. European Telecommunications Standards Institute, 2006.

[15] H. Haas, W3c working group, A. Brown, and Microsoft, “Web Services Glossary,” *w3c*, 2004. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.

[16] D. Driscoll and A. Mensch, “Devices profile for web services version 1.1,” *OASIS, Mai*, 2009.

[17] M. Weiser, R. Gold, and J. S. Brown, “The origins of ubiquitous computing research at PARC in the late 1980s,” *IBM systems journal*, vol. 38, pp. 693–696, 1999.

[18] A. Malatras, A. H. Asgari, and T. Baugé, “Web enabled wireless sensor networks for facilities management,” *Systems Journal, IEEE*, vol. 2, no. 4, pp. 500–512, 2008.

[19] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TinyDB: An acquisitional query processing system for sensor networks,” *ACM Transactions on Database Systems (TODS)*, vol. 30, no. ACM, pp. 122–173, 2005.

[20] W3c working group, “PervasiveComputing,” *W3C*, 2004. [Online]. Available: http://www.w3.org/wiki/PervasiveComputing.

[21] F. Mazzenga and M. Vari, “Indoor localization techniques based on active and passive devices,” *RFIDays*, 2008.

[22] J. Soldatos, N. Kefalakis, and N. Leontiadis, “ASPIRE Programmable Engine (APE),” *ASPIRE Consortium*, vol. 3, 2011.

[23] A. Salehi, M. Riahi, S. Michel, and K. Aberer, “GSN, middleware for stream world,” in *10th international conference on mobile data management (MDM 2009), May*, 2009, pp. 18–20.

[24] P. Eisenhauer, Markus Rosengren and P. Antolin, *Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems*. Springer, 2010, pp. 367–373.

[25] C. Hartung, R. Han, C. Seielstad, and S. Holbrook, *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM, 2006, pp. 28–41.

[26] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, *Wireless sensor networks for habitat monitoring*. ACM, 2002, pp. 88–97.

[27] J. Werner-Allen, Geoffrey Johnson, M. Ruiz, J. Lees, and M. Welsh, “Monitoring volcanic eruptions with a wireless sensor network,” in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, IEEE, 2005, pp. 108–120.

[28] P. Jo and D. Garlan, “Aura: an architectural framework for user mobility in ubiquitous computing environments,” *Kluwer*, 2002.

[29] J. Domingues, A. Damaso, R. Nascimento, and N. Rosa, “An Energy-Aware Middleware for Integrating Wireless Sensor Networks and the Internet,” *International Journal of Distributed Sensor Networks*, vol. 2011, pp. 1–19, 2011.

[30] C. S. M. Villa, D. Rotondi, M. Comolli, (TXT), A. M. H. (Siemens) H.–P. Huth, C. K. (City), H. T. (inIT), and J. C. (EMIC), “WP 1 – PLUG&WORK IOT REQUIREMENT ASSESSMENT AND ARCHITECTURE,” 2010.

[31] D. Yazar and A. Dunkels, “Efficient application integration in IP-based sensor networks,” in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, Berkeley, California: ACM, 2009, pp. 43–48.

[32] D. Guinard, I. Ion, and S. Mayer, “In Search of an Internet of Things Service Architecture : REST or WS- *? A Developers ’ Perspective,” in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Springer, 2012, pp. 326–337.

[33] Z. Song, A. C. Alvaro, and R. Masuoka, “Semantic Middleware for the Internet of Things,” 2010.

[34] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer Networks*, vol. 54, no. elsevier, pp. 2787–2805, 2010.

[35] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W. . Meuter, “Middleware for the internet of things, design goals and challenges,” *CAMPUS 2010*, vol. 28, no. Electronic Communications of the EASST, pp. 1–7, 2010.

[36] Y. Liang, X. Zhou, Z. Yu, H. Wang, and B. Guo, “A Context-Aware Resource Management Framework for Smart Homes,” *Ubiquitous Information Technologies and Applications (CUTE), 2010 Proceedings of the 5th International Conference on*, no. IEEE, pp. 1–8, 2010.

[37] ONVIF, “Specifications,” *ONVIF*, 2013. [Online]. Available: http://www.onvif.org/Documents/Specifications.aspx.

[38] I. 9241-11, “ISO. Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability,” 1998. .

[39] Sh.zarghami, “Architectural support for dynamic homecare service provisioning,” 2013.

[40] Mohammad Zarifi Eslami, “Service Tailoring: A Method and Tool for User-centric Creation of Integrated IT-based Homecare Services to Support Independent Living of Elderly,” Twente, 2013.

[41] A. H. Maya Daneva, Luigi Buglione, “Software Architects’ Experiences of Quality Requirements: What We Know and What We Do Not Know?,” *Proceedings of REFSQ*, pp. 1–17, 2013.

[42] R. S. Peter B. Seddon, “Towards the improved treatment of generalization of knowledge claims in IS research: drawing general conclusions from samples,” *EJIS 21(1)*, pp. 6–21, 2012.

Appendix A: Questionnaire sample

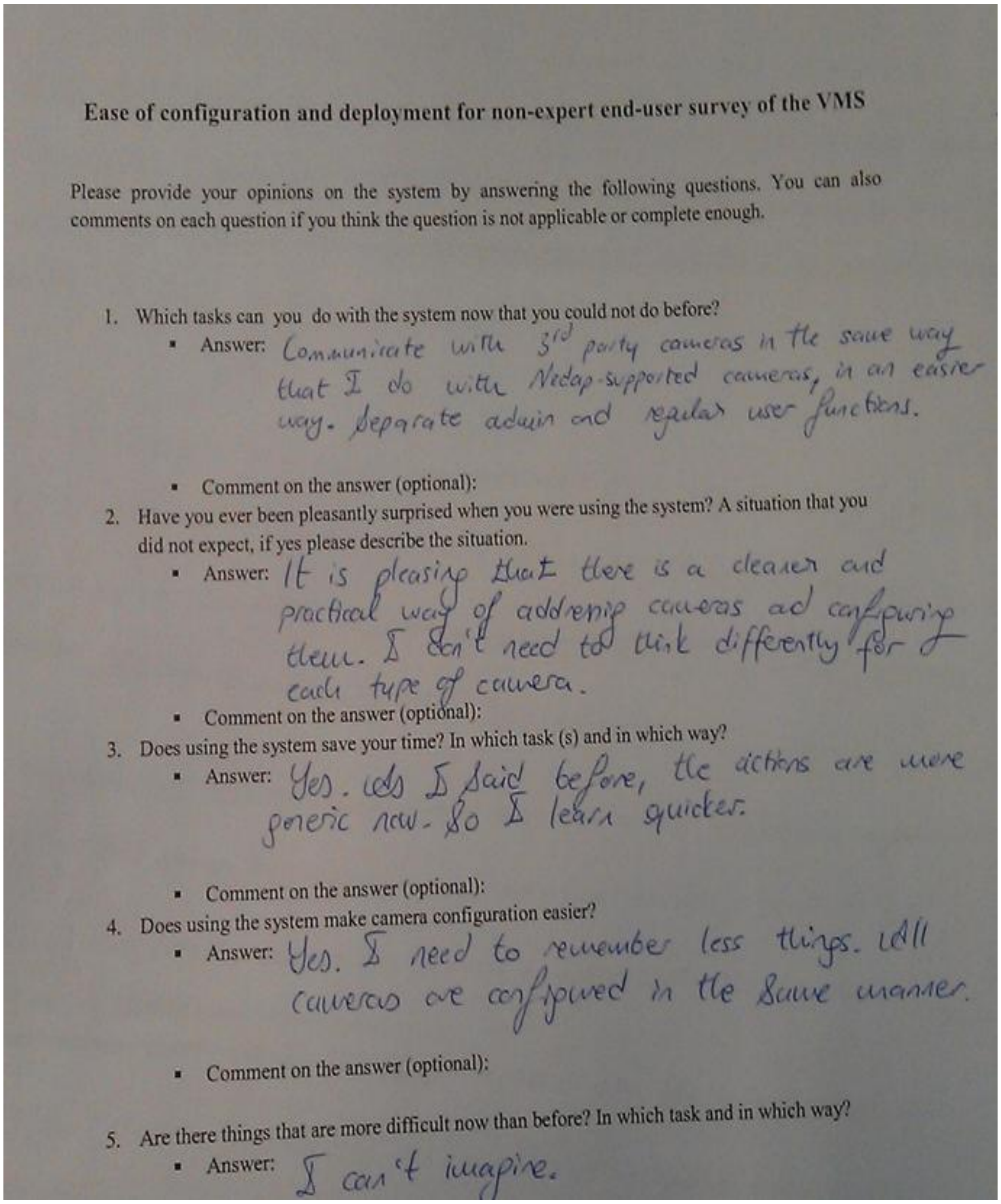
Ease of configuration and deployment for non-expert end-user survey of the VMS

Please provide your opinions on the system by answering the following questions. You can also comments on each question if you think the question is not applicable or complete enough.

1. Which tasks can you do with the system now that you could not do before?
    - Answer:
    - Comment on the answer (optional):
2. Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.
    - Answer:
    - Comment on the answer (optional):
3. Does using the system save your time? In which task (s) and in which way?
    - Answer:
    - Comment on the answer (optional):
4. Does using the system make camera configuration easier?
    - Answer:
    - Comment on the answer (optional):
5. Are there things that are more difficult now than before? In which task and in which way?
    - Answer:

- Comment on the answer (optional):

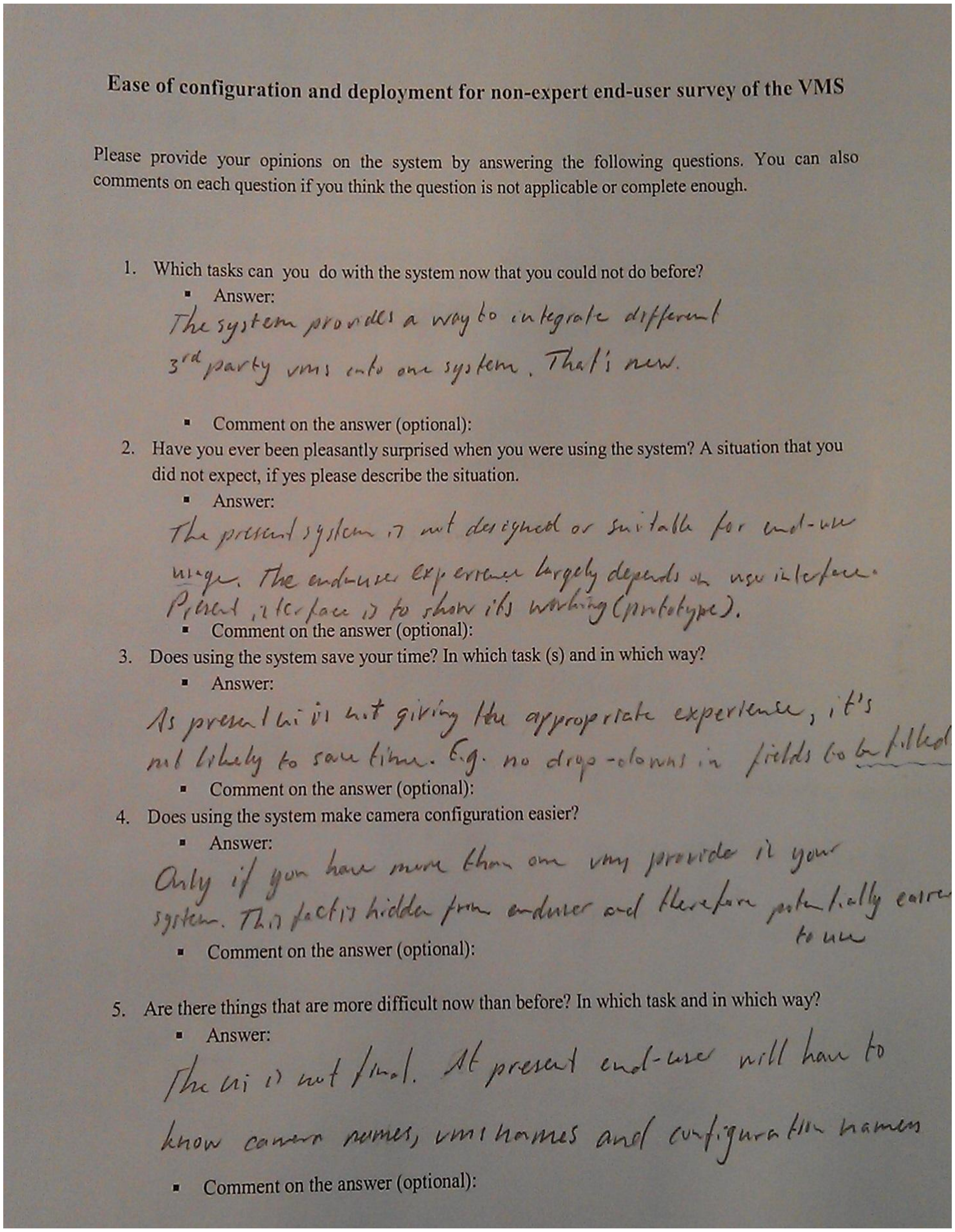
Appendix B: Questionnaire Results



Ease of configuration and deployment for non-expert end-user survey of the VMS

Please provide your opinions on the system by answering the following questions. You can also comments on each question if you think the question is not applicable or complete enough.

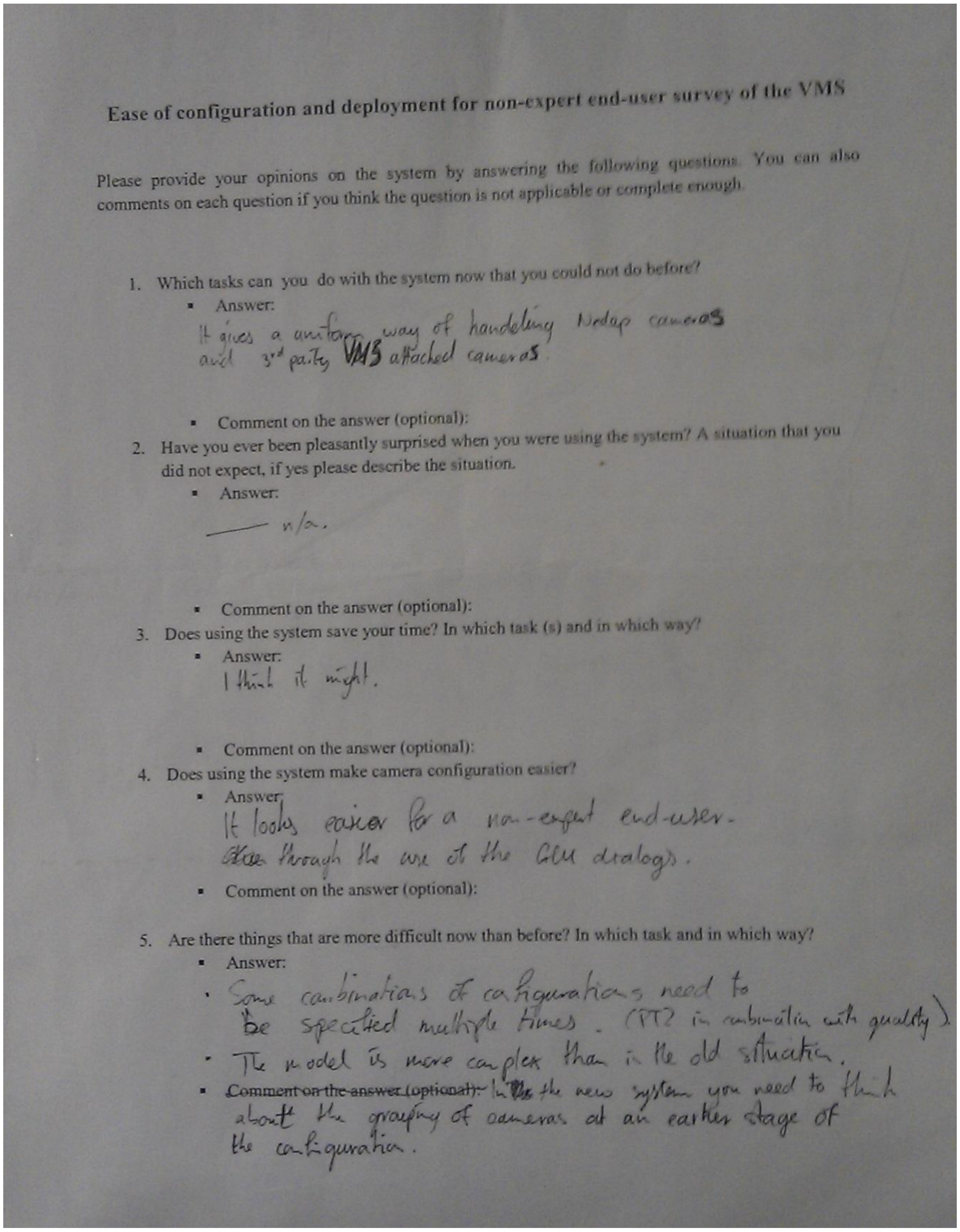
1. Which tasks can you do with the system now that you could not do before?
   - Answer: Communicate with 3rd party cameras in the same way that I do with Nedap-supported cameras, in an easier way. Separate admin and regular user functions.
   - Comment on the answer (optional):
2. Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.
   - Answer: It is pleasing that there is a clearer and practical way of addressing cameras and configuring them. I don't need to think differently for each type of camera.
   - Comment on the answer (optional):
3. Does using the system save your time? In which task (s) and in which way?
   - Answer: Yes. As I said before, the actions are more generic now. So I learn quicker.
   - Comment on the answer (optional):
4. Does using the system make camera configuration easier?
   - Answer: Yes. I need to remember less things. All cameras are configured in the same manner.
   - Comment on the answer (optional):
5. Are there things that are more difficult now than before? In which task and in which way?
   - Answer: I can't imagine.



Ease of configuration and deployment for non-expert end-user survey of the VMS

Please provide your opinions on the system by answering the following questions. You can also comments on each question if you think the question is not applicable or complete enough.

1. Which tasks can you do with the system now that you could not do before?
   - Answer:
     The system provides a way to integrate different 3rd party vms into one system. That's new.
   - Comment on the answer (optional):
2. Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.
   - Answer:
     The present system is not designed or suitable for end-user usage. The end-user experience largely depends on user interface. Present interface is to show its working (prototype).
   - Comment on the answer (optional):
3. Does using the system save your time? In which task (s) and in which way?
   - Answer:
     As present ui is not giving the appropriate experience, it's not likely to save time. E.g. no drop-downs in fields to be filled
   - Comment on the answer (optional):
4. Does using the system make camera configuration easier?
   - Answer:
     Only if you have more than one vms provider in your system. This fact is hidden from enduser and therefore potentially easier to use
   - Comment on the answer (optional):
5. Are there things that are more difficult now than before? In which task and in which way?
   - Answer:
     The ui is not final. At present end-user will have to know camera names, vms names and configuration names
   - Comment on the answer (optional):



Ease of configuration and deployment for non-expert end-user survey of the VMS

Please provide your opinions on the system by answering the following questions. You can also comments on each question if you think the question is not applicable or complete enough.

1. Which tasks can you do with the system now that you could not do before?
   - Answer:
     It gives a uniform way of handeling Nedap cameras and 3rd party VMS attached cameras.
   - Comment on the answer (optional):
2. Have you ever been pleasantly surprised when you were using the system? A situation that you did not expect, if yes please describe the situation.
   - Answer:
     —— n/a.
   - Comment on the answer (optional):
3. Does using the system save your time? In which task (s) and in which way?
   - Answer:
     I think it might.
   - Comment on the answer (optional):
4. Does using the system make camera configuration easier?
   - Answer:
     It looks easier for a non-expert end-user. Also through the use of the GUI dialogs.
   - Comment on the answer (optional):
5. Are there things that are more difficult now than before? In which task and in which way?
   - Answer:
     - Some combinations of configurations need to be specified multiple times. (PTZ in combination with quality).
     - The model is more complex than in the old situation.
   - ~~Comment on the answer (optional):~~ In the new system you need to think about the grouping of cameras at an earlier stage of the configuration.