Ivan Suvorov

# Servers and Extendable Server's Management System for the Robot-platform

## Part of the "Roboteh" project

Bachelor's Thesis
Information Technologies

May 2012

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

# DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

**Date of the bachelor's thesis**
May 2012

**Author(s)**
Ivan Suvorov

**Degree programme and option**
Information Technologies

**Name of the bachelor's thesis**

Servers and Extendable Server's Management System for the Robot-platform
Part of the "Roboteh" project

**Abstract**

This thesis shows how to create the system which allows to control the robot-platform "Decartus" through a network, store data from the sensors on the robot-platform, transmit the real-time video stream from the robot-platform video camera. The aim of this work is to create the system between a client side like an application on a laptop, a smartphone or regular PC and the robot-platform "Decartus". This system should provide communication between them, transmit commands from a client side to the robot-platform and transmit the video-stream from the robot-platform in the real-time mode. The requirements were successfully met. All of these processes are provided throughout a network by this system. Also this work shows how to minimize the delay of data, video stream and commands interchanging. I have created more than one server to provide these functionalities between a client side and the robot-platform and reached even more than acceptable delay for the real-time video and commands transmission. The system successfully provides its own APIs, control and video stream interfaces for the third party applications which can run on a client side and intercommunicate with the robot-platform "Decartus".

**Subject headings, (keywords)**

Servers, database, serial port, video streaming, RTP, JSON,JSONP, AJAX, Roboteh, controlling through a network, robot, Java, web-servers

| Pages | Language | URN |
|---|---|---|
| 75 | English | |

**Remarks, notes on appendices**

| Tutor | Employer of the bachelor's thesis |
|---|---|
| Matti Koivisto | Mikkeli University of Applied Sciences |

**CONTENTS**

# 1 INTRODUCTION

Nowadays in the century of computer's technologies, digital information and data have a great value in our life. Amount of data is increasing day by day. It needs complex systems to manage data flow as well as important information and data which are stored on servers or in data centre. This problem is quite actual, so in my thesis I introduce the management system developed by me which includes more than one server and has interfaces to communicate with other parts which can be other servers or clients.

The topic of the thesis is "Servers and Extendable Server's Management System for the Robot-platform", part of the "Roboteh" project. It is aim is to find ways to manage all data flows between the remote robot-platform "Decartus" and the "Extendable Server's Management System" and other servers or clients parts. In this thesis some technologies and techniques of the IT field connected in the one big project "Roboteh" are introduced.

This project consists of three parts: "Web Control System of Mobile Platform" part, the server side part which is called "Servers and Extendable Server's Management System for the Robot-platform" and the hardware part which is called "Decartus". "Web Control System of Mobile Platform" part is developed by Vitalii Klimenko. "Decartus" is developed by Stanislav Shults. They are members of this project which is called the "Roboteh".

In this project I have developed "Servers and Extendable Server's Management System for the Robot-platform" and I will explain that part in a more detailed way. In our project we have robot-platform controlled through a network remotely and also send and receive data to and from the platform. It means that if we have a network with one or more computers we can control our robot-platform from any computer in the network. The network can be the Internet or a local network, it depends on intention.

At the moment we have few requirements to the project. Firstly, we have to control robot-platform remotely. Secondly, server has to be able to receive a video stream from the robot-platform and spread it among customers or end-users. Thirdly, it should be a cross-platform and be easy to expand. In this project we are trying to

reach these three requirements.

Nowadays, the Internet has become more popular and it has spread everywhere. And the idea of combining a low level components (as sensors, controls components and etc.) and a high level features (as a network or the Internet) together is good from my point of view for most of people in the world, because everything is in digital form or has digital information.

The main goal of that work (project) is to find some combination of technologies which are related to robotized machines and web-technologies. Here we want to understand which technologies are needed to make a system be able to handle a remote robot-platform through a network.

"Servers and Extendable Server's Management System" part is the core of whole project which contains few servers and has interfaces to communicate with inner modules or servers and also communicate with outer world. Also "Servers and Extendable Server's Management System" part is the middle layer which helps to communicate to end-users with robot-platform. There are a few methods to rich that communication. The main problem is that if we want to control some robot or a robot-platform in a regular way we need directly to be connected to certain computer system which controls a robot-platform. It can be the system which is connected to COM-port or Ethernet-port or something else with some application on computer. But if there is a need to control or analyse data from a robot-platform it is going to be a problem. In terms of regular users "Extendable Server's Management" can be like Operation System which abstracts a hardware stuff and offers services to an end-user or users. That is what I want to achieve in my part of this project. For instance, we have the source of data (the robot-platform) with the video camera and sensors and all data is transmitted to "Servers and Extendable Server's Management System" part. "Servers and Extendable Server's Management System for the Robot-platform" part converts the data and then, "Web Control System of Mobile Platform" part can use data through the interfaces. In the other direction "Web Control System of Mobile Platform" part sends a control data to an interface and on "Servers and Extendable Server's Management System for the Robot-platform" part that data is converted and sent to hardware.

The structure of final thesis is the following. Chapter 2 Network and Internet describes

the main network technologies and protocols also their benefits and drawbacks. Chapter 3 Servers and services. Databases describes the main purposes of servers, services and their definitions. It also describes a role of a database in the system. Chapter 4 Audio/video streaming describes the main entities of the media streaming. Chapter 5 Java and cross-platform solutions describes different platforms for applications and the main differences between them. Chapter 6 Design of the servers and Extendable Server's Management System shows the main steps in design of the servers and Extendable Server's Management System. Chapter 7 Implementation describes implementation and deploy of the servers and systems. Finally in Chapter 8 I make some final conclusions and summarise the results of my thesis.

# 2 NETWORK AND INTERNET

Nowadays networks are used everywhere. Almost everybody has a mobile phone, a laptop, a PC, etc. Each of these devices can be a part of a network as GSM, the Internet, a local and other networks. A network may contain many different kinds of devices. It is possible, because all network system's architectures have specific interfaces for that.

## 2.1 Advantages and drawbacks of a network

Every technology has its own benefits and drawbacks. The main benefits of a network are:

- Exchange data at the distance (the global distances).
- Share a broad area of resources.

On the other hand the drawbacks, which exist in a network, are:

- Security problems.
- Addition resources for exchange data.
- Drawbacks of wired and wireless communication link.

In general case a network gives lots of advantages. It helps to establish a mesh topology to connect different devices in different places around the world. Also, a network

helps to distribute among devices computations and data. A distributing data and computations among devices is extremely important for growing technologies.

A good example of requirements can be computation of a 3D environment. A modeling of 3D objects requires computing power as much as possible. And the calculation or rendering can take from few minutes to many days, as the rendering time depends from many factors inside a 3D scene. To shorten the time specific techniques were developed to distribute computation among devices through network. This kind of method reduces cost of equipment and increases the speed of a computing process. The example above is a sort of specific purpose of a network.

### 2.2 The Internet

The Internet is the hugest world-wide network which consists of many kinds of networks. Almost everyone has access to that network. And that is the biggest advantage of the Internet. The Internet consists of many services for many types of purposes. Social networks have become the most popular activity in the Internet. On-line gaming, storage sources, an enterprise collaboration, multimedia streaming and even services from real live as newspapers; books are moving to digital format and especially to the Internet. Many medical centers have services in the Internet to help customers choose a right product. Banks offer on-line banking and etc. This list is endless.

The Internet can be defined in many ways. For example, Jim Kurose and Keith Ross in their book “Computer Networking” have described the Internet as an infrastructure that provides services to applications, or more precisely an infrastructure for providing services to distributed applications. (Kurose, 2010, 5-6).

The definition points out that the main purpose of the Internet is to be publicly available and provide necessary services for end-users. An end-user is able to connect to a network through network devices and use services which are available in the Internet.

### 2.3 Network security

In some cases being a part of some network is dangerous. For instance, if some company has private information and to that resource, somehow, some connection is es-

tablished to that resource, somebody could copy or rewrite information without rights to do it. At the moment, many of “dark” developers are able to make many specific applications break the rules and get “secret” and “valuable” information. From that perspective, a network is not a safe place in its original state.

# 3 SERVERS AND SERVICES. DATABASES

## 3.1 Servers and services

### 3.1.1 Servers

The server is one of the main parts of a network. The client-server architecture has two entities. There are a client and a server there. The name of architecture refers to the main entities: the client and the server entities as it is shown in Figure 1.

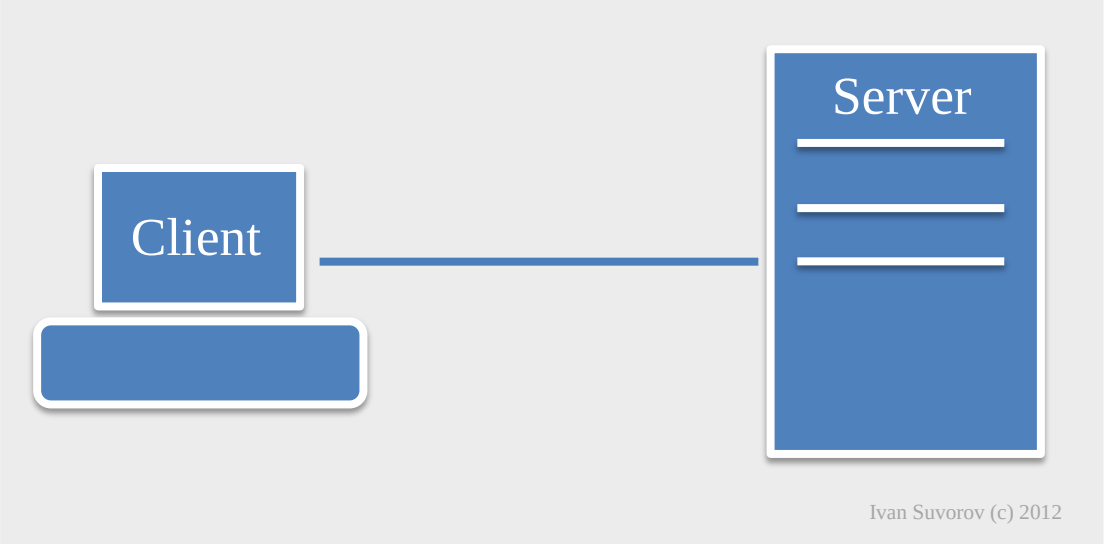


Figure 1. The client-server architecture

Nowadays almost everyone has a personal computer (PC), a laptop, a smartphone and other gadgets. Each of those devices has potential to be a part of a network as a client or as platform for a server.  Checking e-mail in vacation, uploading videos or photos, sending reports to a manager is possible because a client can have access to servers and their services. In the context of the client-server architecture, a server is a comput-

er program running to serve the requests of other programs, the "clients" (Server (computing), [referred 19.03.2012]).

The definition shows that server is not hardware. The server is a program running on server hardware. Almost each a computing platform with interfaces to other computing devices has ability to be a server. Efficiency and cost are the main reason why the server programs require special hardware.

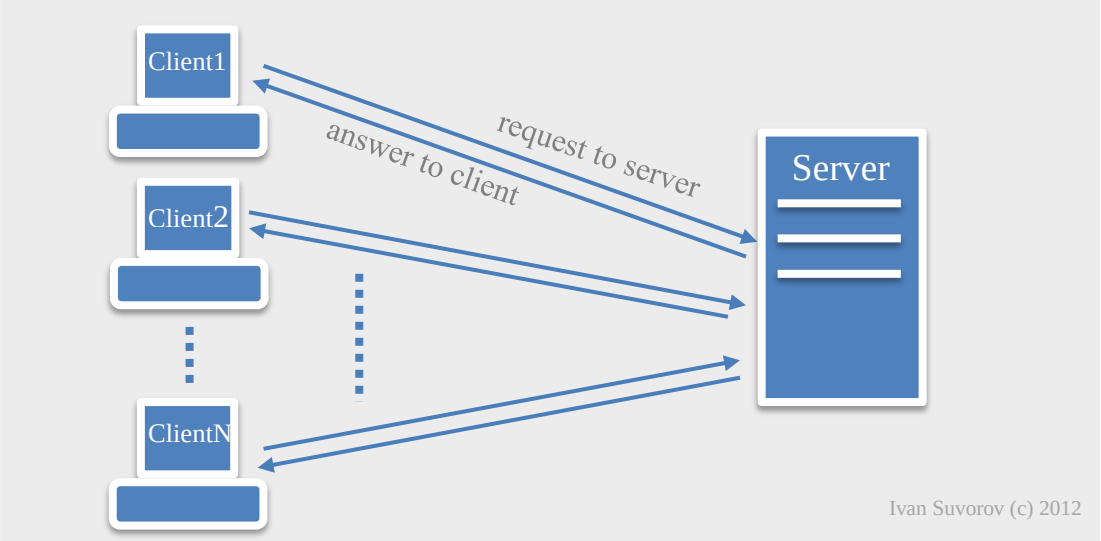


Figure 2. Many to one structure of the client-server communication

Figure 2 shows that basic type of the client-server architecture is many-to-one. It is the star-structure. That structure is the common way to distribute a service among many clients.

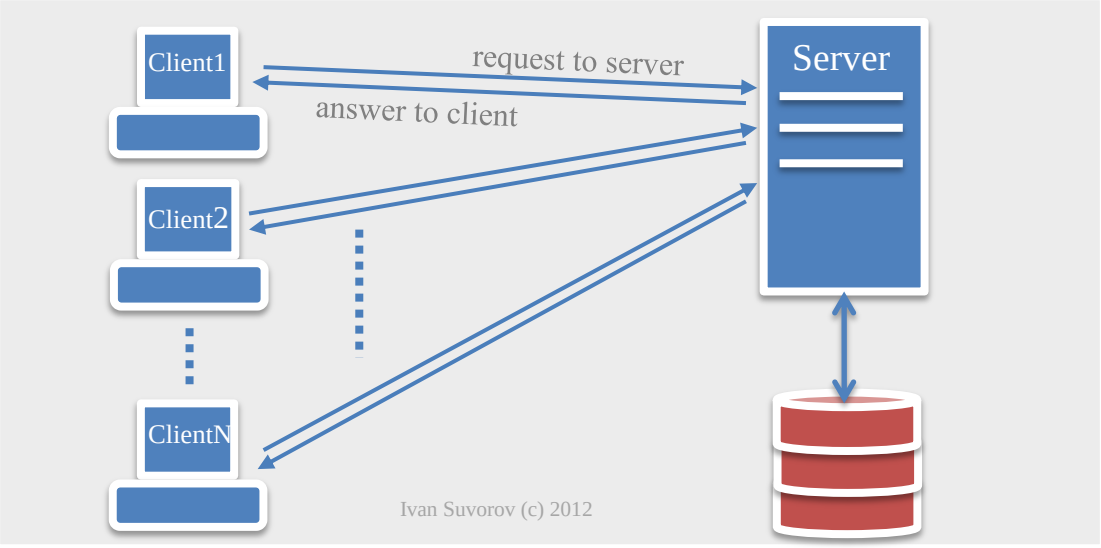


Figure 3. Modern version of the traditional client-server structure.

Figure 3 shows the same structure of relations between the server and clients as Figure 2, but further to that, Figure 3 emphasises an important part of the server. There is a database, which has recently become an undivided part of almost all servers. More details of the database can be found in section 3.2.

The server provides us with many things like computing, storing, searching, managing and etc. The Internet is based on servers. The Internet is a big virtual server with many different services.

### 3.1.2 The Web Services

Recently the Web services have become fundamental part of people's activities. Almost all services are available in the Internet or on a network. A communication and collaboration have become a basic of the Web-services. Many social, streaming and storage services compete with each other. The biggest companies started to invest in web-projects and technologies a lot of money. Each of them tries to renew its own technologies and establishes new departments to find out new ways to reach a better position among competitors. In information technology, a server is a computer program that provides services to other computer programs (and their users) in the same or other computers (server, [referred 19.03.2012]).

In other words, server is a program which offers some service to end-users or clients. The difference between servers and services is that a server offers services in other words server can offer from one to many services.

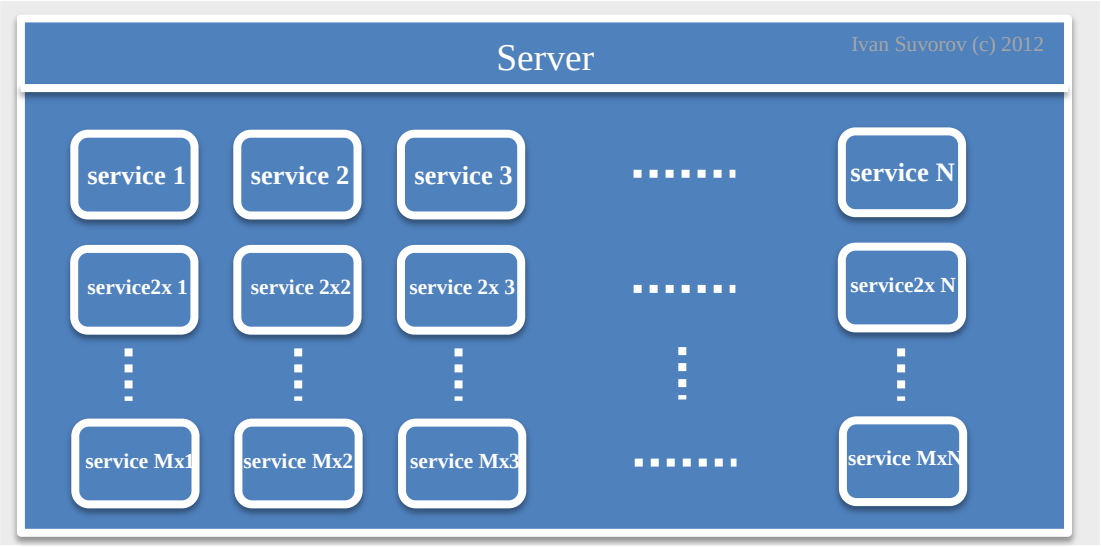


Figure 4. Services inside a server

Figure 4 shows the typical structure of a modern server. A server processes one or more services simultaneously. In this case, virtualization and proper hardware help a lot to carry and process services in proper way. At the same time, a design of server should be kept in one area of specialization. For example, a database server has services which are related with data management and another server is tuned for a computation. These two servers are different in configurations and have different bundle of services.

### 3.2 Database

Databases have a great role in modern life. From the ancient time humanity has had a need to store, classify and manage its knowledge. A good example of this is a library. However, the important development step humanity made was the invention of the digital computer. From that the digital era was started.

At the moment there are almost seven billion people on the Earth. Each of them has pieces of information in the "global" database either in a digital, paper or in some oth-

er form. Not everything has become digital, but it the amount of a digital information is increasing all the time.

Thus digital databases grow every day. Private and public information records are stored in digital form and are used daily. Many types of information are collected for different purposes. It can be either personal information like names, surnames, dates of birth and etc. or secure information about bank accounts, passwords, information about taxes and etc. Shop, enterprises, banks, governments, schools, universities all of them have databases.

The definitions below describe basic purposes of a database system. A database is an organized, machine-readable collection of symbols, to be interpreted as a true account of some enterprise. A database is machine-updatable too, and so it must also be a collection of variables. A database is typically to a community of users, with possibly varying requirements. (Darwen, 2010, 14) Also Carolyn Begg has written in his book "Database Systems. A practical Approach to Design, Implementation, and Management" that a database is a shared collection of logically related data and its description, designed to meet the information needs of an organization. (Begg, 2010, 15).

First of all the database is a system plus data. Unstructured data cannot become database. It is important for a database to have a structured form. Data structuring is supported by DataBase Management System (DBMS).

### 3.2.1 Main types of DBMS

The Database Management System (DBMS) is the core of a database. In terms of information technologies, DBMS is operation system for structured data. That system allows users to create, modify and manage the information. DBMS has four main types which are described in the Database Fundamentals topic of the Penn State University (Database Fundamentals, [referred 19.03.2012]). The types are:

- hierarchical databases.
- network databases.
- relational databases.
- object-oriented databases.

In addition to the main types there are combinations of those types. One of the popular combinations is "object-relational database". One example of these bound systems is PostgreSQL.

All DBMS types have their own advantages and drawbacks. The most popular type is relational database with convenient structure of language (SQL). Relational databases reduce redundancy during design.

The main requirements for DBMS are:

- Agility
- Reliability
- Scalability

Agility describes the time spent to modifications or retrieving. That parameter is important, because many systems need to insert, modify or read data as fast as possible. Additionally, if that system is used by thousands or million users, every millisecond is important.

Reliability is important, because if, the service doesn't work properly, it decreases the worth of that service. Also, it's important when that service has a critical significance as banking, medical issues, air transport and etc.

Scalability has recently become more and more important. Enterprises are growing constantly. The number of users of services is increasing daily. Flexible system is necessary to cope with the growth.

### 3.2.2 Web and DBMS

Many services are available from the web. The Internet offers lots of social, streaming, storage, and computing services. Each of them needs a place to store information about visitors, customers, and etc. A service needs a system to manage its content or part of that content.

### 3.2.3 PostgreSQL

PostgreSQL is object-relational database system (ORDBMS). It is open-source project for enterprise level. It has many features to reach scalability, reliability and security.

PostgreSQL is an enterprise class database. PostgreSQL has sophisticated features such as Multi-Version Concurrency Control (MVCC), point in time recovery, table-spaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write ahead logging for fault tolerance. It supports international character sets, multibyte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of four terabytes of data. (PostgreSQL About, [referred 19.03.2012])

PostgreSQL is the one of the most popular DBs in the world. It is scalable, reliable and able to manage data flows as quick as possible. Many projects are delay sensitive and they must be able to manage many users. From these perspectives, many companies have chosen PostgreSQL as their main DBMS.

## 4 AUDIO/VIDEO STREAMING

Decreasing cost for PCs and photo-video cameras as well as a network accessibility of almost everyone makes a new field of service. There is a multimedia streaming. The range of a multimedia data consists of three main parts. There are digital images, video files and audio files. In multimedia streaming that data is converted into a stream and transmitted across a network.

### 4.1 Streaming Process

The Streaming process has four parts: an acquisition, a preparation, a distribution and a playback. The first step is acquisition of media from its source. A source can be a camera, a microphone or some storage device (e.g. DVD, CD or stored multimedia on HDD). On preparation step, a source is applied to some sort of modifications. Basical-

ly, a source is converted to a suitable form for transmission. Also, at this step a source can be modified, compressed, applied pre-processing filters. On distribution step the process establishes connection to the media player from the streaming server. Finally, the client receives, decompress and play multimedia stream. (Media Streaming, [referred 19.04.2012])

Multimedia streaming has the three main types (Kurose, 2010, 598).

- A stored audio/video
- A live audio/video
- A real-time interactive audio/video

### 4.1.1 Stored audio/video

The stored audio/video (a/v) streaming is still growing and becoming the biggest part of the Internet. iTunes, YouTube, Netflix, Spotify and other streaming services are the most popular web-resources in the world. Their main content is stored audio/video. That source is growing daily. Many movies and drama's episodes filmed on today; also TV shows and etc. are stored on these servers. The main source of that content is multimedia file on a server. When the once produced content is stored on server it can be delivered to customers on demand.

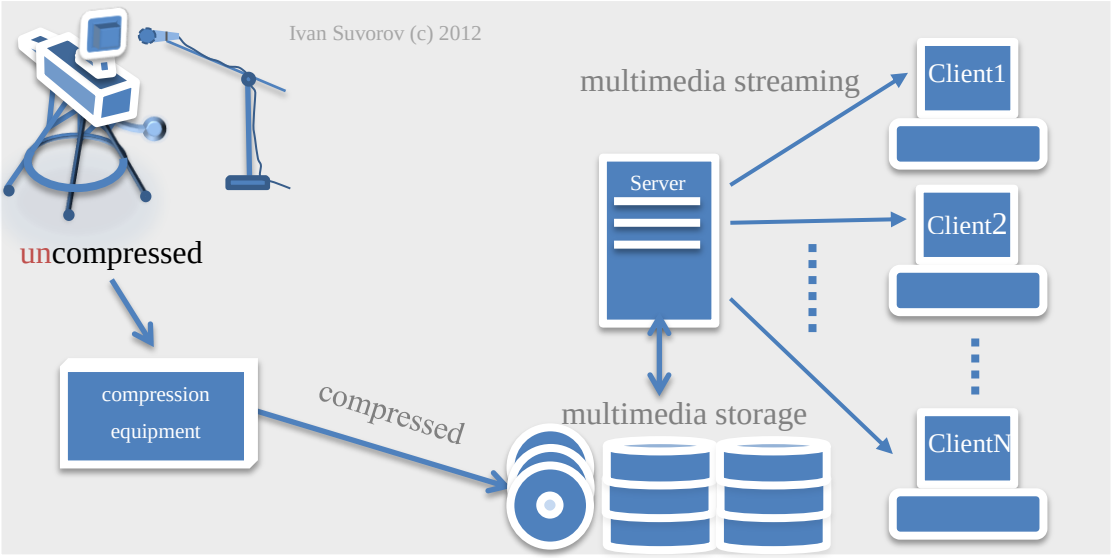


Figure 5. Stored multimedia streaming structure

Figure 5 shows how a typical process of the stored multimedia streaming works. First of all, a streaming needs a source or in other words – content. Content can be taken from video-cameras in studio, microphones or rendering by software for 3d modelling. Then, the content is compressed and stored in multimedia storage with other sources. When the user wants to watch or listen a video or a music, he sends through the client a request to the server. The server takes the multimedia source from the multimedia storage and converts it into a multimedia stream. After that, the server transmits the multimedia stream to the client. The client, on its side, gets the multimedia stream and decompresses it and plays it to the user.

That type of streaming allows storing multimedia content in a centralized place. To that place many users from different places can be connected and get access to multimedia content throughout a network. It replaces the traditional way to store multimedia content in a local storage as HDD, CD, DVD or a flash drive.

### 4.1.2 Live audio/video

The live audio/video streaming is used to transmit live events, competitions and live-shows. The name refers to type of multimedia source. The main purpose of live streaming is to transmit a captured multimedia content immediately after capturing. However, a live streaming can have bigger delays than a real-time streaming which is described in section 4.1.4 (Kurose, 2010, 599).

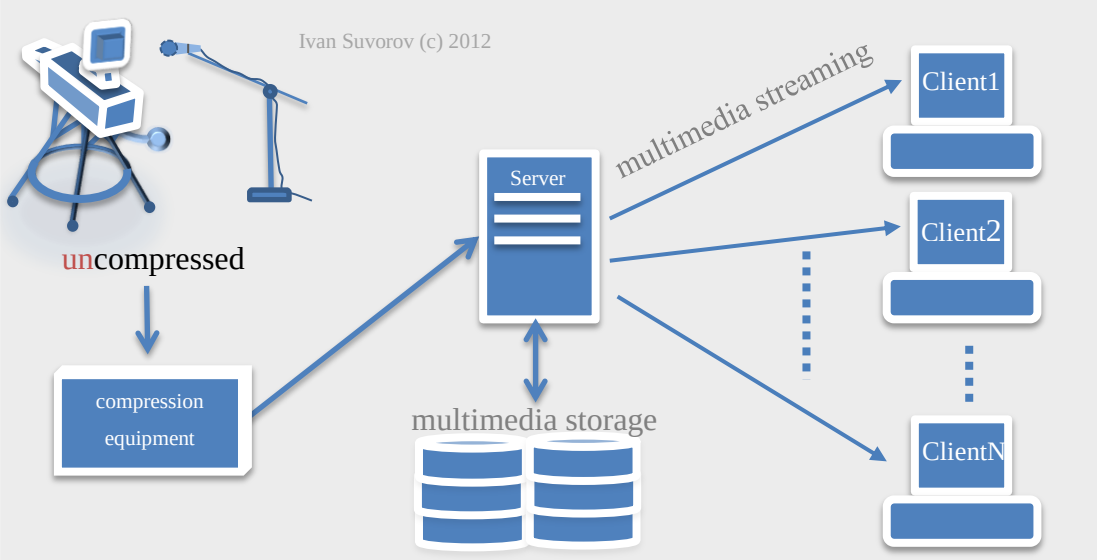

**Figure 6. Live multimedia streaming structure**

Figure 6 shows the structure of the live multimedia streaming with its components and streams. The difference between a stored streaming and a live streaming contains in the type of content and the purpose of the multimedia storage. The live streaming content is being captured at the same time with its transmission to end-users. In that case, the multimedia storage has the role of buffer for the multimedia content after compression and before converting it to a multimedia stream. The multimedia storage can be used to store multimedia content as well. The end-to-end delay can be within a range from one second to few minutes. The live multimedia streaming is delay tolerant, because an end-user does not care about delay in seconds between original time of a filming and his own time of watching.

The live multimedia streaming is also loss tolerant. A user can understand the meaning of the content without its milliseconds of data. If a network drops or loses few packages of multimedia streaming it will be just few frames with duration less than second. For the user it's not a problem, because he still is able to recognize a sound or a video picture.

At the moment, the live multimedia streaming is replacing regular telecasting. It is more convenient to produce, store and share directly in digital form.

### 4.1.3 Real-time interactive audio/video

The real-time interactive audio/video is the latest type of streaming. It became possible after hardware became cheaper and got more computing power to handle multimedia stream transformations.

The real-time streaming is described by two main terms. The first term is end-to-end delay. End-to-End delay is the accumulation of transmission, processing, and queuing delays in routers; propagation delays in the links; and end-system processing delays (Kurose, 2010, 618).

The definition shows few reasons why delay arises during the whole process. The two of them affect more to the delay. There are the processing and the propagation delays in the links. The processing is process of coding and decoding of a signal. It takes the main part of a computing power and increases the delay. The propagation delays appear when the distance between a server and a client grows. If a distance grows then the delay also grows.

The acceptable levels of delay have some limits. For highly interactive audio applications, such as Internet phone, end-to-end delays smaller than 150ms are not perceived by a human listener; delays between 150 and 400ms can be acceptable but are not ideal (Kurose, 2010, 618).

Decreasing of delays is the main challenge in digital streaming technology. Skype, Facebook, Gmail and etc. use interactive transmissions for video/audio-chatting and conferences between their customers. Millions end-users are using one of those resources or more than one for connecting with parents, friends or co-workers around the world.

Surveillance systems transmit audio/video streaming from the streets, houses, buildings, banks daily. They can be used for real-time transmission from unreachable for human places by robot-platforms. Robot aircrafts, cars and humanoid robots are becoming part of our life. Usually, the video and the pictures give necessary information about many things. After nature catastrophes or accidents, the information about current state is extremely important. Many lives can be saved based on video/audio or pictures from those systems.

### 4.1.4 Comparison of streams

Each type of streaming has its own characteristics as shown in Table 1.

**Table 1. Comparing of streams tolerances**

| Streaming type | **Delay tolerant** | **Loss tolerant** |
|---|---|---|
| **Stored** | yes | no/yes |
| **Live** | yes | yes |

| | | |
|---|---|---|
| **Real-time** | no | yes |

Video may be captured and encoded for real-time communication, or it may be pre-encoded and stored for later viewing. Interactive applications are one example of applications which require real-time encoding, e.g. videophone, video conferencing, or interactive games. However real-time encoding may also be required in applications that are not interactive, e.g. the live broadcast of a sporting event (in other words it is live streaming). (Apostolopoulos, 2002, 3)

### 4.2 Compression

Audio/video (a/v) streaming needs plenty of resources are needed to handle a stream. For example a big buffer or storage is required to save temporally parts of a stream. Each pixel has a certain amount of data to describe its color. Pixels build the frame and frames build the video stream. Assume that the one pixel has 16 bits of a data. The normal size of a frame begins from 640 pixels x 480 pixels resolution.

640 * 480 * 16 = 4 915 200 bits = 4.6875 MBit per frame

The average rate of frames per second is from 15 to 25. It means that almost 100 MBit/sec is used by video stream. The 100MBit/sec bandwidth is typical standard for local networks. But the average Internet connection bandwidth is much lower varying typically from 1 Mbit/sec to 10Mbit/sec. Thereby, a/v streaming needs compression. For this purpose there exists many codecs for a/v streaming. MJPEG, H.261, H.263, H.264 are used codecs for that. The compression standards and specifics are vast. Many books and articles describe them. In other words, the codec modifies the original source to new compress by skipping redundant data.

The compression is important, nowadays. It helps to save bandwidth and storage space from overflowing. Video compression uses modern coding techniques to reduce redundancy in video data. Most video compression algorithms and codecs combine spatial image compression and temporal motion compensation. Video compression is a practical implementation of source coding in information theory. In practice most video codecs also use audio compression techniques in parallel to compress the separate, but combined data streams. (Video compression, [referred 19.04.2012])

### 4.3 Real-time protocol (RTP)

The Real-time Transport Protocol (RTP) and Real-time Control Protocol (RTCP) are IETF protocols designed to support streaming media. RTP is designed for data transfer and RTCP for control messages. (Apostolopoulos, 2002, 27)

The name of the protocol refers to the main aim of it. The aim of the protocol is to decrease network delay. RTP, typically, is extended UDP. At the sending side a media chunk is encapsulated into RTP packet, then encapsulated in a UDP, and then passed as the segment to IP. The receiving side do the same but in the opposite order. Then media chunk is decoded and rendered by a client's application.

The conventional approach for media streaming is to use RTP/UDP for the media data and RTCP/TCP or RTCP/UDP for the control. Often, RTCP is supplemented by another feedback mechanism that is explicitly designed to provide the desired feedback information for the specific media streaming application. Other useful functionalities facilitated by RTCP include inter-stream synchronization and round-trip time measurement. (Apostolopoulos, 2002, 27)

### 4.4 Streaming transferring methods

#### 4.4.1 Broadcast

The broadcast streaming has a long history. The first broadcasting stream was a radio streaming. After a radio streaming broadcast TV appeared. The main idea of broadcasting is transmitting or transferring stream with data to everyone and who wants to receive the data needs to switch on its receiver and tune it to certain frequency. The network broadcast has almost the same principle. The biggest difference between TV/radio broadcasting and a network broadcasting is that the broadcast transmitter sends data by an address instead of a frequency as in TV/radio broadcasting. As TV/radio broadcasting has the certain range of frequencies to broadcast many channels, a network broadcasting has a range of addresses. Figure 7 shows that the server transmits data to more than one client simultaneously. It is possible because the IP address specification has the rule about broadcasting and multicasting addresses. That

rule claims that the last address in an address range is the broadcast address. For example, if network has 192.168.100.0 – 192.168.100.255 range (with subnet mask 255.255.255.0), the broadcast address is 192.168.100.255.

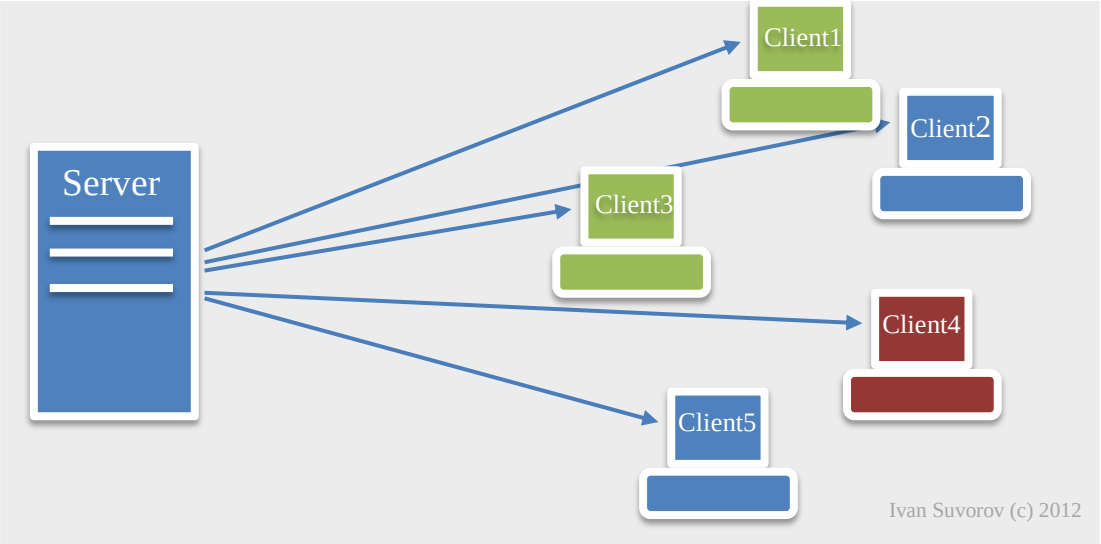


Figure 7. Broadcast streaming

When the server sends to a broadcast address data it is transferred to all clients in that network. However the broadcasting can be used only for local purposes. Routers which connect networks together do not route broadcast packets from one network to another. Therefor other methods are needed as a multicasting or unicasting streaming.

### 4.4.2 Multicast

As pointed out broadcasting is not an able to transfer data or stream to clients belong to different networks. Also it can be inefficient to broadcast the data to all clients. Instead an alternative concept called multicasting can be used. Figure 8 shows that the server tries to transfer data not to all clients in the network, but to some of them. IP address specification has special range of IP-addresses for multicasting. It is the range from 224.0.0.0 to 239.255.255.255.

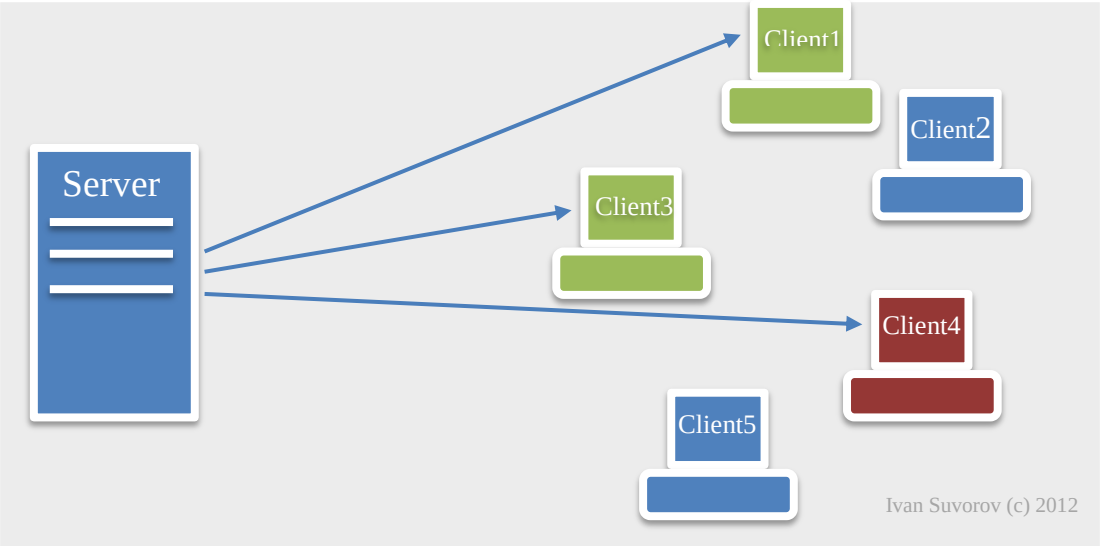


Figure 8. Multicast streaming

Multicasting is not limited to one network but it can be used also between networks. The network devices have support for multicast services. When multiple clients are requesting the same media stream, IP multicast reduce network resource usage by transmitting only one copy of the stream down shared links, instead of one per session sharing the link (Apostolopoulos, 2002, 28).

### 4.4.3 Unicast

Unicasting uses the common range of IPv4 (v6) addresses to transfer data. Unicasting is a common concept of IP network. As Figure 9 shows, that connection is established between Server and Client4. The main idea is concluded in that that data packet has one source IP address and one destination IP address to a target. Usually, the connection is established between two computers by network protocol (TCP, UDP, RTP and etc.). In that case data is transferred from a sender to a recipient.

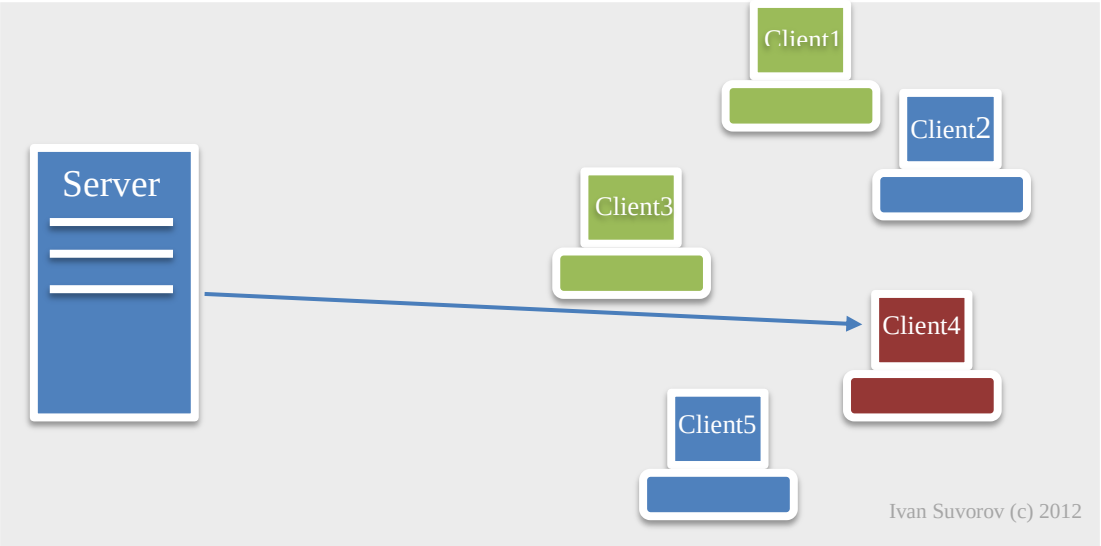


Figure 9. Unicast streaming

# 5 JAVA AND CROSS-PLATFORM SOLUTIONS

## 5.1 Java-platform

Many operation systems exists nowadays. Linux, Unix, Windows, Mac OS and even more operation systems have their own architecture, features and specifications. Each of them needs its own compiler to compile applications. Hundreds of different compilers exist just to convert algorithm to executable file. The wide spectrum of OSs and compilers means that program developers must adjust their programs for different platforms. It is hard and inefficient in many cases. For that purpose a new logic to design applications was established. That logic contains one more layer in whole process called virtual machine layer. That layer is an adapter to different platforms and it helps a lot to deploy application on a host machine.

Traditional way to make a program or an executable file from source code is described on Figure 10.

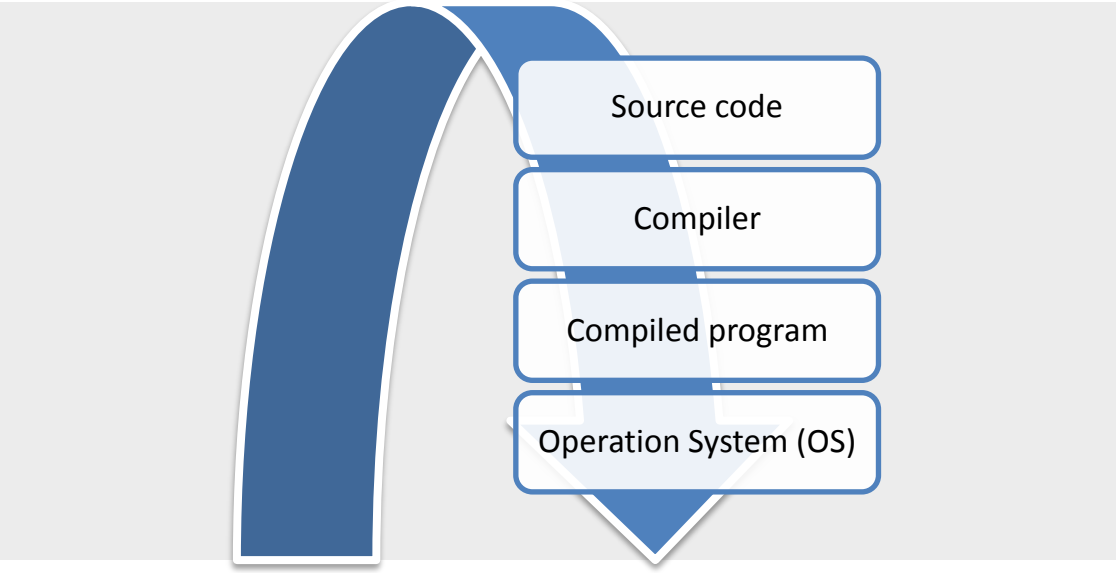


Figure 10. Traditional way to compile a program

Usually a source code is a text file with algorithm written in some programming language. A compiler relays both to the programming language and the platform on which the program will be running. From that point, a developer cannot write the algorithm just once and compile it for many platforms. The developer needs to write for a particular OS and use a particular compiler.

Java has one big difference compared with traditional approaches. There is Java Runtime Environment (JRE). Basically, it is a virtual machine within that precompiled application will be running. JRE has different versions for different OSs, but the byte code is the same.

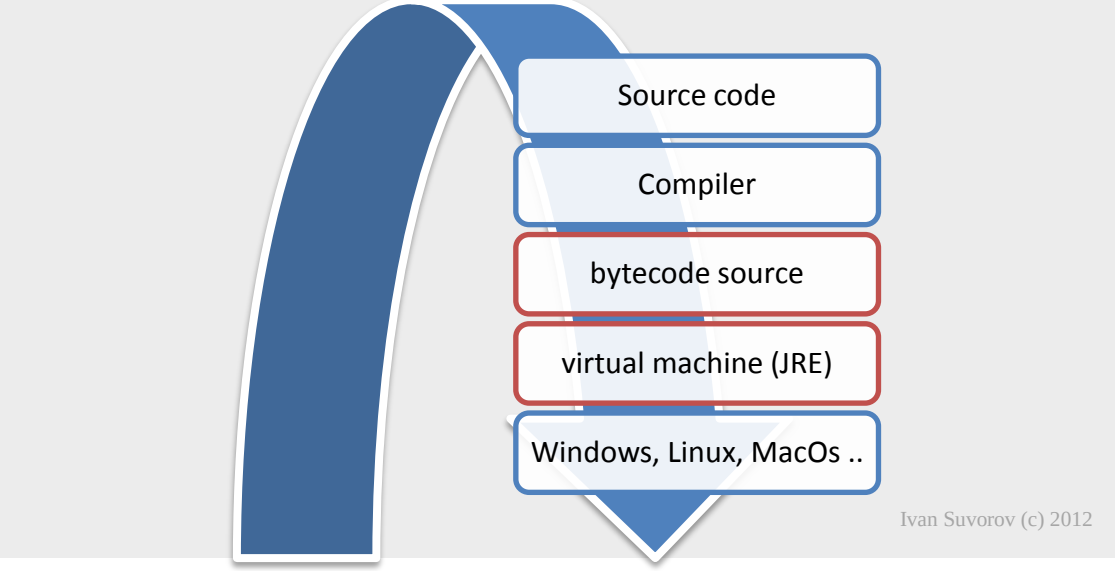


Figure 11. Java-platform way to compile a program

Figure 11 shows those two different steps on the compiling process. There are bytecode source and virtual machine (JRE) instead of a compiled program on Figure 10. In the case of Java, the compiler compiles the source code (.java) to the bytecode source (.class). Then, the virtual machine can execute a bytecode source, which is compiled just once and uses them on different platforms. In that case the difference is just in a preinstalled JRE.

The cross-platform application is an application which is able to run on different OSs. Thereby, Java-platform was developed to make cross-platform solutions. It turned out to be extremely suitable for enterprises. As de facto, the enterprise's systems consist of many technologies, platforms and have many purposes. Java-platform helps a lot with that.

## 5.2 Java Enterprise Edition (Java EE)

### 5.2.1 Introduction to Java EE

Java EE is extended from Java Standard Edition (Java SE). Libraries, modules and web-technologies were included there. The main aim of that edition is to solve the problem with deploying and managing multi-vendor information complex on enterprises.

### 5.2.2 JavaBeans

JavaBeans makes possible it to reuse software components. Developers can use software components written by others without having to understand their inner workings. (JavaBeans, [referred 19.03.2012])

Many components are written to extend the standard platform of Java to Java-platform for enterprise's purposes. The graphical components for a user interface and web-components were written under those technologies. That helps to save time and money in system design. Also it helps to separate a part of system and this separated part can then be implemented for example by some external company.

### 5.2.3 Java Server Pages (JSP)

Java Server Pages (JSP) is a Java-based server-side scripting language that allows static HTML to be mixed with dynamically generated HTML (Begg, 2010, 1030). Originally, HTML is a static source to a page layout. That is a big restriction to permanently varying information environment. With JSP, developers are able to write server-side script which describes not only the page layout, but also dependences between the page and other information resources dynamically. In that case, it's not a necessary to make a static page. It is possible just to write the basic structure and bind it with other dynamic resources such as databases, data about a current user's state, his information and etc., then generate the final page.

## 5.3 Web Technologies

Nowadays the web technologies have become an important part not only for enterprises, companies, governments and educational institutions. Also, it is widely used in regular activities of people in social life. Many people have an access to the Internet. Many services online are available for many purposes. However, the main customers of the latest web technologies are commercial organizations and nonprofit institutions like an open source community. There're many words what describe advantages of web technologies including collaboration, the optimization, the communication, the

integration and etc.  But the main goal of that is to make life easier and more comfortable.

### 5.3.1 HyperText Markup Language and Cascading Style Sheets

HyperText Markup Language (HTML) is the base of web user interfaces. HTML is an interpreted language. It describes the rules how to show the content to end-users. Cascading Style Sheets (CSS) is a style language which describes graphical rules for markup languages. Although both of these technologies are important for web designers. In my thesis I do not cover them in any more details.

### 5.3.2 JavaScript

JavaScript is an object-based scripting language and has become a standard dynamic web-pages and web applications (Begg, 2010, 992). JavaScript is a powerful tool for putting dynamic elements to a static web-page on a client side. Typically, a web user interface is static; even when it uses a server-side dynamic page generation. JavaScript is a client-side scripting language which can run in web-browser environment.

### 6.3.3 Asynchronous JavaScript and XML (AJAX)

Asynchronous JavaScript and XML (AJAX) is a web-technique for web-developers allowing to retrieve and send data without necessarily reloading the page. The original way to send and retrieve data is to send the URL-address with parameters to the server, and then the server sends its answer in a new HTML-page. It needs reloading, what is extremely inconvenient in many cases.

In case with AJAX, a client-side can send a request throughout the special object – XMLHttpRequest. It happens in the background without reloading a web-page. Interchanging of data can happen with not only XML-formatted data, but usually, with JSON-structured data.

AJAX-technique made a great impact in interactive applications on web. With AJAX it became possible to get a functional and a rich web user interface.

### 6.3.4 JavaScript Object Notation (JSON) and JSONP

A web client needs to interact with a server-side. JavaScript Object Notation (JSON) helps to send structured data either to a server or to a client. JSON is a lightweight data-interchange format (Introducing JSON, [referred 19.03.2012]).

JSON has the structure that is easy to read and easy to write. A json-object is created in a JavaScript environment and is loaded data. The structure of a json-object can be as rich as it needs.

JSONP works by making a <script> element (either in HTML markup or inserted into the DOM via JavaScript), which requests to a remote data service location (Defining Safer JSON-P, [referred 19.03.2012]). JSONP allows to establish cross-domain communication which is forbidden by AJAX-requests. This helps to design API and distribute it for third party applications or web-sites in the JSON unified format which is easy to understand and use.

## 6.4 Java Media Framework (JMF)

The Java Media Framework (JMF) is the multimedia Java sub-system. JMF enables to media content being added to a Java-application or an applet. Also JMF allows to capture, playback, stream and transcode multiple media formats. The JMF api-functionality allows capturing media stream from a source and transmitting it under RTP protocol among clients. The JMF an example application JMStudio to show its functionality. Figure 12 shows the user interface of JMStudio.


Figure 12. JMStudio user interface

JMStudio application has ability to capture a media source from different media sources and transmit it in a network or save it in a file system. Figure 13 shows the main menu of JMStudio with range of the functionality.

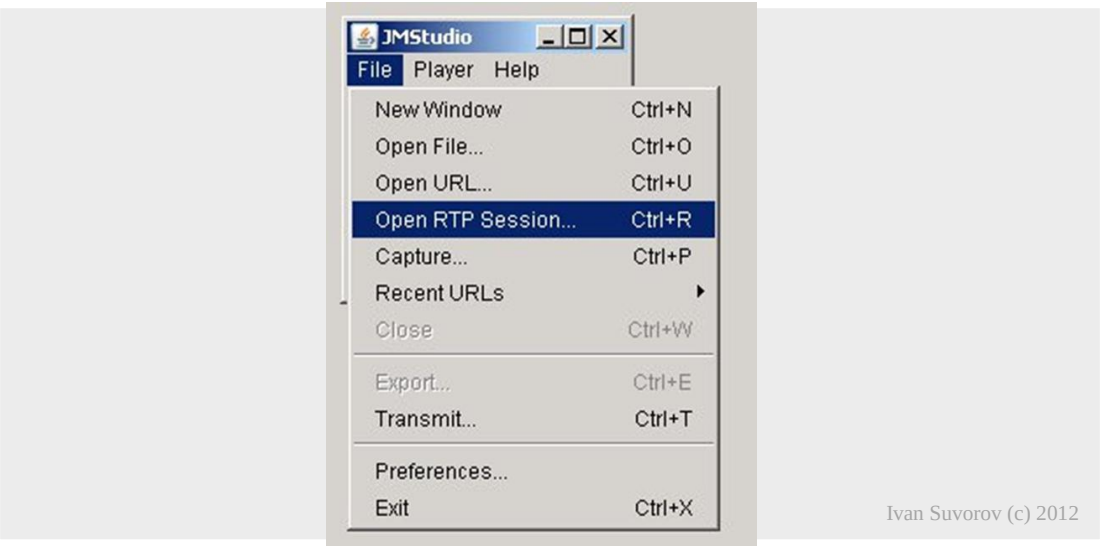


Figure 13. JMStudio main menu

# 6 DESIGN OF EXTENDABLE SERVER'S MANAGEMENT SYSTEM AND SERVERS

## 6.1 Basic design of the project

The main idea of the whole project is to control a robot-platform throughout a network. That requires many layers and modules to bind a client and a robot-platform together. Synchronization and an end-to-end delay, in that case, play a major part. Agility of sending makes controlling of the car smooth and precise.

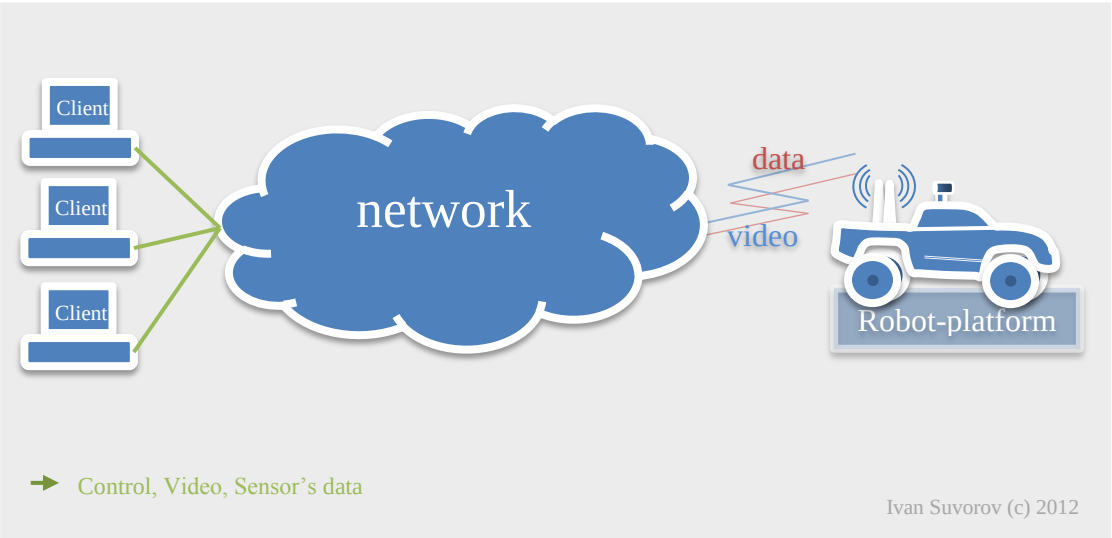


Figure 14. The basic concept of the project

Figure 14 shows the basic scheme of the original project concept. A client has access to the robot-platform throughout a network. There are the three important types of data: control data, video stream and data from the sensors on the robot-platform.

The control data is sent from the client side by the user to control the robot-platform. Left, right, forward, backward and other commands are encapsulated inside the control data flow. Each command needs to be delivered throughout a network with minimum delay and without losses. The target of this data is the robot-platform. The robot-platform gets data and runs commands which are encapsulated in that data flow.

Data from the sensors is sent from the robot-platform to the client side. The client can see what is happening nearby the robot-platform and analyse that data. Many sensors

are able to deploy on the robot-platform now. Among popular sensors are sensors as a GPS sensor, an ultrasonic and a laser sensors to measure the distance, a thermo-sensor and etc.

The third type is an audio/video (a/v) stream. From the robot-platform it is possible to transmit a/v stream throughout a wireless connection. It helps a lot, when a user needs to observe an environment around and makes a decision about the next step.

### 6.2 Design of the project

The design of Roboteh project was started from describing the main requirements. Figure 15 shows the concept of Roboteh project. In Figure 15 it is shown that the system has to receive/transmit data from/to the robot-platform Decartus by wireless connection. Data from Decartus is saved in to a database. A user can connect to MainServer and send control commands to Decartus through a network. Decartus is able to transmit a/v stream. MainServer is to be able to capture that stream and resend it to end-users. Admin panel allows managing the whole system through a web-page.

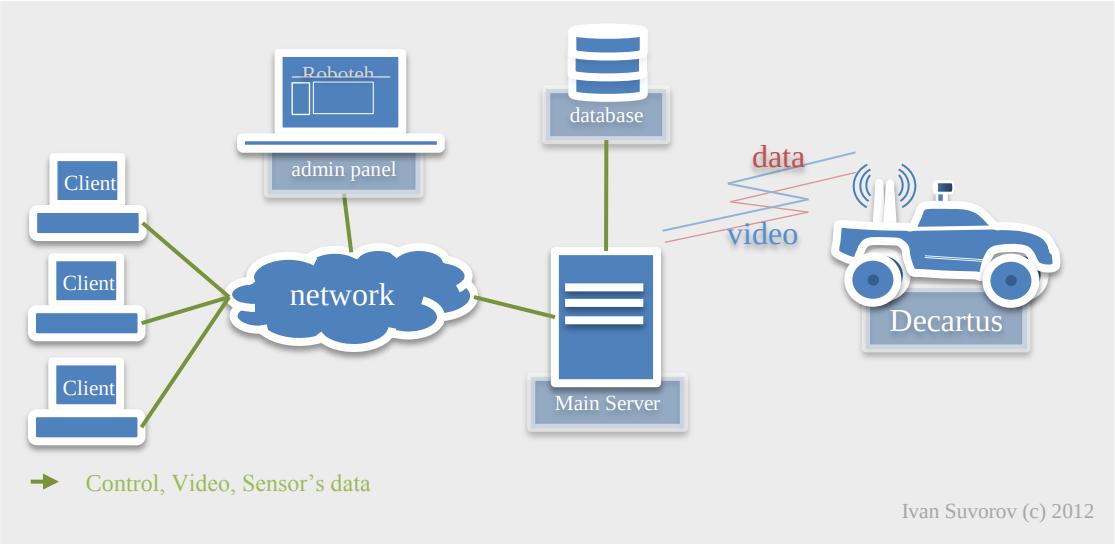


Figure 15. The basic scheme of the Roboteh project

As Figure 15 shows, the system must be able to manage one or more clients. It is the important requirement to have a flexible system. MainServer should be able to handle many clients simultaneously.

The next step is to divide the basic structure to several pieces. MainServer is divided into the two servers and one subsystem. As it is shown on Figure 16, there are Extendable Server's Management System, SeCoDa and RV_v2 servers.

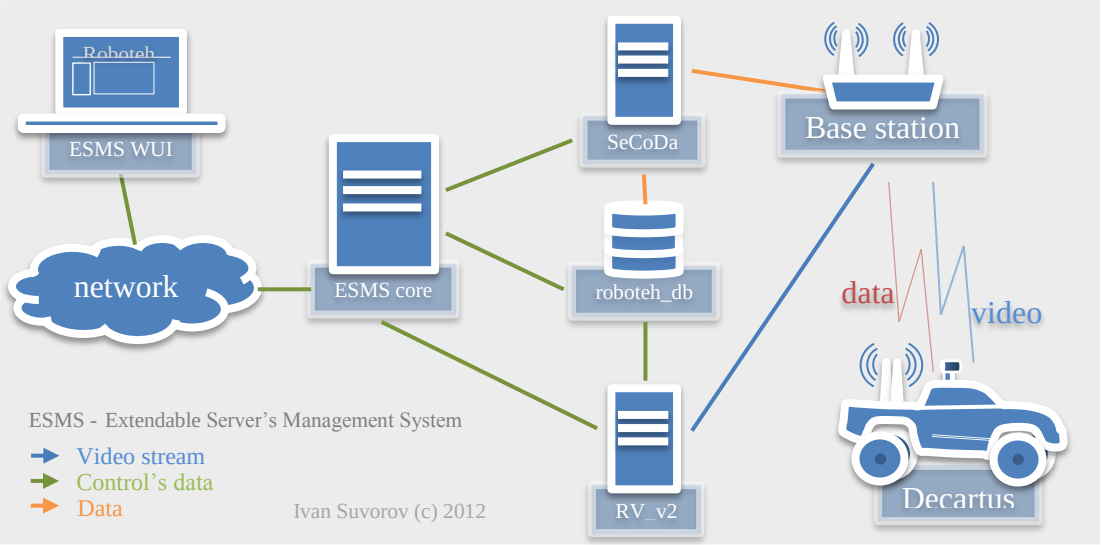


Figure 16. The main scheme of the Extendable Server's Management System with the servers for the Roboteh project

### 6.3 SeCoDa server

SeCoDa is the abbreviation from three words: Sensors, Commands and Data. These words describe its functionality. SeCoDa server is the main server which allows to communicate with the robot-platform (Decartus) through a serial port. As shown on Figure 17, the server has two main purposes. The first purpose is that the server has to get a control command packet from the client and send it to the robot-platform. The second purpose is SeCoDa server has to get data from the robot-platform from its sensors and put it into database.

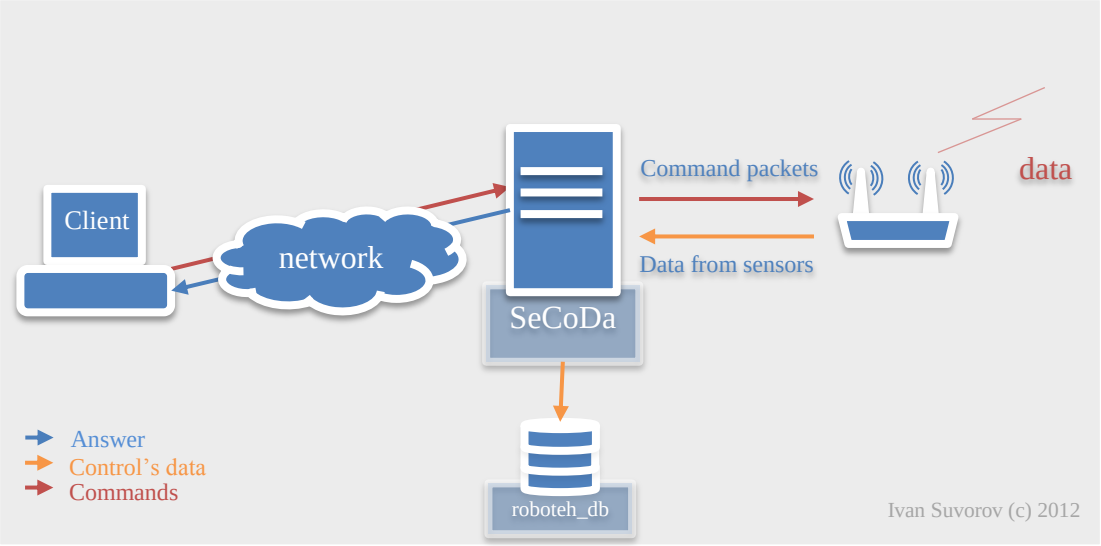


Figure 17. SeCoDa server activity

SeCoDa server should convert control packets from the client and send them to the robot-platform. It should convert data packets from the robot-platform and write it to the roboteh_db database. SeCoDa server should check availability of the robot-platform and if the connection is lost it should try to reconnect to the robot-platform.

### 6.4 RV_v2 server

RV_v2 (Roboteh Video v2) server takes video stream from the robot-platform and broadcast, multicast or unicast it to clients. As it is shown on Figure 18, RV_v2 server casts stream and takes configuration data from the database.

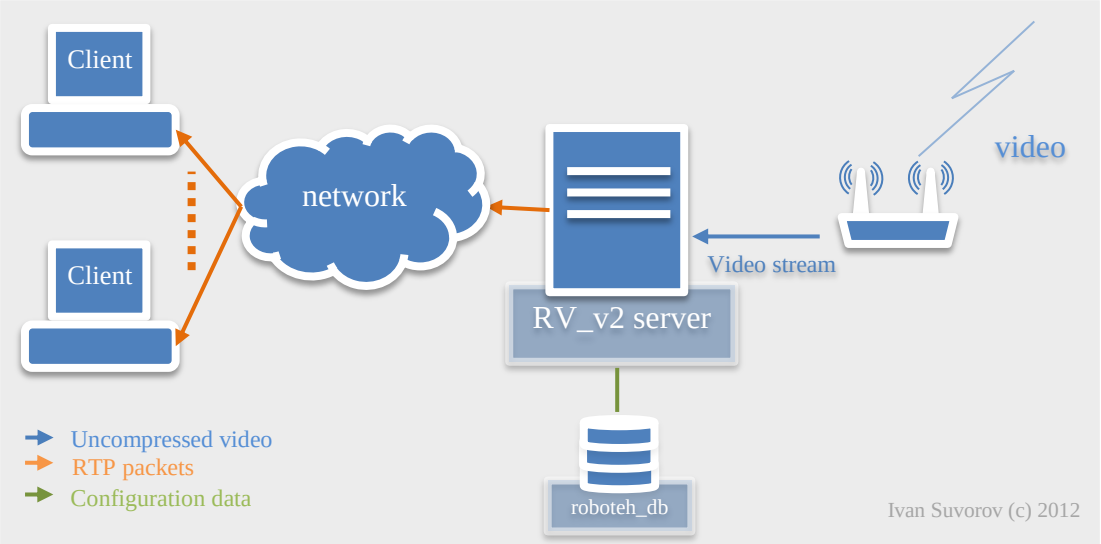


Figure 18. Video v2 server activity

Java Media Framework (JMF) was chosen as a platform for that server. The main reason of that is JMF allows developing a cross-platform application and making not only as stand-alone applications, but also as applets for a browser.

### 6.5 Extendable Server's Management System

Extendable Server's Management System (ESMS) provides a main functionality to control more than one server. Very seldom a web system is based on one server. Usually, a web-system has many servers to handle different services. As a rule, a complex web-system consists of more than one server and offers many services. There is a tough challenge to configure, collect information about current state of an each server and make simple procedures as start, stop or reset a server or a service for an administrator. Controlling servers from a centralized system gives and raises performance of the whole system. ESMS was made to avoid that problem.

The name refers to ability of the ESMS to extend its own functionality easily. That system could be implemented for other servers and services with minimum and simple changings.

The structure of ESMS consists of three components, which are described below.

- ESMS Web User Interface (ESMS WUI)
- WUI's Background  Services (WUI BS)
- ESMS core

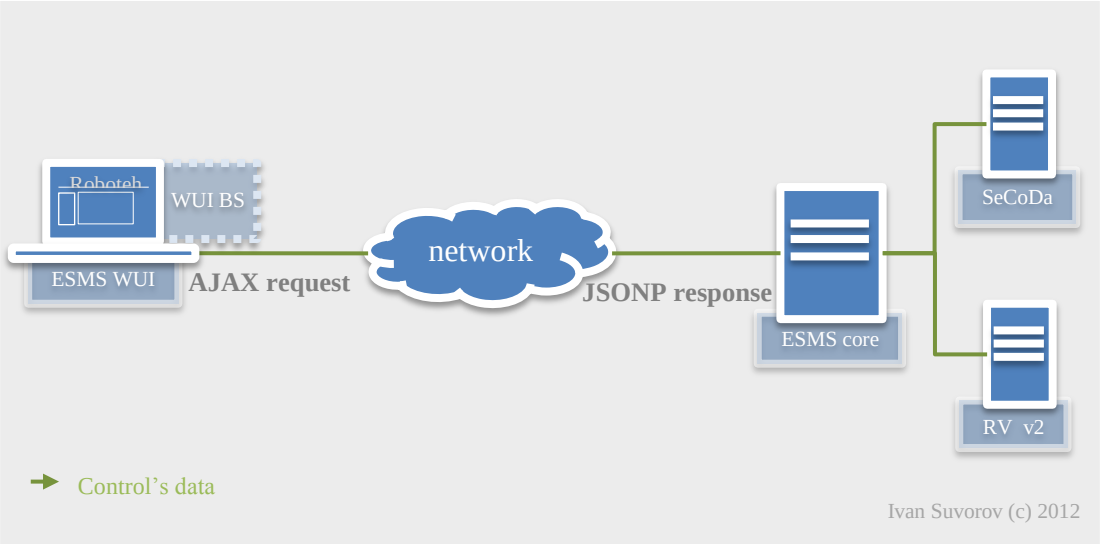


Figure 19. Extendable Server's Management System  and the servers

Figure 19 shows dependences between each of the components. WUI is a graphical representation of the rules described by scripting language XHTML.

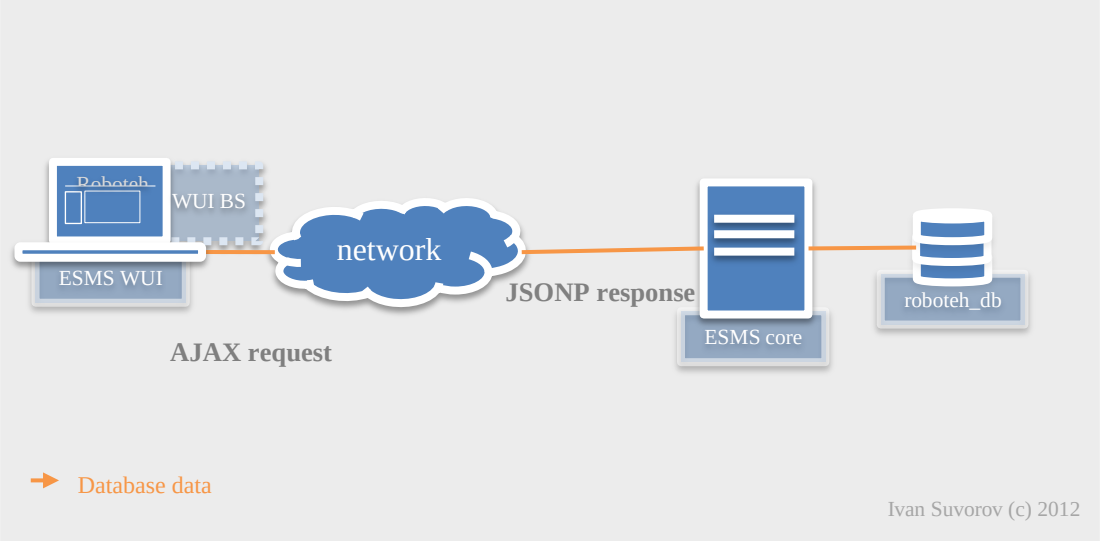


Figure 20. Extendable Server's Management System and database

Figure 20 shows relationship ESMS with roboteh_db database.

### 6.6 Database design

The *roboteh_db* database should contain configuration data, data from sensors and client's data. For that purpose the hardware and users schemes are made in the roboteh_db database. Figures 21 and 22 show schemes and tables of these schemes.

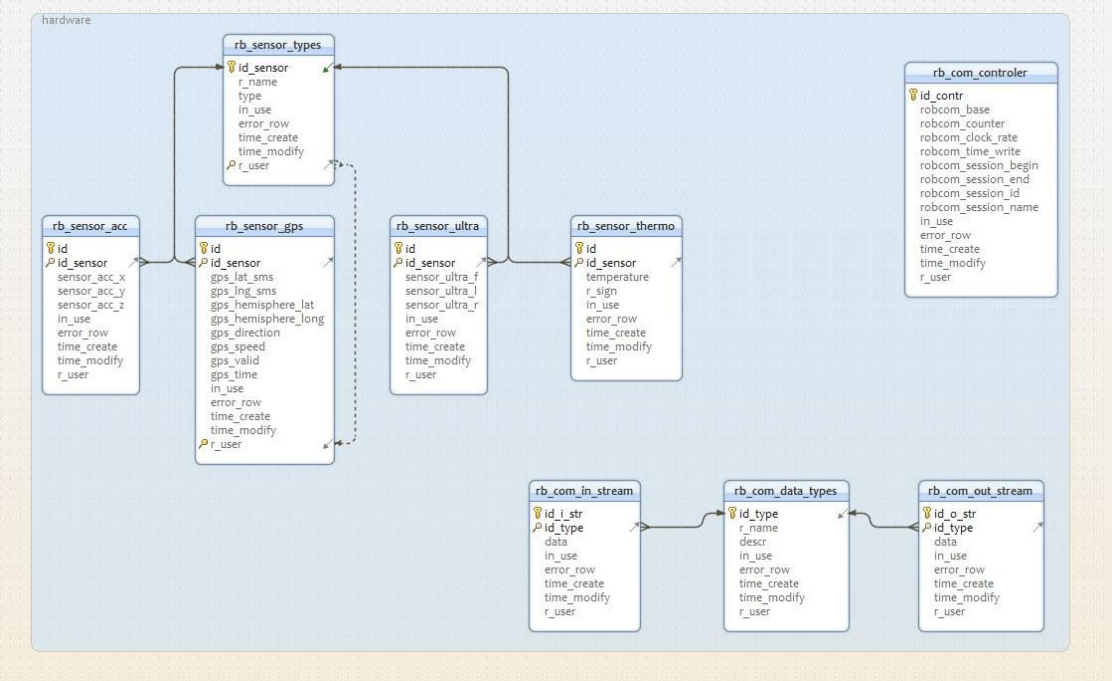


Figure 21. Database schema hardware

Figure 21 shows the hardware schema which consists of tables for data from sensors and for configurations for a com-port. SeCoDa server retrieves configuration for com-port from that schema. Then SeCoDa server makes call to its methods to insert captured data from the sensors to the roboteh_db database.

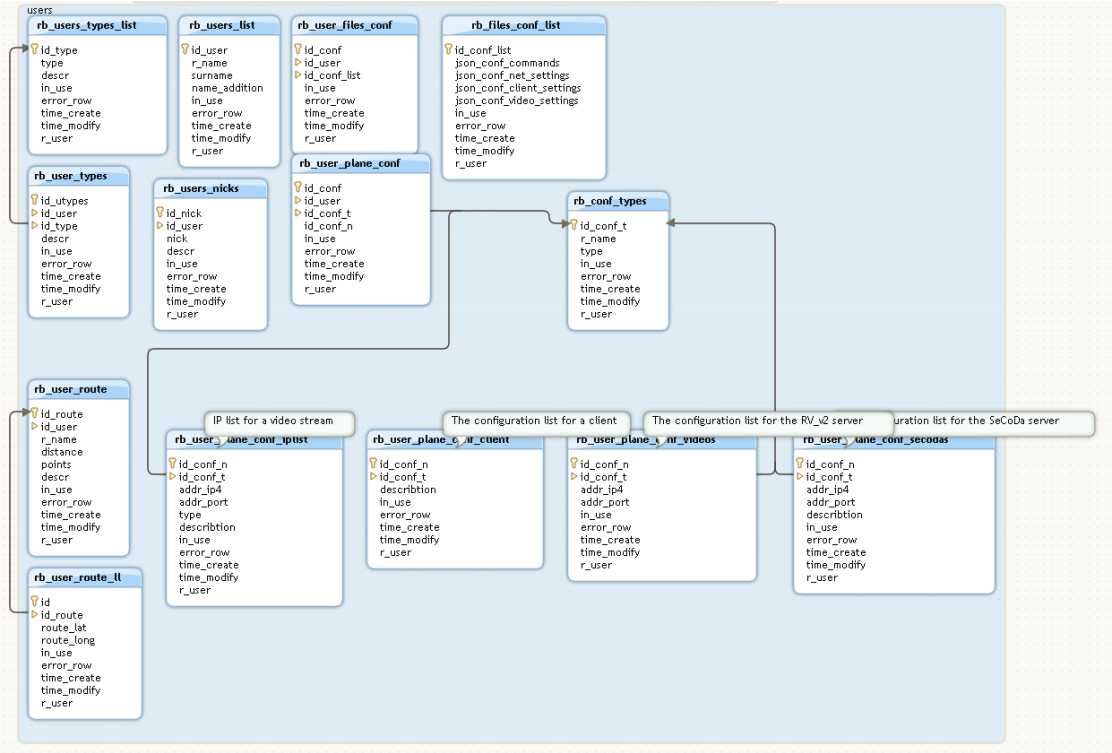


Figure 22. Database schema users

Figure 22 shows users schema which consists of tables which contain servers configuration information and user's data.

## 7 IMPLEMENTATION OF THE SERVERS AND ESMS

The implementation of a project like this raises many challenges. A wide range of OSs, developing tools, database systems exists nowadays, many technologies can improve the project and right selections can make the project successful.

Originally, the project was started under Windows XP OS platform. But after some time the critical nodes were decided to move to CentOS OS platform (Linux based OS). It increased the performance of the critical nodes like roboteh_db server (database) and SeCoDa server remarkably. Also ESMS was moved to CentOS in order to understand challenges with configuration and deploying a JavaEE project to a Web-server. Glassfish web-server was chosen as a part of Netbeans IDE for testing and Tomcat web-server was used in the operating enviroment.

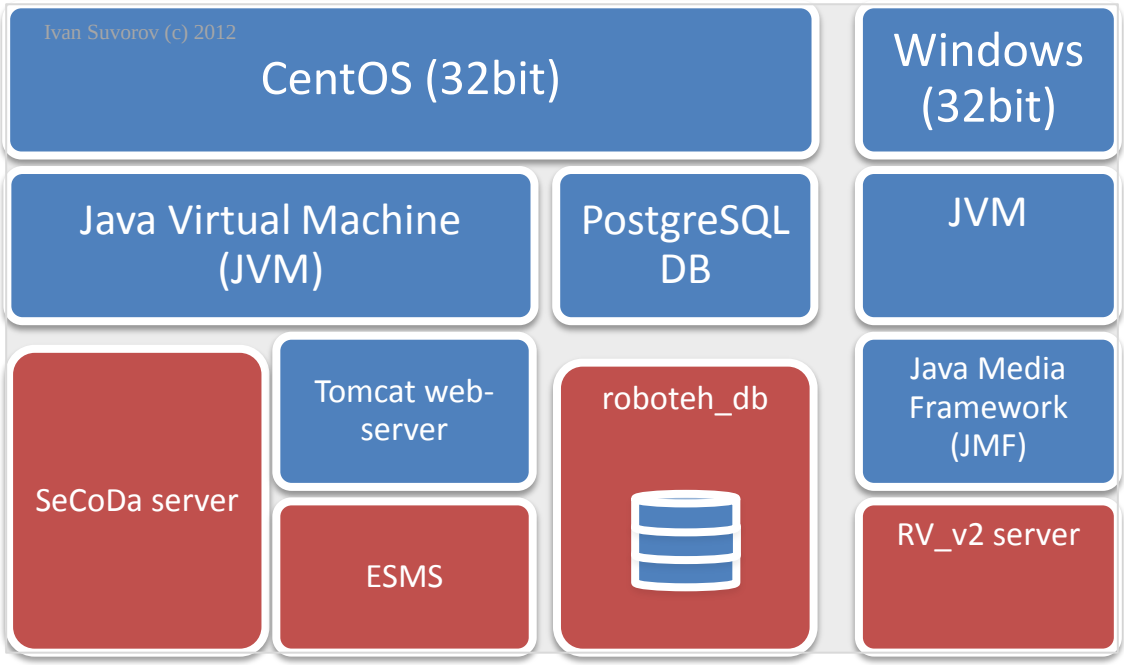


Figure 23. The schema of deployment of the servers

Figure 23 shows the main layers of the whole system and how it is deployed. As Figure 23 shows the system has two OSs CentOS and Windows XP. The hardware with preinstalled Windows XP OS is used for RV_v2 video server. Other critical nodes are put on CentOS OS platform.

## 7.1 Extendable Server's Management System (ESMS)

### 7.1.1 Web User Interface (WUI)

As explained in section 7.4, Extendable Server's Management System (ESMS) has three sub-systems. There are ESMS Web User Interface (ESMS WUI), WUI's Background Services (WUI BS) and the ESMS core. On Figure 24 is shown ESMS WUI.

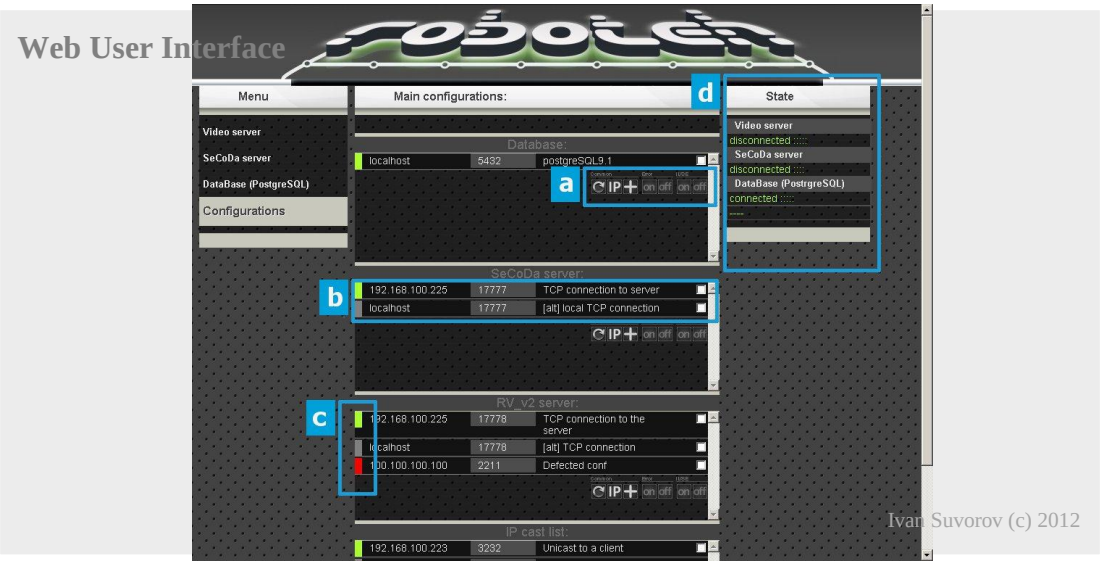


Figure 24. ESMS main WUI

On Figure 24 in the section “a” (coloured by blue colour) is shown the control panel for a table. That panel has three sections. First section is for “Common” commands as refreshing, getting current IP-address on a local host and inserting a new row into a table. Second section responses for updating “Error” state of a row in a table. The “on” button refers that that row is erroneous. The “off” button refers that that row is not erroneous. The third section responses for “iUse” state. In database this state is called “in_use”. That state signifies it a row is available to use or not. The “on” button puts a row in the available mode and the “off” button puts a row in the unavailable mode.

On Figure 24 in the section “b” is shown a table with configurations for servers. Each row has five sections. First section refers to a row’s mode. Second section is a column for an IP-address data. Third section is for Port-address data. Fourth section is for a description of an each row. Fifth section is a checkbox element to choose some certain rows.

These three modes are shown in the section “c” on Figure 24. A row has three modes in WUI. First mode is when a row is available to use and not erroneous. In that case this section is coloured in the green colour. If a row is not available to use and not erroneous, the mode section is coloured in the grey colour. And last row’s mode is the

red coloured. It happens when a row is erroneous.  By using checkboxes it is possible to apply some certain modes to more than one row.

On Figure 24 in the section “d” are shown status menu. That status menu has three sections. The “Video server” section shows a status of RV_v2 server. The “SeCoDa server” section shows a status of SeCoDa server. And the “Database (PostgreSQL)” section shows a status of the roboteh_db database. A status can be in two modes, “connected” or “disconnected” mode.

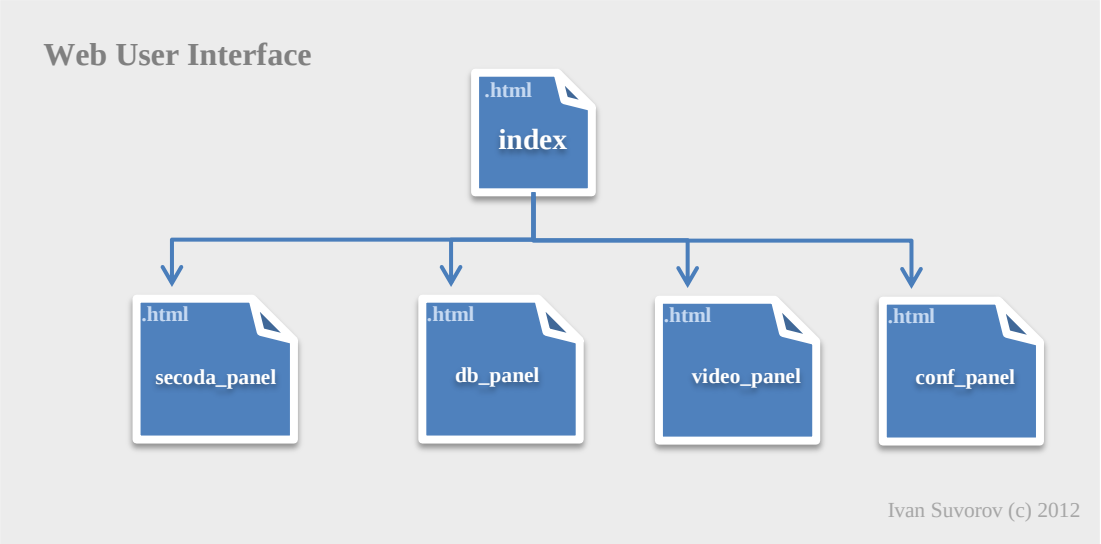


Figure 25. WUI files structure

As shown in Figure 25, the entry point of WUI is *index.html* file. That file contains XHTML script which describes layout of the main page. WUI has four sub-panels. Each of them describes its related configurations and activities for server.


Figure 26. SeCoDa server sub-panel

On Figure 26 is shown the sub-panel of the SeCoDa server configurations. SeCoDa and RV_v2 servers have three control buttons as “Start”, “Stop” and “Reset” button as shown on Figure 26 and Figure 27.The “Start” button sends AJAX-request to start a server. The “Stop” button sends AJAX-request to stop a server. And The “Reset” button sends AJAX-request to stop and then to start a server.

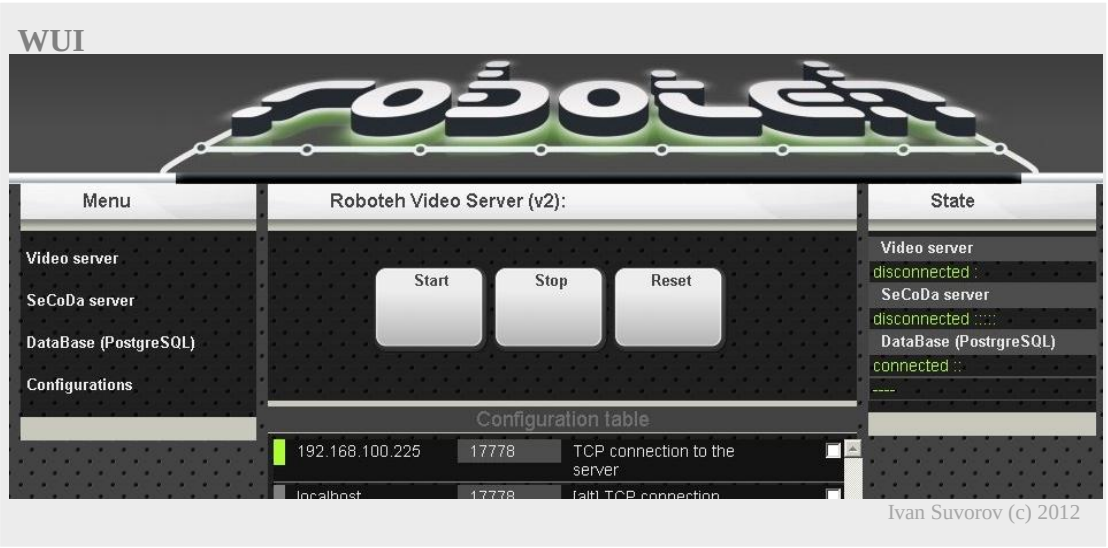


Figure 27. RV_v2 server sub-panel

On Figure 28 is shown what happens when “+” button in “Common” section of the table menu is pressed. Three editable inputs appear. The button on the right side responses to accept changes and inserts a new row in the table.


Figure 28. Insert functionality

On Figure 29 is shown modification functionality of the WUI. A double click on a cell allows you to modify that cell. Changes are accepted by pressing the entry key.

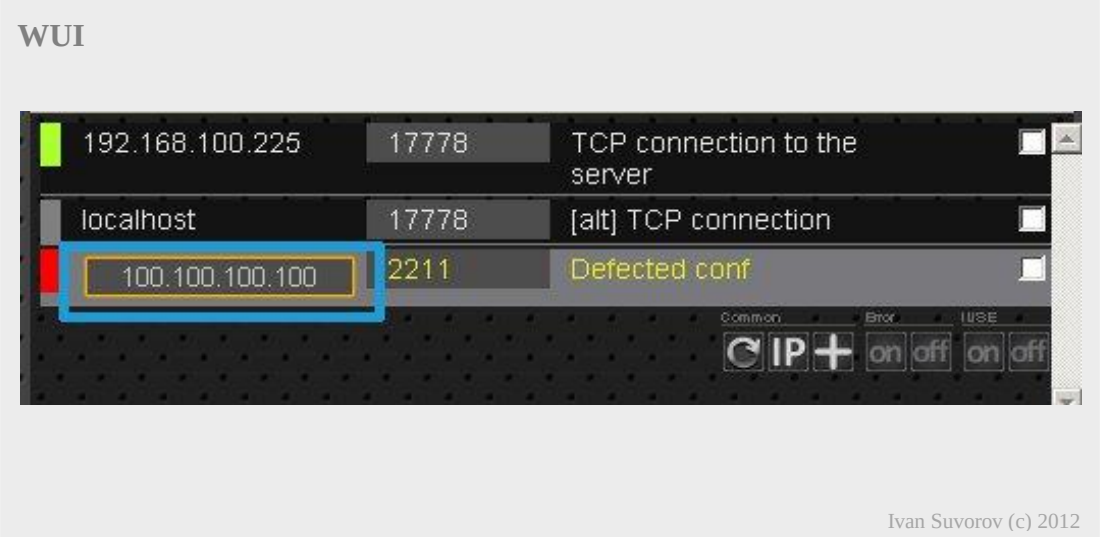


Figure 29. Modification functionality

### 7.1.2 Web User Interface Background Services (WUI BS)

Web User Interface Background Services (WUI BS) consists of dynamic services which work as background services. JavaScript is the main engine or the platform for those services. The *robo_admin_control.js* file contains the JavaScript (JS) script for BS. BS requests are sent through AJAX requests with parameters. Fetching, inserting, modification data are allowed by WUI BS. The functionality described above is based on WUI BS.

As Figure 30 shows, a request is sent from JS-script inside *robo_admin_control.js* file. That script is processed by browser. The entry point of that activity is the *get_jsonp_api()* function. Figure 31 shows the consecutive process of function callings. After *get_jsonp_api()* is called, it calls the *connect_ajax()* function to initialize a request through the JS-object XMLHttpRequest (XHR). The *url* parameter *connect_ajax()* function consists of two parts. The first part is URL-address of jsp-file and the second part consists of parameters with a value. When *connect_ajax()* has got a response, it calls JS-function *eval()* with a text of the response as a parameter. A response is JSONP formatted answer. The *eval()* function with JSONP-formatted parameter calls a function which is described in the response.

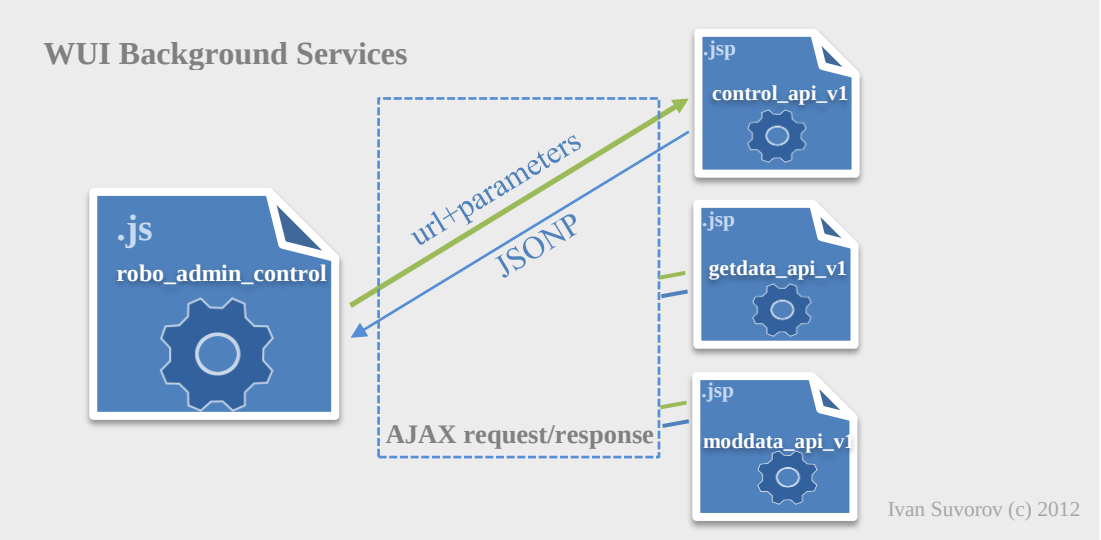


Figure 30. WUI Background Services activities

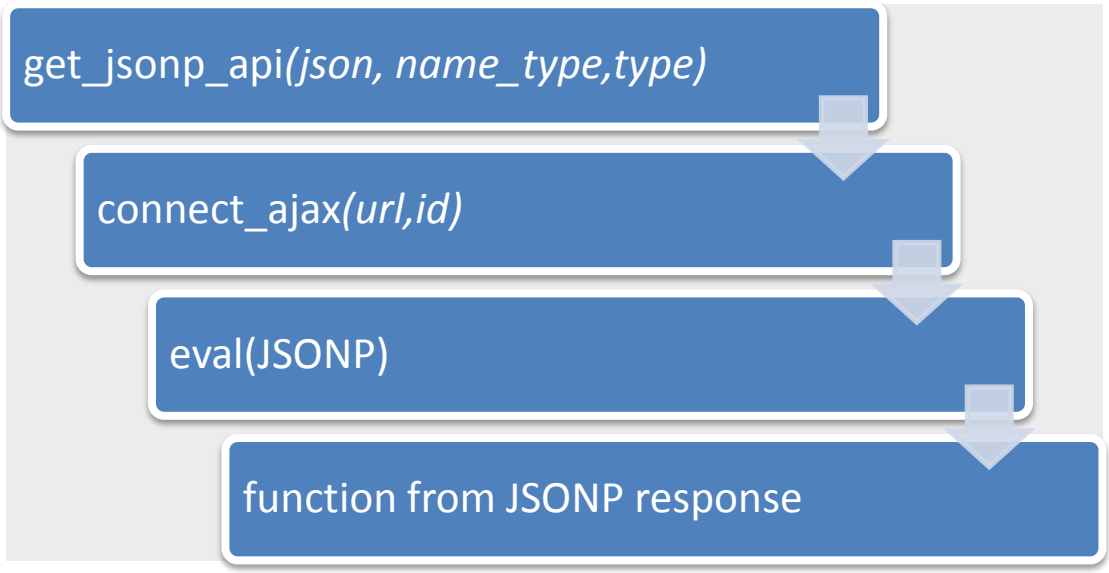


Figure 31. API core functions relation

The url-address with parameters below shows an example of the AJAX-request to modify the port number in the SeCoDa server table.

```
http://localhost:8080/esms/db_moddata_api_v1.jsp?db_mod=secodas_conf&
ctype=port&id=1&value=17777
```

The url-address with parameters below shows an example of the AJAX-request to retrieve the RV_v2 server configuration table.

```
http://localhost:8080/esms/db_getdata_api_v1.jsp?db_conf=videos_conf
```

The AJAX-response for that request is shown below. A response for Roboteh API-functions is sent in the JSONP format.

```
jsonOUTPUT(
    {
    "table":
            {
            "rows":
                [
                    {
                    "id" : 7,
                    "IP" : "192.168.100.225",
                    "port" : 17778,
```

```
                    "describe" : "TCP connection to the server",
                    "in_use" : 1,
                    "error_row" : 0
                    },
                    {
                    "id" : 8,
                    "IP" : "localhost",
                    "port" : 17778,
                    "describe" : "[alt] TCP connection",
                    "in_use" : 0,
                    "error_row" : 0
                    },
                    {
                    "id" : 9,
                    "IP" : "100.100.100.100",
                    "port" : 2211,
                    "describe" : "Defected conf",
                    "in_use" : 1,
                    "error_row" : 1
                    }
                ],
                "table_info":
                    {
                    "len":3,
                    "type":"videos_conf"
                    }
            }
    }
);
```

### 7.1.3 Roboteh Application Programming Interface (Roboteh API)

The *connect_ajax(url,id)* function handles AJAX requests and responses. ESMS answers to WUI BS in the JSONP-format. That is showed in Figure 19 and Figure 31. In Figure 30 three jsp-files are shown. There are *control_api_v1.jsp, getdata_api_v1.jsp* and *moddata_api_v1.jsp*. These jsp-files are made to handle parameters from the request and call a necessary function from a JavaBean-object in the ESMS core.

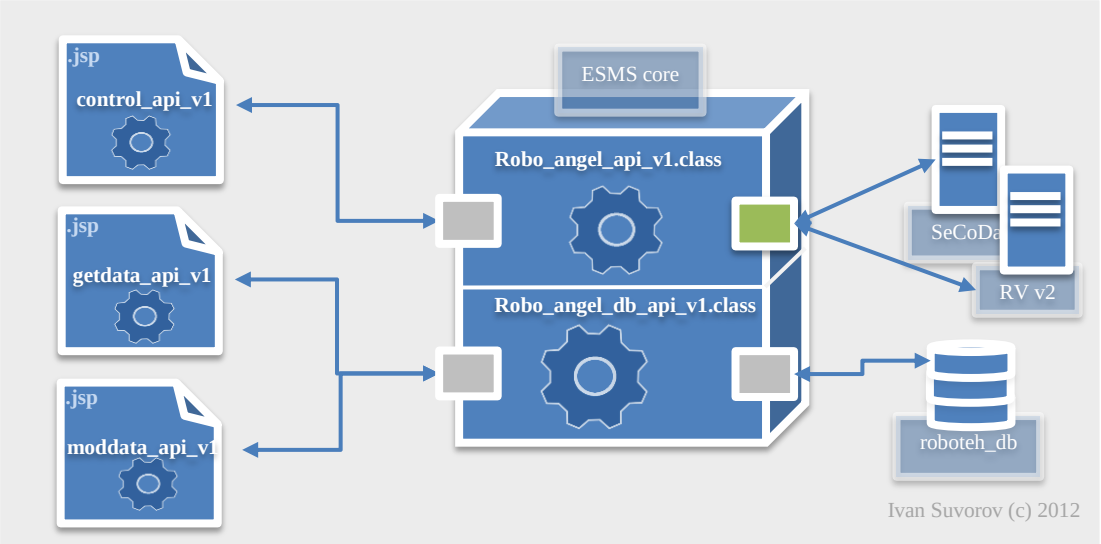


Figure 32. ESMS core activities

As shown in Figure 32, the *control_api_v1.jsp* server-page initializes JavaBean-object *Robo_angel_api_v1.class*. The *getdata_api_v1.jsp* server-page and the *moddata_api_v1.jsp* server-page initialize JavaBean-object *Robo_angel_db_api_v1.class*. The JSP-files calls methods from JavaBean-object dependants on received parameters. The list of parameters is shown in Table 2.

**Table 2. The list of the Roboteh API parameters**

| URL-parameter | Value | JSONP call back | JavaBean class |
|---|---|---|---|
| | ***control_api_v1.jsp*** | | |
| **control** | start_secoda | jsonSTATE | Robo_angel_api_v1 |
| | stop_secoda | | |
| | reset_secoda | | |
| | check_secodas | | |
| **control** | start_vserver | jsonSTATE | |
| | stop_vserver | | |
| | reset_vserver | | |
| | check_rvs_v2 | | |
| | ***db_getdata_api_v1.jsp*** | | |

| | | | |
|---|---|---|---|
| **db_conf** | cast_ip_list | jsonOUTPUT | Robo_angel_db_api_v1 |
| | secodas_conf | | |
| | videos_conf | | |
| **db_data** | sensor_ultra_last | jsonDATA | |
| | sensor_gps_last | | |
| | route_list | | |
| | route, value | | |
| | **db_*moddata_api_v1.jsp*** | | |
| **db_mod** | id, value, ctype, type | | Robo_angel_db_api_v1 |
| **db_insert** | new_secodas_conf, id, value, ctype, type | | |
| | new_videos_conf, id, value, ctype, type | | |
| | new_cast_ip_list, id, value, ctype, type | | |

The typical structure of an url-address and a parameter has this form: "*http://domain.domain/some.jsp?parameter=value&parameter_2=value_2*". As Table 2 shows, there are the three main JSP server-pages to control servers as well as send and get data from DB.

The *control_api_v1.jsp* server-page handles requests which are related to controlling of the servers. The requests to stop, start, reset and get the current state of a server are handled by its logic statements.

The db_*getdata_api_v1.jsp* server-page handles requests which are related to collection of information from the Database.

The db_*moddata_api_v1.jsp* server-page handles requests which are related to inserting and modification of information into the Database.

A response in JSONP-format allows use Roboteh API-functions not only inside one domain but also use the cross-domain communication. In other words these Roboteh API functions can be use by third party web-resources, not only in one domain. The "Web Control System of Mobile Platform" part also uses these Roboteh API functions which were designed specially for that part.

Table 3 shows Extendable Server's Management System (ESMS) files to deploy and its description.

**Table 3. Extendable Server's Management System files with description**

| Path | File name | Description |
|---|---|---|
| | *esms.war* | Pre-deploy WAR file with ESMS project |
| **%esmsroot%\build\web\WEB-INF\classes\org\roboteh\angels** | | |
| | Robo_angel_api_v1.class | The JavaBean class with methods to control servers |
| | Robo_angel_db_api_v1.class | The JavaBean class to establish communication with roboteh_db database |
| **%esmsroot%\build\web\WEB-INF\classes\org\roboteh\local** | | |
| | en_point | Configuration to connect to the roboteh_db database |
| | read_conf_files.class | Class with methods to read configuration from a configuration file. |
| **%esmsroot%\build\web** | control_api_v1.jsp | Parsing "Control Roboteh API" |
| | db_getdata_api_v1.jsp | Parsing "Get Roboteh API" |
| | db_moddata_api_v1.jsp | Parsing "Set Roboteh API" |
| | index.html | Entry web point |
| | robo_admin_control.js | WUI Background Services |
| | robo_style.css | |
| **%esmsroot%\build\web\sub_panels** | conf_panel.html | Sub-panel to summarize configurations |

| | | |
|---|---|---|
| | db_panel.html | Database configurations |
| | secoda_panel.html | SeCoDa configuration panel |
| | video_panel.html | RV_v2 configuration panel |
| **%esmsroot%\build\web\images** | | |
| **%esmsroot%\build\web\WEB-INF\lib** | postgresql-9.0-802.jdbc3.jar | Library to establish communication with PostgreSQL database. |

### 7.2 SeCoDa server

SeCoDa server is the main server in the project. The name is derived from the main activities of the server. The SeCoDa abbreviation means Sensors Commands and Data. All commands and data from the sensors go through SeCoDa server and reach the destination through certain interfaces. Figure 33 shows that the server has the three main interfaces. They are Socket, Database (DB) and serial port interfaces which respond for network connections, data storing/retrieving and communication with the robot-platform through a serial port, respectively. Each interface has a separated thread for execution.

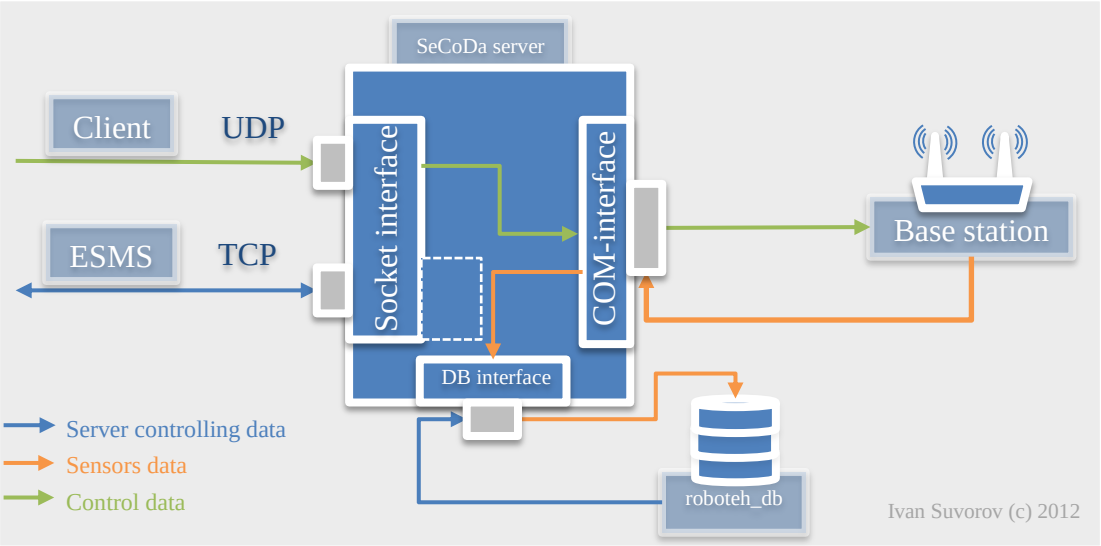


Figure 33. SeCoDa server structure

COM-interface takes its name from the type of the serial-port which communicates the server and the robot-platform. Java-platform does not have original libraries to use

parallel and serial ports. Special third party libraries from RXTX are therefore needed. The official web-site for that is http://rxtx.qbang.org. That web-site contains the archive with libraries for different platforms. SeCoDa server uses the version for Windows and Linux platforms. The *rxtx-2.2pre2-bins.zip* contains folders with native libraries and the library for Java-platform in the root folder. The library for Java-platform is called RXTXcomm.jar and the native library for Windows and Linux are called *rxtxSerial.dll* and librxtxSerial.so, respectively. However, the *rxtx-2.2pre2-bins.zip* file contains libraries for other platforms as, for example for MacOS and Solaris OS.

COM-interface is implemented by *comm_connector* class and *SerialPortEventListener* class from *RXTXcomm.jar* library. *Comm_connector* class is based on *SerialPortEventListener* class and has its methods to catch port events. Also, it uses *CommPortIdentifier* and *SerialPort* classes to establish communication with the com-port. The code below shows the main method to catch data from a serial port.

```
public void serialEvent(SerialPortEvent spe) {
        switch(spe.getEventType()) {
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[70];
            b_lost_con = false;
            try {
                while (inp_stream.available() > 0) {
                    int numBytes = inp_stream.read(readBuffer);
                                System.out.println("Buffer = "+(new
String(readBuffer)));
                                try {
                                    this.setCommands_buffer(readBuffer);
                                } catch (SQLException ex) {
                                    Log-
ger.getLogger(com_connector.class.getName()).log(Level.SEVERE, null,
ex);
                                }
                                readBuffer = new byte[20];

                }
                 System.out.println("-------");

            } catch (IOException e) {System.out.println(e);}
            break;
        }
    }
```

Com-port has the main event DATA_AVAILABLE. When this event occurs the com-port has data which can be read. *SerialPort* class has an *OutputStream* object to send

data to com-port. Before com-port can be used, it must be configured. Basic configuration is shown in Table 4. These parameters are put by *setSerialPortParams* method from *SerialPort* class. The code below shows using of that method.

```
com_port.setSerialPortParams(this.getI_com_speed(),
                             SerialPort.DATABITS_8,
                             SerialPort.STOPBITS_1,
                             SerialPort.FLOWCONTROL_NONE);
```

**Table 4. Configuration of serial port**

| Parameter | Value |
|---|---|
| **Port Name** | COMn (Windows) or /dev/ttyUSBn (*nix and it's for UART) |
| **Bound Rate** | 38400 |
| **Data bits** | 8 |
| **Stop bits** | 1 |
| **Flow control** | NONE |

The interaction between SeCoDa server and robot-platform has its own specific features and algorithms. Robot-platform sends two types of data. The first type is request controlling data from the server. This is "r" letter which is sent to the server. When the server has caught the request, it sends the packet with control data. If the server does not have any data from robot-platform for 800ms, it tries to send the default control packet "%C0dNN." and write to the system output the notification "lost connection". The code below shows the implementation of the "lost connection" method.

```
tt = new TimerTask() {
        @Override
        public void run() {
            if (b_lost_con)
            {
                System.out.println("lost connection");
                 try {

com_connector.this.com_write("%C0dNN.".getBytes());
                } catch (IOException ex) {
                    Log-
ger.getLogger(com_connector.class.getName()).log(Level.SEVERE, null,
ex);
                }
            }
            b_lost_con = true;
        }
```

```
    };
timer = new Timer();
timer.scheduleAtFixedRate(tt, 0, 800);
break;
```

A control packet has “12345678” structure which is described in Table 5.

**Table 5. The structure of a controlling packet from a client**

| Position | Value | Definition |
|---|---|---|
| 1 | % | Beginning of the packet |
| 2 | C | Control data |
| 3 | [L,0,R] | Position of wheels<br>L - turn left<br>R – turn right<br>0 – default position |
| 4 | [ASCII] | Speed(ASCII code, in decimal form range is from 1 - 200) |
| 5 | [L,N,R,S] | Position of the camera by X-axis<br>L - turn left<br>R – turn right<br>N – default position<br>S – hold current position |
| 6 | [U,N,D,S] | Position of the camera by Y-axis<br>U - turn up<br>D – turn down<br>N – default position<br>S – hold current position |
| 7 | . | Ending of the packet (dot) |
| 8 | [ASCII] | Checksum |

The second type of data from the robot-platform is data from the sensors. Data is divided to two different packets with their own formatted structures which are shown in Table 6 and Table 7. Table 6 describes the structure of packet which contains data from the four sensors. There are left, front, right ultra-sonic sensor and the thermo-

sensor. That packet is called Ultra-Thermo (UT) data packet and has "%D,LLL,FFF,RRR,SDD." format.

**Table 6. The structure of UT data packet**

| Position | Value | Definition |
|---|---|---|
| | % | Beginning of the packet |
| | D | Data from sensors |
| **LLL** | [000-400] | Distance from left ultra-sonic sensor |
| **FFF** | [000-400] | Distance from front ultra-sonic sensor |
| **RRR** | [000-400] | Distance from left ultra-sonic sensor |
| **SDD** | [S {-,+} DD{00-99}] | Data from thermo-sensor |
| | . | Ending of the packet (dot) |

Table 7 describes the second type of data-packet from the robot-platform which contains data from the GPS sensor (NMEA data, [referred 19.03.2012]). GPS data packet structure has elaborate structure with more than one parameter. GPS data packet has the following structure.

```
"$TYPE,HHMMSS,S,LLLL.LLL,H1,LLLLL.LLL,H2,SP,TA,DDMMYY,MV,MVD,CHS"
```

**Table 7. GPS data packet**

| Position | Value | Definition |
|---|---|---|
| | $ | Beginning of the packet |
| **TYPE** | GPRMC | GPS (Recommended Minimum sentence C) |
| **HHMMSS** | hhmmss | UTC time from the GPS |
| **S** | [A,V] | Status A=active or V=Void. |
| **LLLL.LLL** | dddd.ddd | Latitude |
| **H1** | [N,S] | North / South hemisphere |
| **LLLLL.LLL** | ddddd.ddd | Longitude |

| | | |
|---|---|---|
| **H2** | [E,W] | East/ West hemisphere |
| **SP** | float | Speed over the ground in knots |
| **TA** | float | Track angle in degrees True |
| **DDMMYY** | ddmmyy | Date |
| **MV** | float | Magnetic Variation |
| **MVD** | [W,E] | Magnetic Variation direction (West or East direction) |
| **CHS** | [ASCII] | The checksum data, always begins with * |

After the server has caught data from sensors it sends it to a method from *db_connect* class. The type of method depends on the type of data packet from the robot-platform. The *db_insert_ultra_imp* method inserts data from the ultra-sonic sensors. The *db_insert_thermo_imp* method inserts data from the thermo sensor. The *db_insert_gps_imp* method inserts data from the GPS sensor. The UT data packet has ultra-sonic sensors data and thermo sensor data. That packet is put to *db_insert_ultra_imp* method and from that method called *db_insert_thermo_imp* method. These methods insert data into roboteh_db database. Inside these methods data packets are parsed by the pieces and the parsed data is inserted to the roboteh_db database.

Table 8 shows SeCoDa server files to deploy and its description.

**Table 8. SeCoDa server files**

| Path | File name | Description |
|---|---|---|
| **%SeCoDaRoot%** | SeCoDa_server.jar | The SeCoDa server core |
| **%SeCoDaRoot%\lib** | postgresql-9.0-802.jdbc3.jar | The P-SQL DB library |
| **%SeCoDaRoot%\lib** | RXTXcomm.jar | |
| **%SYSTEMROOT%\System32** | SerialPort.dll | Win32 Serial Port native library |
| **$JAVA_HOME/lib/i386** | SerialPort.so | *nix Serial port native library |

### 7.3 RV_v2 server

RV_v2 is the video server which provides video transmission from the robot-platform Decartus towards the end-users. The main transferring protocol for that purpose is the RTP protocol. The core of RV_v2 server is the RV_v2.jar file. Additionally, the server needs libraries, which provide compatibility with the operation system. Java platform offers additional framework to build multimedia applications. It's called Java Media Framework (JMF). The core of JMF consists of JMF library called *jmf.jar* and native libraries. The list of native libraries is shown in Table 9 which shows RV_v2 server files to deploy and its description.

**Table 9. RV_v2 server files**

| Path | File name | Description |
|---|---|---|
| **%SeCoDaRoot%** | RV_v1.jar | The RV_v2 server core |
| **%SeCoDaRoot%\lib** | postgresql-9.0-802.jdbc3.jar | The P-SQL DB library |
| | jmf.jar | |
| **%SYSTEMROOT%\System32** | jmacm.dll jmam.dll jmcvid.dll jmdaud.dll jmdaudc.dll jmddraw.dll jmfjawt.dll jmg723.dll jmgdi.dll jmgsm.dll jmh261.dll jmh263enc.dll jmjpeg.dll jmmci.dll jmmpa.dll jmmpegv.dll jmutil.dll jmvcm.dll jmvfw.dll jmvh263.dll jsound.dll | Win32 Video/Audio native libraries |

The *jmf.jar* should be imported into project before deploying. The native libraries are placed into the system folder of the OS. In the project RV_v2 server's platform uses Windows platform and the native libraries are placed into the *%SYSTEMROOT%\windows\system32* folder.

*RobotehVideoServerV2* is the entry point class to the server. That class has all necessary methods to establish video transmission. The main classes and the objects which were created from those classes are listed below.

```
MediaLocator media_locator;
Processor media_processor;
DataSink dsink_trasmitter;
DataSource dsourse_out;
RTPManager rtp_man;
```

These classes are a part of the jmf.jar library. The *media_locator* object provides localization of the media source. On Windows platform the video media source has its own address – "vfw://0". That address is used to create the *media_locator* object. That object is needed to catch media data source and to put that data to object with DataSource type. DataSource class object contains data from media source. In the case of this project it is a multimedia capture system. The code below shows creation process of video transmission.

```
rtp_man = RTPManager.newInstance();
SessionAddress sa_local = new Ses-
sionAddress(InetAddress.getLocalHost(), this.get_video_IPport+10);
InetAddress inet_addr_remote =
InetAddress.getByName("192.168.100.167");

rtp_man.initialize(sa_local);
SessionAddress sa_remote = new SessionAddress(inet_addr_remote,
this.get_video_IPport);
rtp_man.addTarget(sa_remote);

media_locator = new MediaLocator("vfw://0");
lds_source = Manager.createDataSource(media_locator);
media_processor = Manager.createProcessor(lds_source);

waitForState(media_processor, Processor.Configured);

TrackControl [] tracks = media_processor.getTrackControls();
System.out.println(tracks.length);
VideoFormat vf_media = new VideoFormat(VideoFormat.JPEG_RTP,
                                       new Dimen-
sion(this.get_video_res_H, this.get_video_res_W),
                                       Format.NOT_SPECIFIED,
                                       Format.byteArray,
                                       Format.NOT_SPECIFIED
        );

tracks[0].setFormat(vf_media);
ContentDescriptor cd = new Con-
tentDescriptor(ContentDescriptor.RAW_RTP);
media_processor.setContentDescriptor(cd);

waitForState(media_processor, Controller.Realized);
```

```
dsourse_out = media_processor.getDataOutput();

SendStream sendStream = rtp_man.createSendStream(dsourse_out, 0);
sendStream.start();
media_processor.start();
```

After DataSource has been created, a Processor class object is created to process data from media source. Data from the media source needs to be coded or decoded. Processor class object *media_processor* is created to process that data. The *media_processor* object has more than one track to transmit media data from source. Each track needs to be configured. The *vf_media* object exists for that purpose. Then, that configuration object is applied to each media track. The *ContentDescriptor* class object provides information about transmission format. In this project it is "RAW_RTP" format of packets. After that stack of processing output data is saved into DataSource class object which is called *dsourse_out*. The RTPManager class object `rtp_man`, manages targets of transmission. Then, processed data is put in stream and transmitted to targets.

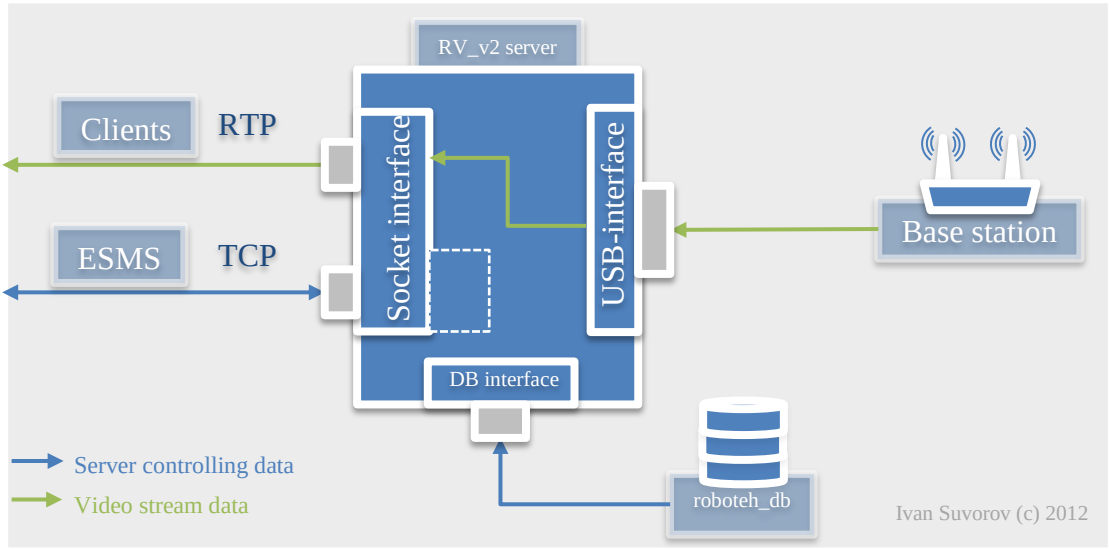


Figure 34. RV_v2 server structure

The *db_connect* class provides communication with a database. On Figure 34 it's shown as DB interface. In that project, it establishes communication with the roboteh_db database to retrieve configuration data from there.

The *tcp_connect* class establishes communication with ESMS. On Figure 34 it's shown as Socket interface. That interface helps to take current status of server.

### 7.4 The roboteh_db database

The roboteh_db database contains all server's configurations and data from the sensors. The description of the tables with the description of the columns is shown in Appendix 1.

After data packet from the robot-platform is caught, SeCoDa server calls its method to parse and insert parsed data. As was described early, GPS and UT packets have their own methods to parse and insert data as *db_insert_gps_imp*, *db_insert_ultra_imp* and *db_insert_thermo_imp* methods respectively. The *db_insert_gps_imp* method inserts data into the *rb_sensor_gps* table in the hardware schema. In Appendix 1 Table 17 describes that table and its columns. Data from ultra and thermo sensors is inserted into *rb_sensor_ultra* and *rb_sensor_thermo* respectively. In Appendix 1 Table 20 and Table 18 describe *rb_sensor_ultra* and *rb_sensor_thermo* respectively.

#### Insert test

Originally, the roboteh_db database was deployed on Windows platform. As Figure 35 shows the average time of inserting is 30ms. However arrangement is not an uniform. The variation is between 20 and 35 I/S.

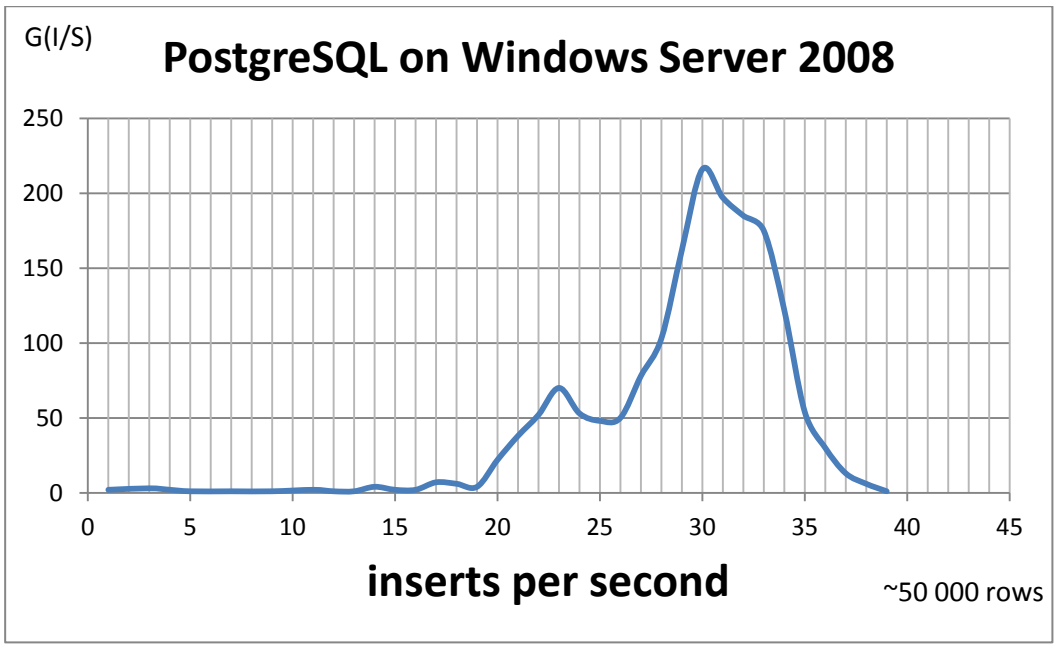


Figure 35. Inserts statistic under Windows platform

But if database is moved to different platform, for example, to Linux family platform, the picture is changed extremely in a better way. Figure 36 shows the difference between two platforms Windows Server 2008 with NTFS file-system and CentOS operation system with EXT3 file-system. CentOS is Linux based operation system. It is used to servers as web-servers or data-servers.

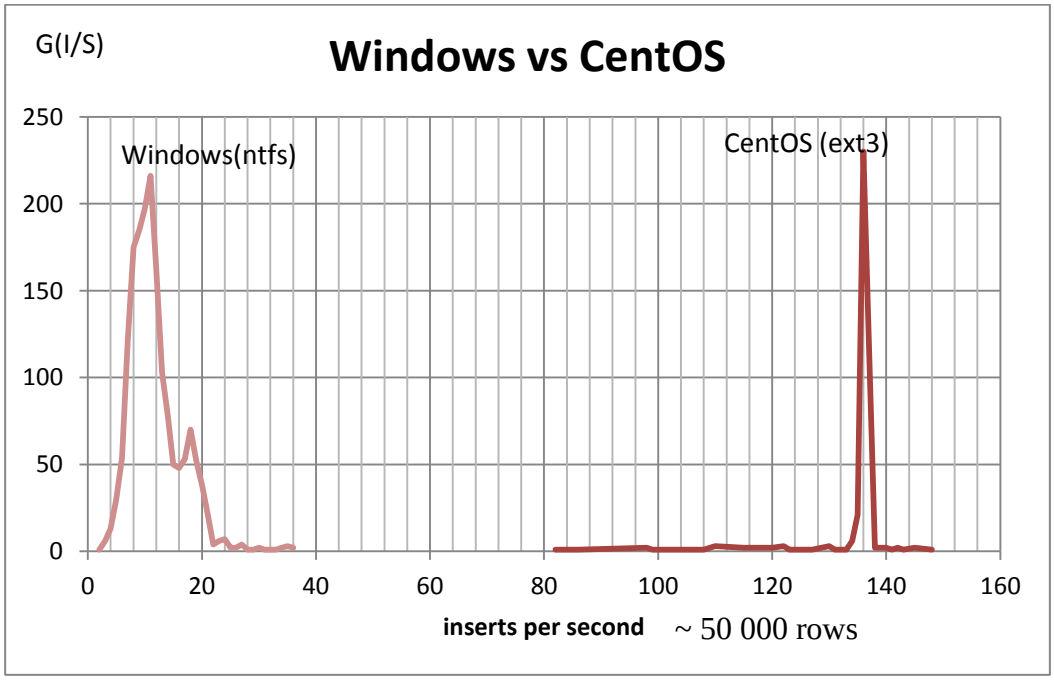


Figure 36. Windows vs. CentOS – comparing I/S

Figure 36 shows average Inserts/Second (I/S) difference between Windows and CentOS. CentOS with EXT3 file-system is four times faster than Windows with NTFS file-system. The average time to insert one row into the roboteh_db database for Windows is 30ms and for CentOS 7ms. This great difference between these two OSs makes an obvious choice to move to Linux platform (CentOS from the Linux family).

The script of the test is shown below. That script counts an amount of inserts per seconds and then groups it.

```
SELECT
    STAT.statistic,
    count(*)
FROM
    (
    SELECT
        TRUNCED_BY_SECOND.time_s,
        count(*) as statistic
    FROM
        (SELECT
            date_trunc('second',time_create) as time_s
        FROM
            HARDWARE.RB_SENSOR_ULTRA) as TRUNCED_BY_SECOND
    GROUP BY
        TRUNCED_BY_SECOND.time_s
    ) as STAT
GROUP BY
    STAT.statistic
ORDER BY
    STAT.statistic  DESC;
```

## 8 CONCLUSION

The aim of the “Servers and Extendable Server’s Management System for the Robot-platform” part of the project was selection and implementation technologies which would help to provide a reliable communication between the robot-platform and a client side through a network. Through this communication a client side is allowed to control the robot-platform, get data from it and receive real-time video streaming.

During the design and implementation period the following main technologies and platforms were chosen to this project:

- Java Virtual Machine – to provide cross-platform capabilities
- CentOS 5.7 – to improve a database, web-server and SeCoDa server performances
- Tomcat5 – as a web-server
- PostgreSQL 9.1 – to store data reliably
- RXTX libraries – to provide communication through a serial port from Java Environment
- JMF – to provide video streaming transmission
- JSONP – to provide cross-domain data interchanging
- UDP – to provide agile controlling communication
- TCP – to provide inter-servers controlling communications
- JSP – to provide Roboteh Application Programming Interface (RAPI)
- JavaBeans – to realise Roboteh Application Programming Interface (RAPI)
- JavaScript – to provide Web User Interface Background Services (WUI BS)
- AJAX - to provide Web User Interface Background Services (WUI BS)
- HTML - to provide Web User Interface (WUI)

The aim of the whole project was provide extremely flexible server management solution for the Roboteh project. And it was achieved by dividing this part of the "Roboteh" project for more than one server and one system. SeCoDa server was designed to manage control data from the client side to the robot-platform and data from the sensors. RV_v2 video server was design to provide video streaming. ESMS was design to provide RAPI. This part of the "Roboteh" project has five main outer interfaces to interact with. They are the RTP audio/video streaming interface (under RTP protocol), the controlling UDP interface (under UDP protocol) and the RAPI interface to set and get data from the system are the three interfaces for a client side; the COM-port interface (under Serial-port) and the USB audio/video interface (under USB interface) are the two interfaces for a hardware side.

The controlling through a network allows to establish communication with a large distance between the client side and the robot-platform. The main difference between local and remote management will be just in an end-to-end delay. It means that everybody can design a simple client side web or stand-alone application under the current specification of the interfaces and control the robot-platform or receive a video

streaming from it. This is possible from any place around the world which has a network communicated to a network where is the robot-platform.

The design of the roboteh_db database allows to store data from the sensors and configurations for the servers. Then this data from the sensors can be used by third party client side applications and configuration data is used by the servers. Extendable Server's Management System provides cross-domain RAPI by JSONP-formatted data. It allows to use RAPI by third party applications and web-sites.

The further steps of the server side part of the Roboteh project are to provide a security level for connections and data; redesign of the database schemes to provide more than one a robot-platform; implementation procedures and triggers in the database to manage data from the sensors and the configuration data; design subsystem to autonomous controlling directly from the server; change the video codec to decrease data payload and increase the resolution and the quality of the video picture; decrease the streaming process delay; expand RAPI and ESMS WUI functionality.

**BIBLIOGRAPHY**

**Books**

**Begg, Carolyn – Cannolly, Thomas** 2010. Database Systems. A practical Approach to Design, Implementation, and Management. Fifth Edition. Addisom-Wesley.

**Kurose, James F. – Ross, Keith W.** 2010. Computer Networking. A Top-Down Approach. Fifth Edition. Addisom-Wesley.

**Stallings, William** 1990. Local Networks. Third Edition. New York. Macmillan Publishing Company.

**Darwen, Hugh** 2010. An Introduction to Relational Database Theory.

**Electronic sources**

**Server (computing)** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://en.wikipedia.org/wiki/Server_%28computing%29>

**Server** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://whatis.techtarget.com/definition/0,,sid9_gci212964,00.html>

**Database Fundamentals** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://www.personal.psu.edu/glh10/ist110/topic/topic07/topic07_06.html>

**PostgreSQL About** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://www.postgresql.org/about/>

**JavaBeans** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://docs.oracle.com/javase/tutorial/javabeans/>

**Introducing JSON** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://www.json.org/>

**Defining Safer JSON-P** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://json-p.org/ >

**Video compression** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://en.wikipedia.org/wiki/Video_compression#Video>

**Apostolopoulos, John G. - Tan, Wai-tian - Wee, Susie J. 2002.** Video Streaming: Concepts, Algorithms, and Systems. [on line]. HP Laboratories Palo Alto. [referred 19.03.2012]. Available in pdf-format: http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf.

**NMEA data** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://www.gpsinformation.org/dale/nmea.htm >

**Media Streaming** 2012. [on line] [referred 19.03.2012]. Available in www-format: <URL: http://www.l1associates.com/technology5.htm >

**APPENDIX 1**

## Table 10. rb_com_controler

| Contains main configurations for com-port. | | |
|---|---|---|
| **id_contr** | serial NOT NULL | |
| **robcom_base** | int2 | Put base commands from client side [0-9:driving control][0-9:camera control][0-9:speed][0-9:extra flags] |
| **robcom_counter** | int2 | Counter of repeated commands from client side |
| **robcom_clock_rate** | int2 | Clock rate of commands (ms) |
| **robcom_time_write** | timestamp | |
| **robcom_session_begin** | timestamp | Beginning of demo-session Timestamp |
| **robcom_session_end** | timestamp | Ending of demo-session Timestamp |
| **robcom_session_id** | int4 | ID session |
| **robcom_session_name** | varchar( 20 ) | Name of session |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_robo_com_controler primary key** | ( id_contr) | |

## Table 11. rb_com_data_types

| Data types for unparsed data | | |
|---|---|---|
| **id_type** | serial NOT NULL | |
| **r_name** | varchar( 100 ) | |
| **descr** | varchar( 100 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_robo_com_data_types primary key** | ( id_type) | |

## Table 12. rb_com_in_stream

| Unparsed data from robot-platform. | | |
|---|---|---|
| **id_i_str** | serial NOT NULL | |
| **id_type** | int4 NOT NULL | |
| **data** | varchar( 100 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use |

| | | |
|---|---|---|
| | | or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_robo_com_in_stream primary key** | ( id_i_str) | |
| **idx_robo_com_in_stream** | ( id_type) | |
| Foreign Keys | | |
| **fk_robo_com_in_stream** | ( id_type ) ref rb_com_data_types ( id_type ) | |
| | | |

### Table 13. rb_com_out_stream

| **Unparsed data to robot-platform.** | | |
|---|---|---|
| **id_o_str** | serial NOT NULL | |
| **id_type** | int4 NOT NULL | |
| **data** | varchar( 100 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_robo_com_in_stream_0 primary key** | ( id_o_str) | |
| **idx_robo_com_out_stream** | ( id_type) | |
| Foreign Keys | | |
| **fk_robo_com_out_stream** | ( id_type ) ref rb_com_data_types ( id_type ) | |
| | | |

**Table 14. rb_conf_types**

| **List types of configurations** | | |
|---|---|---|
| **id_conf_t** | serial NOT NULL | The id of a type configuration |
| **r_name** | varchar( 100 ) | Sensor's name |
| **type** | varchar( 100 ) | The type of a sensor |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_sensors_types_0 primary key** | ( id_conf_t) | |

**Table 15. rb_files_conf_list**

| **Constains configurations in json-format files** | | |
|---|---|---|
| **id_conf_list** | serial NOT NULL | |
| **json_conf_commands** | text | json-file Commands descriptions |
| **json_conf_net_settings** | text | json-file Network settings for individual user |
| **json_conf_client_settings** | text | json-file Client's interface |
| **json_conf_video_settings** | int4 | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_rb_configures_0 primary key** | ( id_conf_list) | |

**Table 16. rb_sensor_acc**

| **Contains data from accelerometer sensor.** | | |
|---|---|---|
| **id** | serial NOT NULL | |
| **id_sensor** | int4 NOT NULL | |
| **sensor_acc_x** | int2 | Accelerometer Sensor: X direction |
| **sensor_acc_y** | int2 | Accelerometer Sensor: Y direction |
| **sensor_acc_z** | int2 | Accelerometer Sensor: Z direction |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |

| | | |
|---|---|---|
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_sensor_acc primary key** | ( id) | |
| **idx_sensor_acc** | ( id_sensor) | |
| Foreign Keys | | |
| **fk_sensor_acc** | ( id_sensor ) ref rb_sensor_types ( id_sensor ) | |
| | | |

**Table 17. rb_sensor_gps**

| **Contains data from GPS sensor.** | | |
|---|---|---|
| **id** | serial NOT NULL | |
| **id_sensor** | int4 NOT NULL | |
| **gps_lat_sms** | float4 | The lattitude in seconds: seconds and milliseconds |
| **gps_lng_sms** | float4 | The longitude in seconds: Seconds and milliseconds |
| **gps_hemisphere_lat** | varchar( 1 ) | North/South (N or W) |
| **gps_hemisphere_long** | varchar( 1 ) | East/West (E or W) |
| **gps_direction** | float4 | Angel from GPS |
| **gps_speed** | float4 | The speed value from GPS |
| **gps_valid** | varchar( 1 ) | validity - A-ok, V-invalid |
| **gps_time** | timestamp | Time Stamp |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_sensor_acc_0 primary key** | ( id) | |
| **idx_sensor_gps** | ( id_sensor) | |
| **pk_rb_sensor_gps unique** | ( r_user) | |
| Foreign Keys | | |
| **fk_sensor_gps** | ( id_sensor ) ref | |

| | | |
|---|---|---|
| | rb_sensor_types ( id_sensor ) | |
| | | |

### Table 18. rb_sensor_thermo

| **Contains data from termo-sensor.** | | |
|---|---|---|
| **id** | serial NOT NULL | |
| **id_sensor** | serial NOT NULL | |
| **temperature** | int2 | |
| **r_sign** | varchar( 1 ) | - or + |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_sensor_acc_2 primary key** | ( id) | |
| **idx_rb_sensor_termo** | ( id_sensor) | |
| Foreign Keys | | |
| **fk_rb_sensor_termo** | ( id_sensor ) ref rb_sensor_types ( id_sensor ) | |

### Table 19. rb_sensor_types

| **Contains different types of sensors.** | | |
|---|---|---|
| **id_sensor** | serial NOT NULL | The id of a sensor type |
| **r_name** | varchar( 100 ) | Sensor's name |
| **type** | varchar( 100 ) | Sensor type |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |

| | |
|---|---|
| **time_modify** | timestamp |
| **r_user** | varchar( 50 ) |
| Indexes | |
| **pk_sensors_types primary key** | ( id_sensor) |
| **idx_rb_sensor_types** | ( r_user) |
| Foreign Keys | |
| **fk_rb_sensor_types** | ( r_user ) ref rb_sensor_gps ( r_user ) |

**Table 20. rb_sensor_ultra**

| **Contains data from ultrasonic sensors.** | | |
|---|---|---|
| **id** | serial NOT NULL | |
| **id_sensor** | int4 NOT NULL | |
| **sensor_ultra_f** | int2 | Data from ultraSonic sensor:Front |
| **sensor_ultra_l** | int2 | Data from ultraSonic sensor:Left side |
| **sensor_ultra_r** | int2 | Data from ultraSonic sensor:Right side |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_sensor_acc_1 primary key** | ( id) | |
| **idx_sensor_ultra** | ( id_sensor) | |
| Foreign Keys | | |
| **fk_sensor_ultra** | ( id_sensor ) ref rb_sensor_types ( id_sensor ) | |

**Table 21. rb_user_files_conf**

| **Contains list of file's configurations for each user.** | | |
|---|---|---|
| **id_conf** | serial NOT NULL | |
| **id_user** | int4 NOT NULL | An user id |
| **id_conf_list** | int4 NOT NULL | An id of configuration file. |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_confs primary key** | ( id_conf) | |
| **idx_rb_user_confs** | ( id_user) | |
| **idx_rb_user_confs_0** | ( id_conf_list) | |
| Foreign Keys | | |
| **fk_rb_user_confs** | ( id_user ) ref rb_users_list ( id_user ) | |
| **fk_rb_user_confs_0** | ( id_conf_list ) ref rb_files_conf_list ( id_conf_list ) | |
| | | |

## Table 22. rb_user_plane_conf

| **The table links id user with id configurations** | | |
|---|---|---|
| **id_conf** | serial NOT NULL | The id of user's configuration |
| **id_user** | serial NOT NULL | The id of the user |
| **id_conf_t** | serial NOT NULL | The id of a type of the configuration |
| **id_conf_n** | int4 | ID from one of the table (only with conf_type) |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |

| | |
|---|---|
| **time_modify** | timestamp |
| **r_user** | varchar( 50 ) |
| Indexes | |
| **pk_rb_user_confs_0 primary key** | ( id_conf) |
| **idx_rb_user_plane_conf** | ( id_user) |
| **idx_rb_user_plane_conf_0** | ( id_conf_t) |
| Foreign Keys | |
| **fk_rb_user_plane_conf** | ( id_user ) ref rb_users_list ( id_user ) |
| **fk_rb_user_plane_conf_0** | ( id_conf_t ) ref rb_conf_types ( id_conf_t ) |

### Table 23. rb_user_plane_conf_client

| **The configuration list for a client** | | |
|---|---|---|
| **id_conf_n** | serial NOT NULL | The id of the configuration |
| **id_conf_t** | serial NOT NULL | |
| **description** | varchar( 200 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_plane_conf_network_0 primary key** | ( id_conf_n) | |
| **idx_rb_user_plane_conf_client** | ( id_conf_t) | |
| Foreign Keys | | |
| **fk_rb_user_plane_conf_client** | ( id_conf_t ) ref rb_conf_types ( id_conf_t ) | |

### Table 24. rb_user_plane_conf_iplist

| **IP list for a video stream** |
|---|

| | | |
|---|---|---|
| **id_conf_n** | serial NOT NULL | The id of a configuration |
| **id_conf_t** | serial NOT NULL | The id of a type of the configuration |
| **addr_ip4** | varchar( 15 ) | The IP address of a video-stream client |
| **addr_port** | int2 | The port of video-stream client |
| **type** | varchar( 100 ) | Type of casting (broad, multi, uni) |
| **describtion** | varchar( 200 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_plane_conf_network primary key** | ( id_conf_n) | |
| **idx_rb_user_plane_conf_network** | ( id_conf_t) | |
| Foreign Keys | | |
| **fk_rb_user_plane_conf_network** | ( id_conf_t ) ref rb_conf_types ( id_conf_t ) | |

**Table 25. rb_user_plane_conf_secodas**

| **The configuration list for the SeCoDa server** | | |
|---|---|---|
| **id_conf_n** | serial NOT NULL | The id of a configuration |
| **id_conf_t** | serial NOT NULL | The id of a type of the configuration |
| **addr_ip4** | varchar( 15 ) | The IP address of the SeCoDa server |
| **addr_port** | int2 | The port of SeCoDa server |
| **describtion** | varchar | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row |

| | | |
|---|---|---|
| | | Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_plane_conf_network_2 primary key** | ( id_conf_n) | |
| **idx_rb_user_plane_conf_secoda** | ( id_conf_t) | |
| Foreign Keys | | |
| **fk_rb_user_plane_conf_secoda** | ( id_conf_t ) ref rb_conf_types ( id_conf_t ) | |
| | | |

## Table 26. rb_user_plane_conf_videos

| **The configuration list for the RV_v2 server** | | |
|---|---|---|
| **id_conf_n** | serial NOT NULL | The id of a configuration |
| **id_conf_t** | serial NOT NULL | The id of a type of the configuration |
| **addr_ip4** | varchar( 15 ) | The IP address of the video server |
| **addr_port** | int2 | The port of video server |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_plane_conf_network_1 primary key** | ( id_conf_n) | |
| **idx_rb_user_plane_conf_video** | ( id_conf_t) | |
| Foreign Keys | | |
| **fk_rb_user_plane_conf_video** | ( id_conf_t ) ref rb_conf_types ( id_conf_t ) | |

**Table 27. rb_user_types**

| **Contains links between users and types of user.** | | |
|---|---|---|
| **id_utypes** | serial NOT NULL | |
| **id_user** | int4 NOT NULL | |
| **id_type** | int4 NOT NULL | |
| **descr** | varchar( 50 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_user_types primary key** | ( id_utypes) | |
| **idx_rb_user_types** | ( id_user) | |
| **idx_rb_user_types_0** | ( id_type) | |
| Foreign Keys | | |
| **fk_rb_user_types** | ( id_user ) ref rb_users_list ( id_user ) | |
| **fk_rb_user_types_0** | ( id_type ) ref rb_users_types_list ( id_type ) | |

**Table 28. rb_users_list**

| **Contains list of users in robo-system.** | | |
|---|---|---|
| **id_user** | serial NOT NULL | |
| **r_name** | varchar( 100 ) | The name of user. |
| **surname** | varchar( 100 ) | The second name (surname) of user. |
| **name_addition** | varchar( 100 ) | The third and next names of user. |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | who modified data |
| **Indexes** | | |
| **pk_rb_users_list primary key** | ( id_user) | |

**Table 29. rb_users_nicks**

| **Contains information about a nick name of the user.** | | |
|---|---|---|
| **id_nick** | serial NOT NULL | |
| **id_user** | serial NOT NULL | The id of the user |
| **nick** | varchar( 100 ) | Nick name |
| **descr** | varchar( 50 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_users_nicks primary key** | ( id_nick) | |
| **idx_rb_users_nicks** | ( id_user) | |
| Foreign Keys | | |
| **fk_rb_users_nicks** | ( id_user ) ref rb_users_list ( id_user ) | |

**Table 30. rb_users_types_list**

| **Contains user's types: an administrator, a client and etc.** | | |
|---|---|---|
| **id_type** | serial NOT NULL | |
| **type** | varchar( 100 ) | |
| **descr** | varchar( 50 ) | |
| **in_use** | int2 DEFO 1 | Shows a state of a row: Available to use or not. Default: 1 (Available). 0 - unavailable; |
| **error_row** | int2 DEFO 0 | Shows a badness of a row; 1 - an erroneous row, 0 - an error-free row Default:0; |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| Indexes | | |
| **pk_rb_users_types_list pri-** | ( id_type) | |

| mary key | | |
|---|---|---|

## Table 31. rb_user_route

| Custom routes for user from Google map | | |
|---|---|---|
| **id_route** | serial NOT NULL | |
| **id_user** | serial NOT NULL | |
| **r_name** | **varchar( 100 )** | The name of a route |
| **distance** | float8 | Distance of a route |
| **points** | int4 | An amount of points in the route |
| **descr** | varchar( 100 ) | |
| **in_use** | int2 DEFO 1 | |
| **error_row** | int2 DEFO 0 | |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_rb_user_route primary key** | ( id_route) | |
| **idx_rb_user_route** | ( id_user) | |
| **Foreign Keys** | | |
| **fk_rb_user_route** | ( id_user ) ref rb_users_list ( id_user ) | |

## Table 32. rb_user_route_ll

| Latitudes and Longitudes of the route | | |
|---|---|---|
| **id** | serial NOT NULL | |
| **id_route** | serial NOT NULL | |
| **route_lat** | **numeric(18,15)** | Latitude of the route |
| **route_long** | **numeric(18,15)** | Longitude of the route |
| **in_use** | int2 DEFO 1 | |
| **error_row** | int2 DEFO 0 | |
| **time_create** | timestamp | |
| **time_modify** | timestamp | |
| **r_user** | varchar( 50 ) | |
| **Indexes** | | |
| **pk_rb_user_route_ll primary key** | ( id) | |
| **idx_rb_user_route_ll** | ( id_route) | |
| **Foreign Keys** | | |

| | |
|---|---|
| **fk_rb_user_route_ll** | ( id_route ) ref rb_user_route ( id_route ) |