Tomas Bata University in Zlín
Faculty of Applied Informatics

Michal Gerža, MSc.

Intelligent Measureserver for Controlling Remote Real Experiments with Embedded Simulations and Advanced Diagnostics

Doctoral Thesis

Branch of study:	Engineering Informatics
Supervisor:	Prof. Dr. František Schauer, DSc.

Zlín,
August 2017

ACKNOWLEDGEMENTS

I would like to thank all the people who helped me along the way to finish this work. My family, colleagues in the team and friends who supported me and one exceptional man, my supervisor Prof. Dr. František Schauer, DSc. who introduced me to the world of science and amazed me with his experiences and brilliant ideas. He was my guide during the doctoral study and helped me to achieve many things I would never dare to think of them.

I would also like to recognize and extend my appreciation to the committee members, for taking time to read the dissertation and offering constructive criticisms that helped improved the final work.

Especial thanks for the support of the present work during majority period of the doctoral study, belongs to the Swiss National Science Foundation via the project SCOPES and thanks to Dr. Denis Gillet for organizational work.

ABSTRACT

People are developing various technologies, using laws of physics behind real-world phenomena. This process is continuous and is presently getting faster and more intensive. Achieved technological advances affect all areas of every day’s life, including the field of education. Contemporary society requires more complex tools providing more accessible and effective educational methods. Information and communication technologies offer many tools to meet the requirements on which rest teaching and learning, especially in natural sciences and engineering fields. Studying these fields is based on real experimental work in laboratories with specific equipment and devices for a better understanding the given phenomenon. This form of study, when the user must regularly attend laboratories to perform necessary measurements, is not always acceptable for many reasons. Most of laboratories cannot provide a wide range of the real experiments, where expensive devices, needed for the elucidating and analysis of desired phenomena, are involved. Development in this direction of these requirements led to the design and implementation of the concept of the remote laboratories to deliver physical experiments via the Internet. The connected users are provided with the different tools as, for example, corresponding theory of the studied phenomenon. In our case, the concept is called Intelligent School Experimental System (ISES) operated as an open system platform.

The doctoral thesis focuses on the design and implementation of software components related to the ISES Measureserver, finite-state machine in principle that is a core unit of this platform. The activities within the thesis are aimed at five tasks, defined as the goals of the thesis. These goals were solved as the independent project works, integrating progressive concepts, approaches and technologies, which bring new features contributing to better teaching outcomes, reliability and maintenance of the ISES remote laboratories.

Keywords: REMLABNET; RLMS; ISES; Measureserver; remote laboratory; data archiving; advanced diagnostics; embedded simulation.

ABSTRAKT

Lidé vyvíjejí různé technologie využívající fyzikálních zákonů, které popisují přírodní jevy. Tento vývoj je kontinuální a je v současné době rychlejší a intenzivnější. Dosažený technologický pokrok ovlivňuje všechny oblasti každodenního života zahrnující i oblast vzdělávání. Současná společnost požaduje složitější nástroje poskytující dostupnější a efektivnější vzdělávací metody. Informační a komunikační technologie nabízejí mnoho nástrojů vyhovující požadavkům pro podporu výuky a studia, obzvláště v oblasti přírodních věd a techniky. Studium těchto oborů je založeno na reálné experimentální práci v laboratořích se specifickým vybavením a zařízeními pro lepší pochopení daných jevů. Tato forma studia, kdy uživatel musí pravidelně navštěvovat laboratoře, aby vykonal nezbytná měření, již není akceptována z mnoha důvodů. Většina laboratoří nemůže poskytnout široký rozsah reálných experimentů, kde jsou k dispozici drahá zařízení pro objasnění a analýzu požadovaných jevů. Vývoj směrem k těmto požadavkům vedl k návrhu a realizaci konceptu vzdálených laboratoří s cílem poskytnout fyzikální experimenty přes Internet. Připojeným uživatelům jsou poskytnuty rozdílné nástroje, jako je například odpovídající teorie ke studovanému jevu. V našem případě je koncept nazván Inteligentní školní experimentální systém (ISES) běžící jako otevřená systémová platforma.

Tato doktorská práce se zaměřuje na návrh a implementaci softwarových komponent související s jednotkou Measureserver ISES, v podstatě s konečným stavovým automatem, jenž je hlavní jednotkou této platformy. Aktivity v této práci jsou zaměřeny na pět úkolů, které jsou definovány jako cíle této disertace. Tyto cíle byly řešeny jako nezávislé projektové práce, integrující progresivní koncepty, přístupy a technologie, které přinášejí nové funkce k dosažení lepších výsledků ve výuce, spolehlivosti a údržbě vzdálených laboratoří ISES.

Klíčová slova: REMLABNET; RLMS; ISES; Measureserver; vzdálená laboratoř; zálohování dat; pokročilá diagnostika; vestavěná simulace.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AC – Alternating Current

ADDA – Analog to Digital/Digital to Analog

ANSI – American National Standards Institute

API – Application Programming Interface

CFG – Configuration

CLSID – Globally Unique Identifier

COM – Component Object Model

DAM – Data Archiving Management

DC – Direct Current

DLL – Dynamic-Link Libraries

DOM – Document Object Model

DS – Diagnostic Server

EjsS – Easy Java/JavaScript Simulations

EPS – Embedded real-world Phenomena Simulation

ER-L – Easy Remote Laboratory

FSM – Finite-State Machine

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

HW – Hardware

ICT – Information and Communication Technology

IMAP – Internet Message Access Protocol

IP – Internet Protocol

ISA – Instruction Set Architecture

ISES – Intelligent School Experimental System

LMS – Learning Management Systems

LOG – Data Log

MAC – Media Access Control

MFC – Microsoft Foundation Classes

MS – Measureserver

MSXML – Microsoft XML Core Services

MUD – Measureserver Unit Diagnostics

ODE – Ordinary Differential Equation

PCI – Peripheral Component Interconnect

PMD – Physical Modules Diagnostics

POP3 – Post Office Protocol 3

RL – Remote Laboratory

RLC – Resistor, Inductor and Capacitor

RLMS – Remote Laboratory Management System

SDK – Software Development Kit

SMTP – Simple Mail Transfer Protocol

SQL – Structured Query Language

STEM – Teaching Science, Technology, Engineering and Mathematics

STL – Standard Template Library

TCP – Transmission Control Protocol

TCP/IP – Transmission Control Protocol/Internet Protocol

UDP – User Datagram Protocol

UML – Unified Modeling Language

XML – eXtensible Markup Language

W3C – World Wide Web Consortium

1. INTRODUCTION

The Information and Communication Technology (ICT) today has enabled fast and rich ways of exchanging information among people from different domains with a variety of applications. One such application is the remote laboratory (RL), which are web-based interactive systems giving clients the ability to control and observe the characteristics and processes of remote equipment through the Internet anytime and anywhere. Teaching Science, Technology, Engineering and Mathematics (STEM) is seen at schools and universities as an effective way of growing interest in science related fields. Such education needs practical hands on experience in combination with theoretical knowledge of STEM concepts. Practical experience is acquired by controlling, observing and recording of facts coming from different equipment setups and operational environments.

1.1 Remote laboratories

The RLs can provide access to resources, which are otherwise inaccessible to users, denoted as clients. Typically, all that is required is a web browser and an Internet connection to enable rich educational possibilities. A number of collaborative projects have integrated physical experiments under a common infrastructure. Aims of these efforts typically include simple access by clients and the support of collaboration among connected clients. Research has mainly focused on the integration of the RLs into higher education, predominantly undergraduate courses, because of a large number of clients (scientists) and predefined sets of common experiments.

The RLs traditionally consist of two parts, the server and the client, as illustrated in Figure 1. The client’s side is used by a client engaging the RL and learning from the use of it. The server side provides the experiment rig (physical hardware), as well as the environment responsible for designing, constructing and maintaining the RL that also provides accompanying teaching and materials. Remote Laboratory Management System (RLMS) serves for the function of components and interfaces of the whole system forming the RL. Typical RLMS architecture comprises following functional units [1]:

1. Scheduling,
2. Rig operations,
3. Transport layer,

4. Experiment user interface,
5. Multimedia tools and experimental data,
6. Accepting and processing user requests,
7. Storing and maintaining user details.

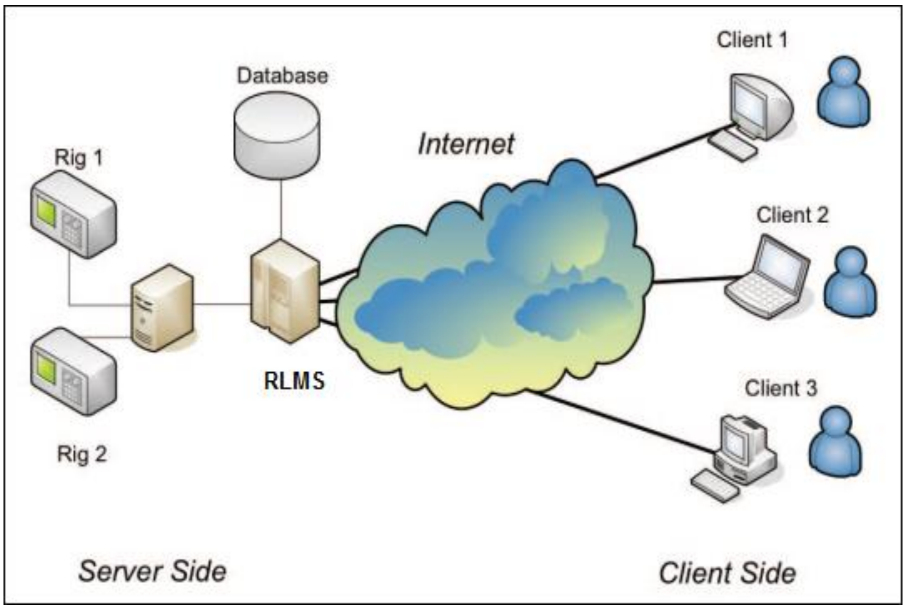


Figure 1 Typical scheme of the RLMS architecture

1.2 Remote laboratories globally

Plenty of RLMS systems covering a broad range of the RLs have been designed and deployed worldwide. Historically, the first two RLs have occurred in Slovakia, nearly simultaneously:

1. In the field of automatic control in the Department of Automatic Control, Faculty of Electrical Engineering of the Slovak University of Technology in Bratislava, that has been developed by the research team of Prof. Mikuláš Huba in 2007 [2][3],
2. In the field of “Electrochemical cell” in 2008, in the workplace of the Department of Physics, Faculty of Education in the University of Trnava in Trnava, has been developed by team of Prof. František Schauer and Assoc. Prof. Miroslava Ožvoldová [4][5].

Some of the largest and most widely used RLMSs are platforms constituted as the iLab [6][7], SAHARA [8], VISIR [9] and WebLab-Deusto [10].

The iLab has disposable three-layer architecture called the iLab Shared Architecture. Users connect with a service broker server, which in turn makes a connection with the actual laboratory server. The system architecture heavily rests on web services [11]. The iLab has also been used to implement extensions such as iLab-MIT-Africa [12] in African countries and some universities in Australia. Experiments in the iLab have been categorized into three different delivery methods: batched, interactive and sensor. The iLab is based on the Microsoft platforms including the Visual C#, .NET framework tools and Microsoft SQL Server. This makes the system very platform dependent, consequently it is difficult to implement on open source platforms. Recent attempts are being made to re-implement the ISA in the Java environment for the purpose of making it platform independent.

The SAHARA originally followed a client-server architecture, where all experiments were hosted at the UTS laboratories, and accessed upon request by remote users. In this design, the lists of experiments are stored by the central server, which is also responsible for other operational aspects including running the RL, scheduling, and operating the rig. Recent developments have moved towards grid architecture, but mostly within partner institutions.

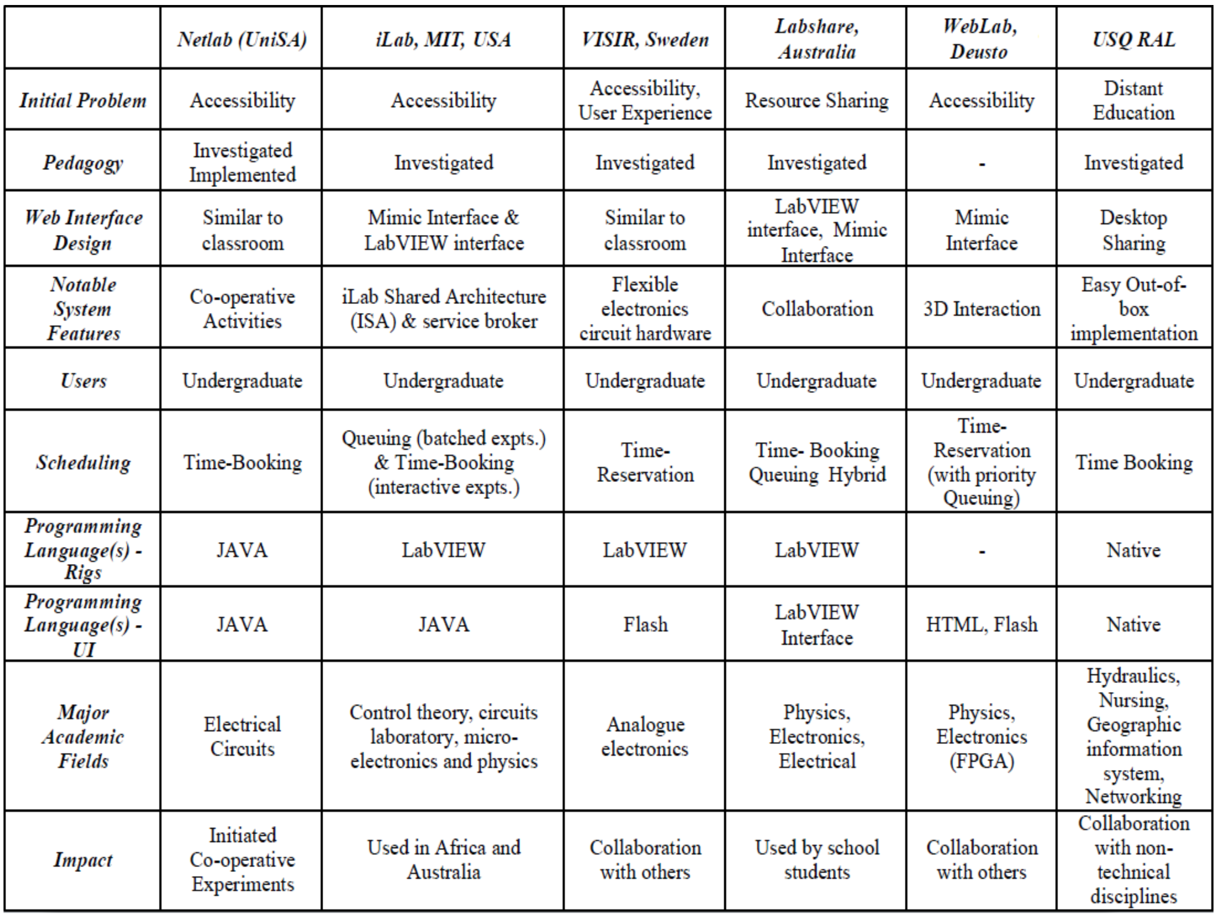
The VISIR implementations also follow client server architecture, where the complete experiment lists are stored in centralized databases along with user details, and connection to the same server is used for booking and operating the experiment. Both iLab and VISIR use the LabVIEW as the main platform and language to compile controlling programs to run RLs.

The next RL is WebLab-Deusto that has been developed at University of Duesto, Bilbao in Spain [13] and [14]. This system platform uses the client-server mechanism utilizing mostly time reservation with priority queuing based scheduling, although the nature of scheduling may change if connecting to other systems. Within this system, there is a wide variety of experiments ranging from basics of physics to Field Programmable Gate Array however the main focus is on electronics and electrical experiments.

In 2011, the project LiLa was started as a collaborative venture between several RL installations throughout Europe. Many virtual laboratories and RLs were effectively shared by partner institutions through a Learning Management Systems (LMS) and the LiLa Internet Portal. The learning aspects of the LiLa were managed by using SCORM, a learning object creation and management tool [15]. Actual operating and related costs of the laboratories however were still covered by the participating universities.

Finally listed RLs in brief, the RemoteElectlab [16] has been developed at University of Porto, Portugal targeting mechanical, physics electronics and meteorological experiments. The eMersion and SMARTLAB have been used at Technical University in Lausanne, Switzerland that uses the Graasp social media platform [17]. This system is also intended for higher education users in the fields of control theory, physics and others. The most widely used RLs with their basic attributes and functions are listed in Table 1.

Table 1 Comparison between the most widely used remote laboratories worldwide to see their main similarities and differences



	Netlab (UniSA)	*iLab, MIT, USA*	*VISIR, Sweden*	*Labshare, Australia*	*WebLab, Deusto*	*USQ RAL*
Initial Problem	Accessibility	Accessibility	Accessibility, User Experience	Resource Sharing	Accessibility	Distant Education
Pedagogy	Investigated Implemented	Investigated	Investigated	Investigated	-	Investigated
Web Interface Design	Similar to classroom	Mimic Interface & LabVIEW interface	Similar to classroom	LabVIEW interface, Mimic Interface	Mimic Interface	Desktop Sharing
Notable System Features	Co-operative Activities	iLab Shared Architecture (ISA) & service broker	Flexible electronics circuit hardware	Collaboration	3D Interaction	Easy Out-of-box implementation
Users	Undergraduate	Undergraduate	Undergraduate	Undergraduate	Undergraduate	Undergraduate
Scheduling	Time-Booking	Queuing (batched expts.) & Time-Booking (interactive expts.)	Time-Reservation	Time- Booking Queuing Hybrid	Time-Reservation (with priority Queuing)	Time Booking
Programming Language(s) - Rigs	JAVA	LabVIEW	LabVIEW	LabVIEW	-	Native
Programming Language(s) - UI	JAVA	JAVA	Flash	LabVIEW Interface	HTML, Flash	Native
Major Academic Fields	Electrical Circuits	Control theory, circuits laboratory, micro-electronics and physics	Analogue electronics	Physics, Electronics, Electrical	Physics, Electronics (FPGA)	Hydraulics, Nursing, Geographic information system, Networking
Impact	Initiated Co-operative Experiments	Used in Africa and Australia	Collaboration with others	Used by school students	Collaboration with others	Collaboration with non-technical disciplines

2. ISES HANDS-ON LABORATORIES, STATE OF THE ART

The Intelligent School Experimental System (ISES) is established as the advanced experimental tool for school laboratories and building of hands-on laboratory with the real-time operation, data acquisition, data processing, experimental hardware controlling, visualizing and analyzing. This system has been designed and developed by Assoc. Prof. Dr. František Lustig in 1992, and its development still continues [18]. There are registered 500 installations of this experimental tool at Czech and Slovak schools.

The ISES hands-on laboratory is an open system platform consisting of the ISES physical hardware (HW) and ISES WIN software components. It provides the processing of measured data, for example the integration, differentiation, approximation and fitting. The data transfer to plotting environments (preset various charts and data tables) is straightforward.

The physical HW is composed of the computer interface card PCI ADDA (Analog to Digital/Digital to Analog) 12-bit convertor, time of the conversion is 0.01 ms, the ISES control board and a set of the sensors (for physics, chemistry and biology). The system offers the possibility of simultaneous measuring and data displaying for 8 input channels and process control via 2 analogue and 2 binary output channels. The analogue outputs channels work as programmable voltage sources (DC, AC with eight kinds of default signals, manual controlling or user defined signals). Maximum sampling frequency is 100 kHz that enables studying of sounds or other high periodicity signals.

The ISES is supplied with a set of meters, sensors and devices (volt-meter, ampere-meter, capacity-meter, thermometer, ohmmeter, anemometer, force meter, heart frequency meter, simple position sensor, microphone, loudspeaker, relay, light gate, current booster, electromagnet valve for liquids, etc.). These specific sensors (modules) are easily interchangeable, the computer, senses their presence and adjusted range.

There is possible to study voltage, current, capacitance, resistivity, mass and force, pressure in liquids, etc. In chemistry, it is allowed to analyze the acidity, exothermic and endothermic reactions, titration and many other processes. Biology users can measure, for example, some periphery blood vessel system functions. The complete set of the physical modules, including the ISES WIN software displayed on the screen, is illustrated in Figure 2.

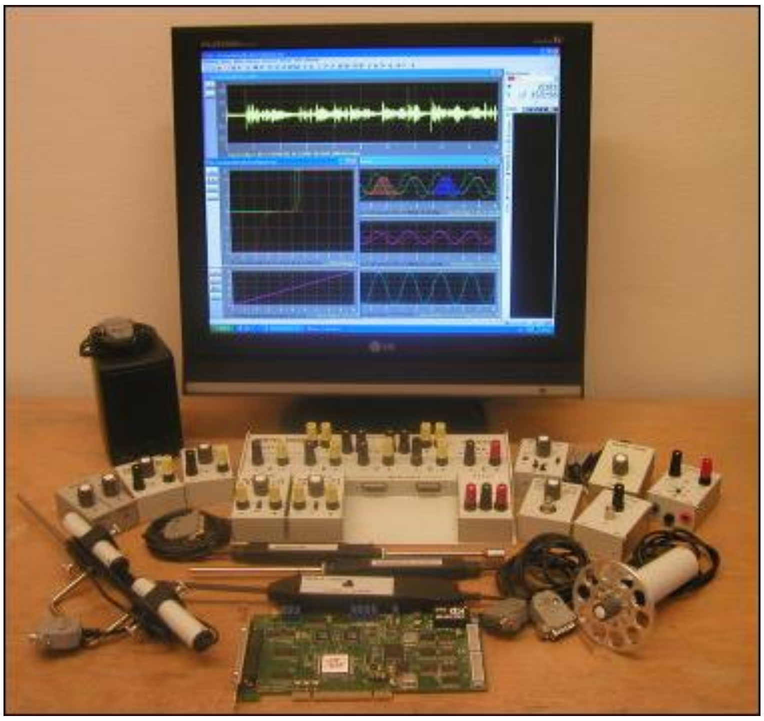


Figure 2 ISES physical hardware and software components; the PCI ADDA interface card, the set of meters, sensors and device

3. ISES REMOTE LABORATORIES, STATE OF THE ART

The ISES remote laboratories have come into existence as the next step of the ISES hands-on laboratories. The ISES physical hardware is unique as it possesses, as the only school experimental system in world, both inputs and outputs, which is a prerequisite for building RLs controlled by clients. Each RL offers a software solution for controlling the physical HW, and the client does not need any special program to be installed, and uses the standard web services (communication protocols and ports, web browsers, etc.) via the Internet. The RLs have been designed and implemented, using the ISES WEB Control Kit in 2002 by the development team at Charles University in Prague led by Assoc. Prof. Dr. František Lustig. Since 2014 RLs have been significantly improved both in Prague and later at Tomas Bata University in Zlín [19][20][21]. The improvement, related to the ISES Measureserver (MS) that is the basic unit of every RL, working as a finite-state machine (FSM), is subject of this doctoral thesis. Because of the implemented substantial improvements of the ISES MS, we will call this new type as ISES Intelligent Measureserver unit.

3.1 ISES remote laboratory history and basic working scheme

The starting version of the ISES RLs has been developed at Charles University in Prague led by Assoc. Prof. Dr. František Lustig in 2002. The software components have been programmed by MSc. Jiří Dvořák. It has been since constantly improved both at Charles University in Prague and Tomas Bata University in Zlín implementing features for the new user environment, Easy Remote Laboratory (ER-L), to simplify a process of the designing, building and maintaining RLs by a laymen. The ER-L also provides the RLs data archiving, MS diagnostics and embedding the simulations.

The ISES RL concept consists of five integral units as the HW 1) physical components (ADDA signal converter, ISES control board, physical modules categorized as meters, sensors, probes and specific devices), and units as the software 2) Measureserver, 3) Imageserver, 4) Webserver and 5) Webclient. More technical details and applications are available in [22][23][24].

All the built ISES RLs were recently integrated into Remote Laboratory Management System (RLMS) called REMLABNET.

3.1.1 ISES remote laboratory units

The ISES RL concept is based on the autonomous hardware and software exploited for the monitoring, controlling and processing to deliver real experiments residing in laboratories to connected clients for their educational purposes. The following subsection deals with a description of these units.

Physical hardware and software

The physical HW serves the standard ISES system. It is a modular platform based on three basic components. As the first component, it is a set of the physical modules like meters, sensors and devices (mentioned in Chapter 2), which are used for physics, biology, chemistry and electro-engineering. The physical modules are wired to the ISES control board involved as the second component. It transfers signals to the PCI ADDA convertor (interface card), as to the last component. The interface card is installed inside a computer to gather and process measured data or to set controlling devices attached to the rig. The software part constitutes the interface card drivers for the commutation and measurement. As the example, the physical components, which represent the ISES RL "Sound laboratory", are shown in Figure 3. There are installed and running the sound generator, voltage source, relay board, control board with computer, and apparatus with speaker and two microphones.

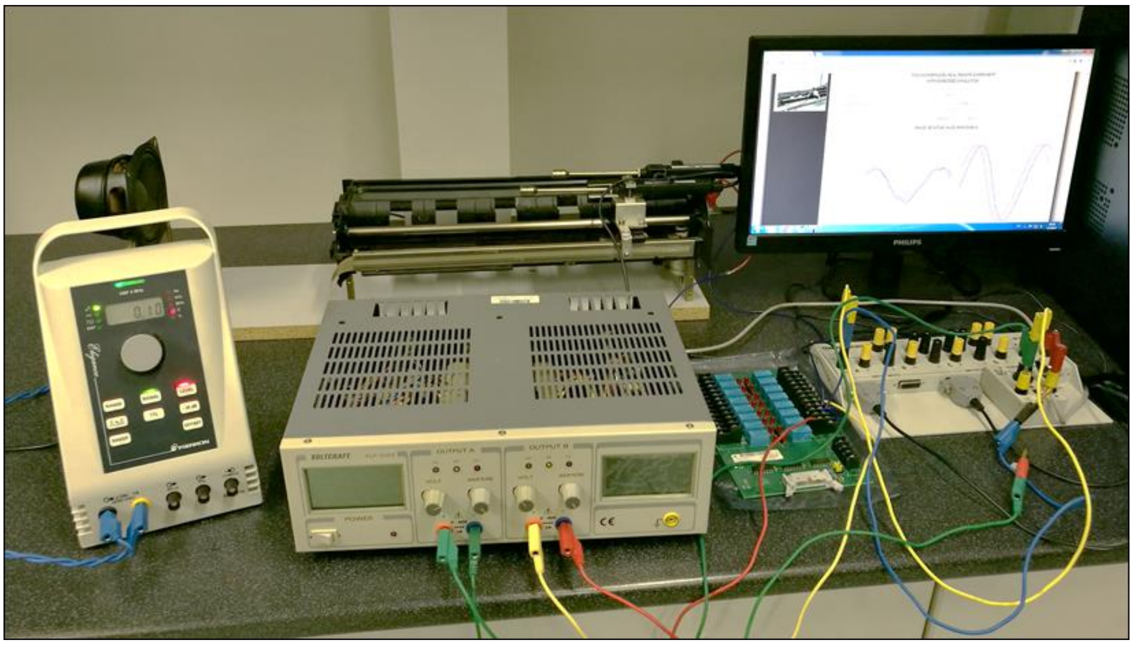


Figure 3 Example of the ISES RL "Sound laboratory" including the sound generator, voltage source, relay board, control board, computer, and apparatus with speaker and two microphones

Informatics hardware and software

The informatics HW serves a personal computer/notebook and network infrastructure to access the RL to connected clients. The software component part constitutes the specific units to ensure the ISES RL online.

Measureserver unit

The MS is a significant software component of the ISES RL concept. It is the processing and communication server located between the physical HW (rig) and remote clients. The MS core is designed as a deterministic finite-state machine to setup and perform all the logical and maintaining instructions for solving the prescribed activities. Its functioning is controlled by the concise program script (psc) file loaded before the unit operation.

With respect to the physical HW, the MS in reality communicates with the PCI ADDA interface card. This is the entirely digital process based on the direct reading of data (real values) from particular pins of the physical sensors and modules, writing data to respective pins, which are translated by the ADDA signal converter. These data pins are both inputs and outputs located on the control board allowing the access to both physical modules (meters, sensors and devices) and providing signals for controlling of experiments.

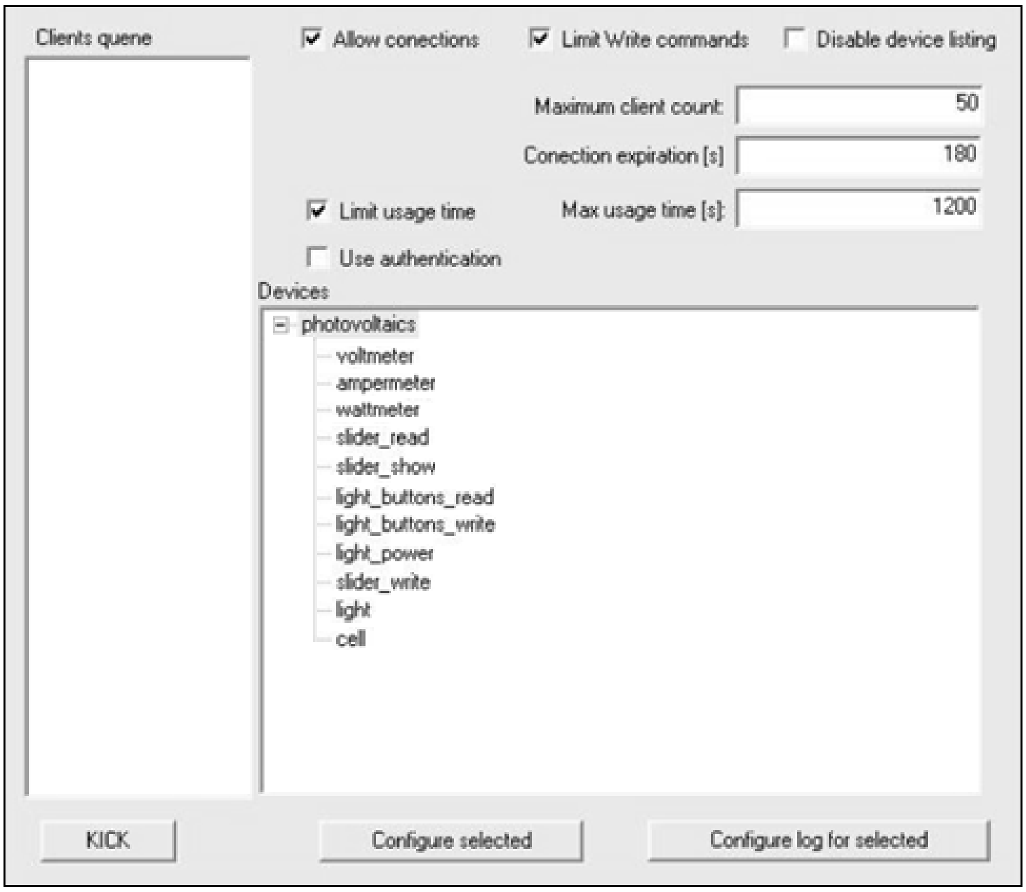
Instructions (specific commands), coming from a remote client, are processed by the listening MS (receiving service requests from connected clients through preset port). The communication is realized by standard protocols via the Internet. Some commands go via the MS translator to the REMLABNET where clients can exploit additional services like the acquirement of measured data and analysis from previously performed RLs in its database (data warehouse).

All the commands (given by psc file) are processed in MS a deterministic way by two different parsers. The first is called the LR(1) parser that processes commands from the configuration file for the purpose of the graphical user interface settings. This parser is based on static state transition tables (parsing tables), which codify a given language grammar. These parsing tables are parameterized together with a lookahead terminal that establishes the maximum tokens, the parser can use to decide, which rule it should use.

The second is the Recursive descent parser that processes commands coming from the psc file to create defined data structures and logic schemes for the RL.

It uses a general form of top-down parsing where backtracking may be involved. The parsing algorithm is based on the walking through a tree.

The MS uses for its operation several plugins providing logical schemes, mathematical functions and communication services essential to design and construct arbitrary RLs. The graphical user interface with its options to set the client's access, connection, expiration and logging is shown in Figure 4.



Clients quene

☑ Allow conections ☑ Limit Write commands ☐ Disable device listing

Maximum client count: 50

Conection expiration [s]: 180

☑ Limit usage time Max usage time [s]: 1200

☐ Use authentication

Devices

- photovoltaics
  - voltmeter
  - ampermeter
  - wattmeter
  - slider_read
  - slider_show
  - light_buttons_read
  - light_buttons_write
  - light_power
  - slider_write
  - light
  - cell

Figure 4 Graphical user interface of the MS unit to provide the clients' access, connections, expirations and logging

Imageserver unit

The Imageserver unit transports image information to connected clients. This unit periodically stores snaps (stream) acquired by the webcam, which are then directly displayed by a special object to the client's screen. It stores the image in two sizes to ensure large and small resolution depending on the Internet speed rate. If the camera is equipped with controls of position and zoom, the Imageserver is able to respond to client's requests and there is available to move the camera directions to have better view of the rig. An example presenting the structure of the Imageserver with its camera is shown in Figure 5.

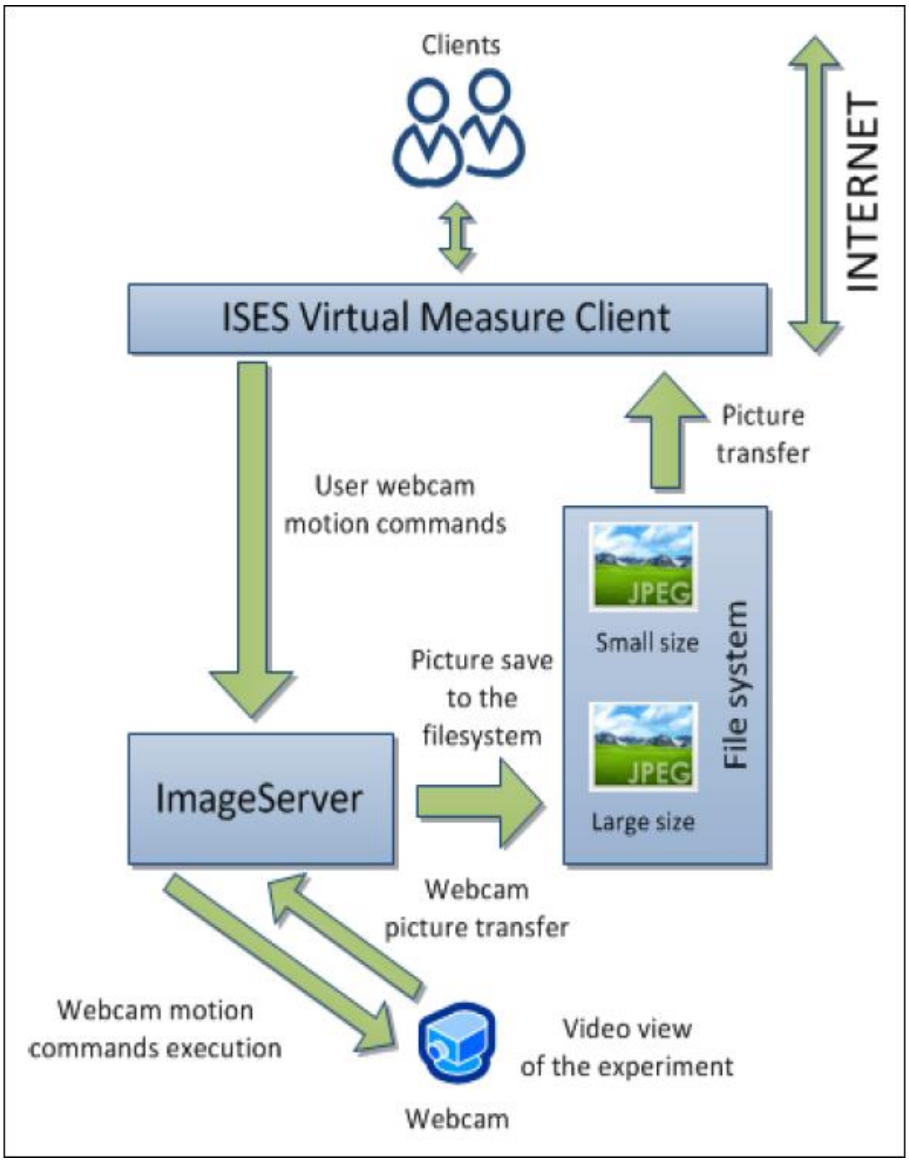

Clients
INTERNET
ISES Virtual Measure Client
Picture transfer
User webcam motion commands
JPEG
Small size
File system
Picture save to the filesystem
JPEG
Large size
ImageServer
Webcam picture transfer
Webcam motion commands execution
Video view of the experiment
Webcam

Figure 5 Diagram of the Imageserver functionality including the camera device and file system to distribute the video

Webserver unit

The Webserver unit provides the web (Nginx) services when client enters a web page of the ISES RL via the REMLABNET platform. The Nginx is an open source reverse proxy server for TCP, UDP, HTTP, HTTPS, SMTP, POP3 and IMAP protocols, as well as an HTTP cache and a load balancer. A scheme showing the Nginx deployed in the RL structure is presented in Figure 6.

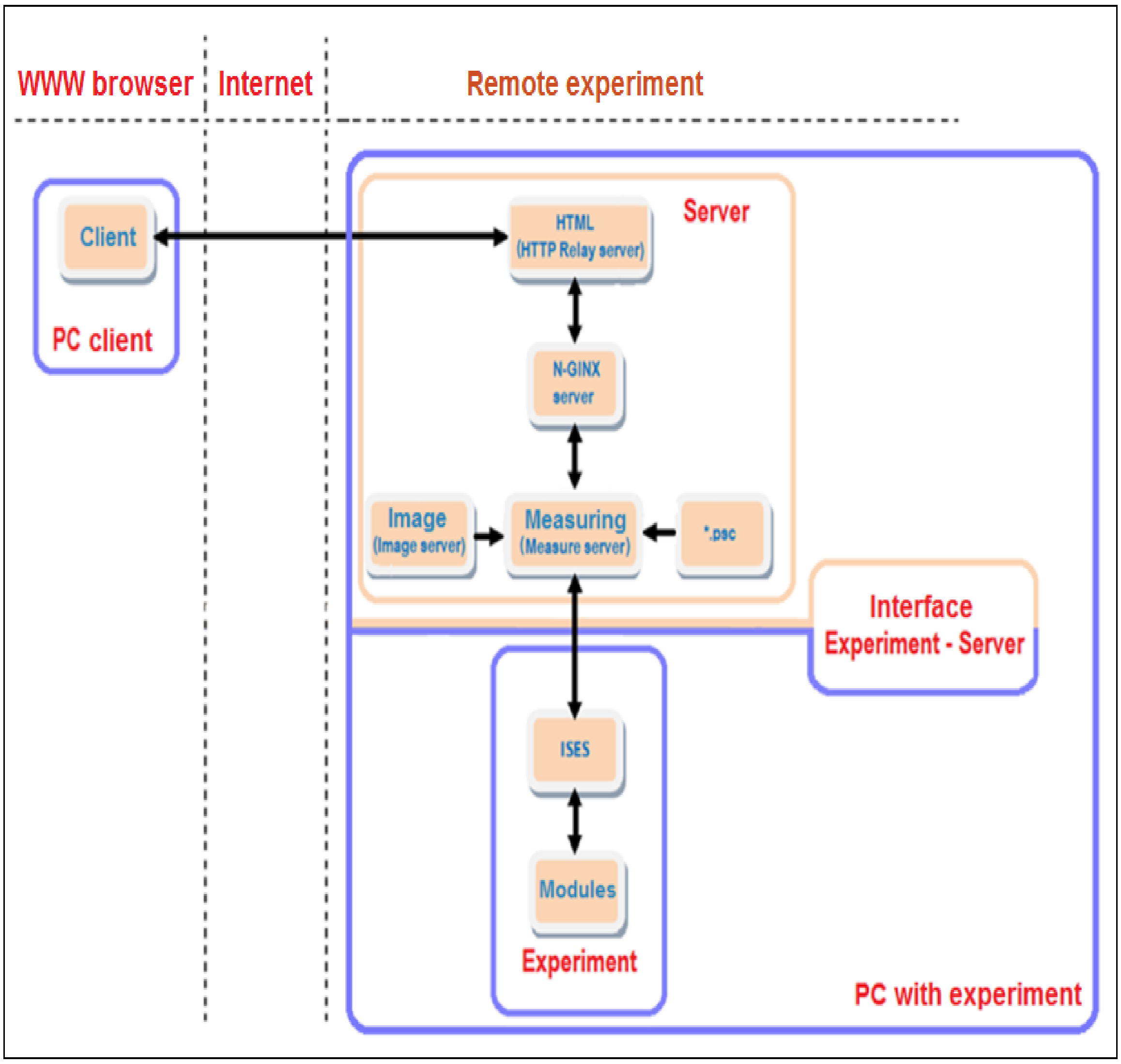
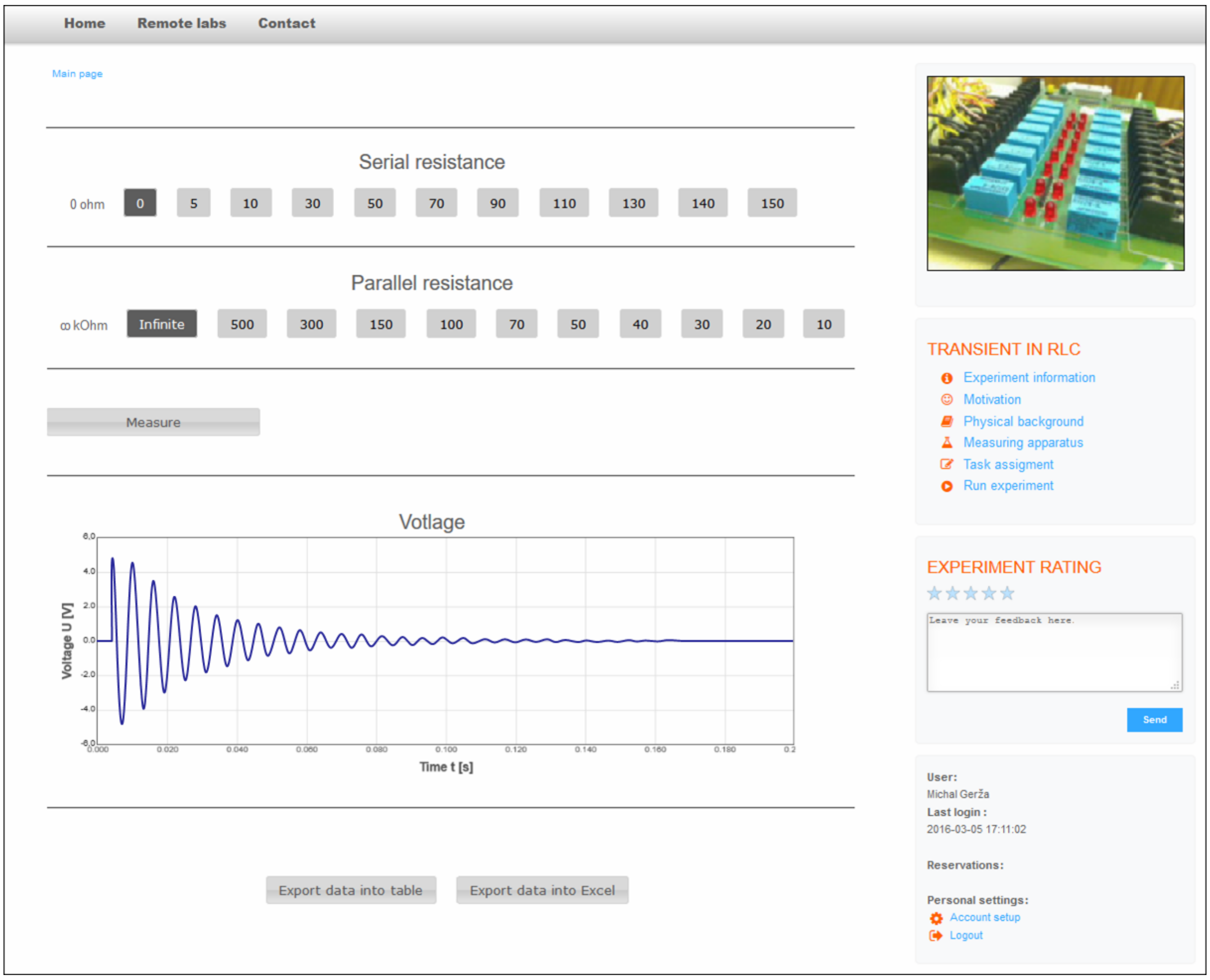


Figure 6 Structure of the ISES RL components including the Webserver unit represented by the Nginx reverse proxy server to provide web services

Webclient unit

The Webclient unit represents web pages accessible via the Internet allowing clients to work with the RLs. It was designed for two separate websites, for classic PCs and mobile devices (mobile phones and tablets). The web page recognizes what device is connecting and chooses appropriate version of the page to display. Since the mobile version is designed separately, it is completely optimized for the mobile devices. The graphical elements are reduced for using smaller volume of transferred data and a faster page loading in mobile devices. The web pages content is independent, providing different information for both versions. The example of client's web page of the RL "Transients in RLC circuits" displayed on the PC screen is shown in Figure 7.



User:
Michal Gerža
Last login :
2016-03-05 17:11:02

Reservations:

Personal settings:

Figure 7 Example of the web page of the ISES RL "Transients in RLC circuits"

Web pages produced by the first version of the ER-L were optimized for a wide screen. As controlling elements, the JavaScript widgets are used to handle the RLs. Each element is designed as a stand-alone library and added to the web page like a building block. This kind of the page can be easily transformed for individual RLs. For data displaying widgets like a graph, the value display and data record standard pre-programed widgets are used. The experiment is controlled by widgets like a button and slider. Connected clients are able to watch a live video stream coming from the experiment's web camera that is provided by the unique JavaScript function.

3.1.2 ISES finite-state machine

Let us describe in more detail the functioning of the RL software. The FSM is a mathematical model of the computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time, the actual state at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition, which is called a transition. The FSM is defined by a list of its states, its initial state, and the triggering condition for each transition [25].

The FSM, implemented inside the MS unit to process particular commands constructing the RL behavior and logic, uses two different types of functional concepts deployed as the parsers for specific reasons. The first concept is called LR(1) parser and the second one is Recursive descent parser, which are both described in the theoretical part of this doctoral thesis.

3.2 ISES Measureserver unit and its communication

The MS is very important component of every ISES RL. This unit takes care for the communication between the rig (physical HW represented by the ISES control board and its used meters, sensors and devices) and connected clients. Many techniques, responsible for the transportation of measured data from the rig’s physical modules to respective web pages and back, are utilized. As mentioned, the communication is bidirectional, from the rig’s side to the client to obtain measured data and back from the client to the rig to send control commands. When the client starts communicating, the generated analog signal goes to the PCI ADDA interface card to convert it to the digital form. After the conversion, the digital signal is transported to the MS by its plugins to gather, filter and distributes the measured data to clients by the Internet protocol suite called TCP/IP (Transmission Control Protocol/Internet Protocol).

The Internet protocol suite provides end-to-end data communication specifying how data should be packetized, addressed, transmitted, routed and received. This functionality is organized into four abstraction layers, which are used to sort all related protocols according to the scope of networking involved. From lowest to highest, the layers are the link layer, containing communication methods for data that remains within a single network segment; the Internet layer, connecting independent networks, thus providing internetworking; the transport layer handling host-to-host communication; and the application layer, which provides process-to-process data exchange for applications [26][27]. Just to shortly mention, the internetworking is the practice of connecting a computer network with other networks through the use of gateways, which provide a common method of routing information packets between the networks.

As described, the MS sends measured data to the client’s web page that is designed for the control, visualization and analysis of studied phenomena. These activities are managed by specific modules underlying on the web page. Some of them are deployed for the communication with the MS unit.

The first generation of RLs was based on the modules called Java applets (small application that is written in Java). However, after some time, the Java applets began to be restricted due to security issues (by the Oracle provider). Users now have to confirm running Java applets with warning about possible security risk. This technology basically became unusable for the interfaces of client-side web users in all applications. Moreover, the support of Java applets in contemporary tablets, smartphones and similar devices was mostly lacking. Hence, the development moved to the JavaScript objects (high-level, dynamic, untyped and interpreted programming language) to create user interfaces and communication. They are built with a new software kit called ISES SDK Remote Lab in Prague laboratory [28]. Philosophy of this new kit is the same as in case of Java applet version ISES WEB Control.

The RLs are accessible via Internet browsers supporting JavaScript and preferably the WebSocket connection. These are considered almost standard features of all modern browsers in most devices [29].

The WebSocket is a computer communications protocol providing full-duplex communication channels over a single TCP connection. The WebSocket is designed to be implemented in web browsers and web servers but it can be used by any client or server application. It is an independent TCP-based protocol having relationship to HTTP only that its handshake is interpreted by HTTP servers as an Upgrade request. The WebSocket protocol makes more interaction between a browser and web server possible, facilitating the real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between the server and remote browser. The communications are done over the port number 80 that is a benefit for those environments, which block non-web Internet connections using a firewall.

3.3 Working scheme of ISES Measureserver unit

The ISES MS, as the core software unit, is consisted of several internal and external modules, which are essential for the RL control and monitoring, data gathering and processing to provide results to connected clients. The schematic overview illustrating typical components is shown in Figure 8.

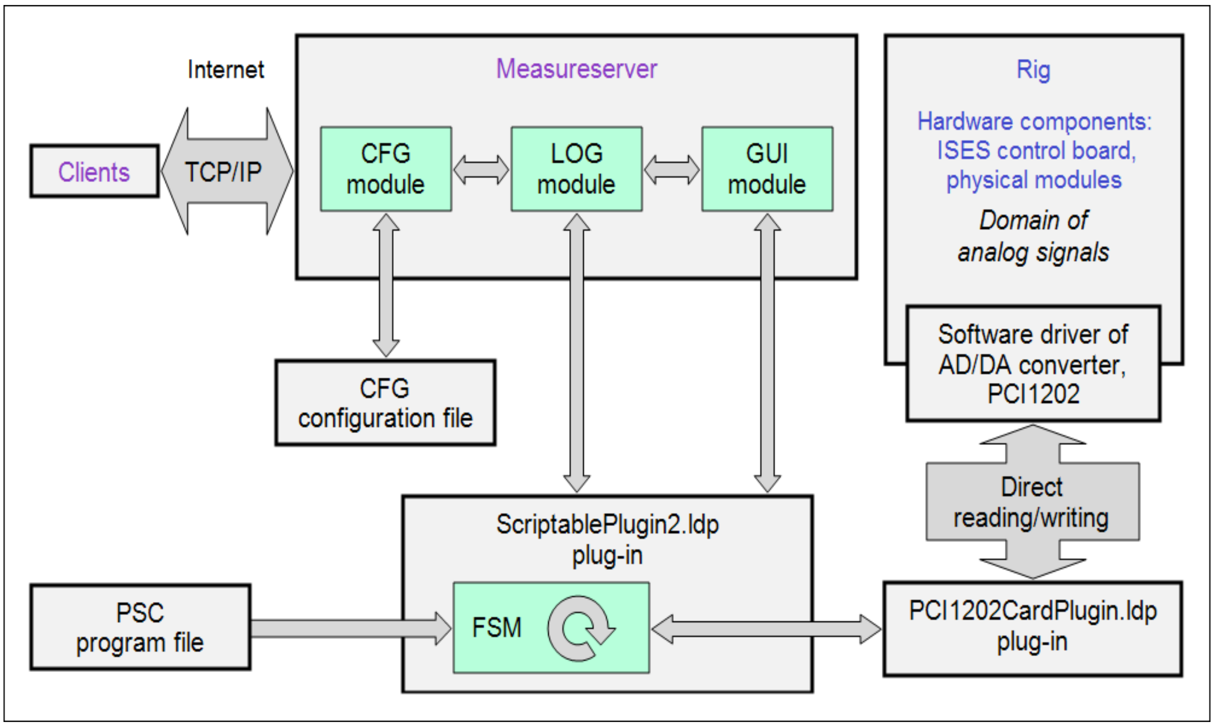
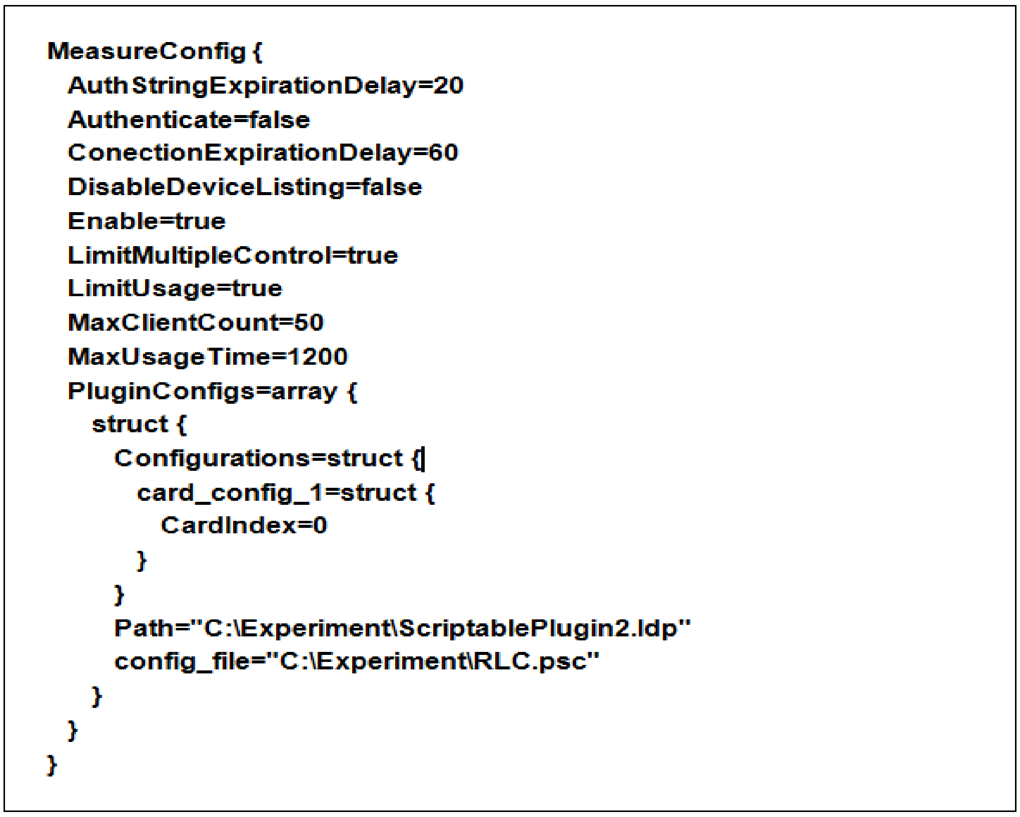

Internet
Clients
TCP/IP
Measureserver
CFG module
LOG module
GUI module
Rig
Hardware components: ISES control board, physical modules
Domain of analog signals
Software driver of AD/DA converter, PCI1202
CFG configuration file
Direct reading/writing
ScriptablePlugin2.ldp plug-in
FSM
PSC program file
PCI1202CardPlugin.ldp plug-in

Figure 8 Scheme of the internal and external modules constituting the MS unit that mediates the connection between the rig and remote clients

Configuration module

The first internal module is called CFG (configuration) intended for the initial setup of the MS behavior and the graphical user interface. During the unit startup phase, the CFG sets all the parameters needed for a proper operation. There is configured a client’s access to the system, client’s connection and expiration. It also includes a setup for the measured data logging and creation of the references to external components (psc files and underlying plugins). The CFG is drawn to read/write all the values of the parameters from/to a text file. The LR(1) parser is involved to parse this text file and to assign these values to internal structure for a consequent use. An example of the text file generated, loaded and parsed by the CFG is shown in Figure 9.



```
MeasureConfig {
  AuthStringExpirationDelay=20
  Authenticate=false
  ConectionExpirationDelay=60
  DisableDeviceListing=false
  Enable=true
  LimitMultipleControl=true
  LimitUsage=true
  MaxClientCount=50
  MaxUsageTime=1200
  PluginConfigs=array {
    struct {
      Configurations=struct {
        card_config_1=struct {
          CardIndex=0
        }
      }
      Path="C:\Experiment\ScriptablePlugin2.ldp"
      config_file="C:\Experiment\RLC.psc"
    }
  }
}
```

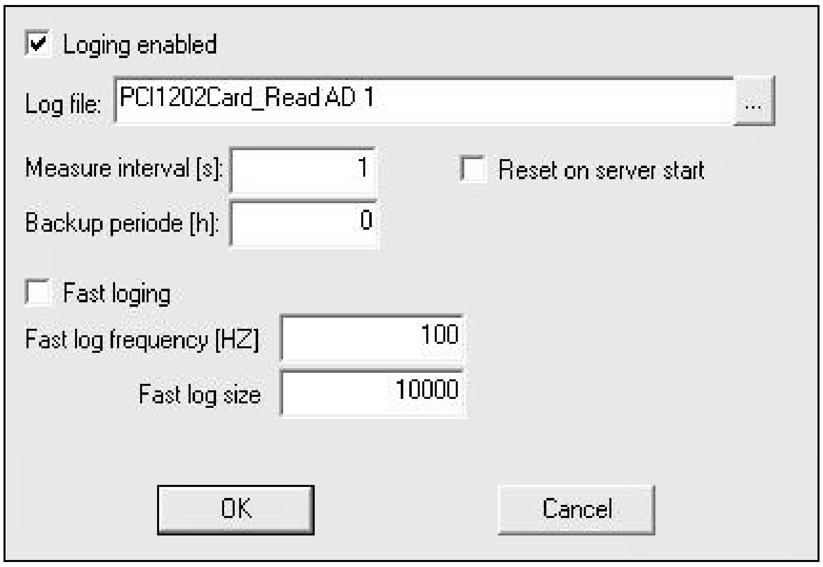
Figure 9 Example of the configuration file, containing the parameters for the initial setup of the MS unit and graphical user interface, parsed and processed by the CFG module during its startup

Data logging module

The second internal module is called LOG (data log) that is responsible for the optional logging of used pins (located on the ISES control board) to files or memory. It allows setting two ways of the data logging as follows:

- Long term log - This log stores data with the period in seconds or minutes for long-term monitoring such as temperature. The logged data are stored to files on a disk and they can survive (unless the setup log indicates that the startup erases them) when the MS restarts or exits. Each file has a place for 5000000 records (about 40 megabytes) and after filling the MS starts overwriting the oldest values in the file.
- Short log - This alternative stores data with a frequency of tens of Hz and limited length. The logged data are stored to an operational memory only, and upon the MS termination, all the accumulated data are permanently deleted. When using JavaScript widgets to manipulate with data from the short-term log, the fast log pins must be turned on first.

The LOG setup (logging options and respective parameters) is available for administrators in the MS Log config panel as presented in Figure 10.



☑ Loging enabled

Log file: PCI1202Card_Read AD 1

Measure interval [s]: 1

☐ Reset on server start

Backup periode [h]: 0

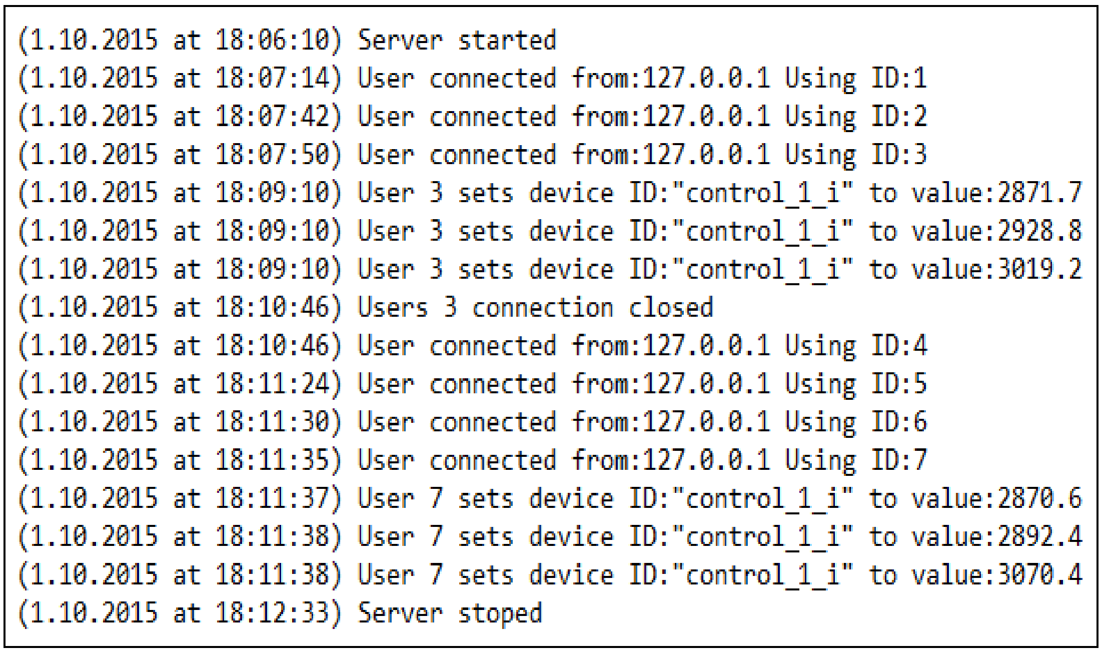
☐ Fast loging

Fast log frequency [HZ] 100

Fast log size 10000

Figure 10 Log config panel to set the long term and short data log

Furthermore, the LOG has a feature to log users' interventions exploitable for an analysis by the RL designers to optimize their experiments and for persons who are interested in the field of pedagogy. Such the users' log is listed in Figure 11 showing continuous interventions in the text file.

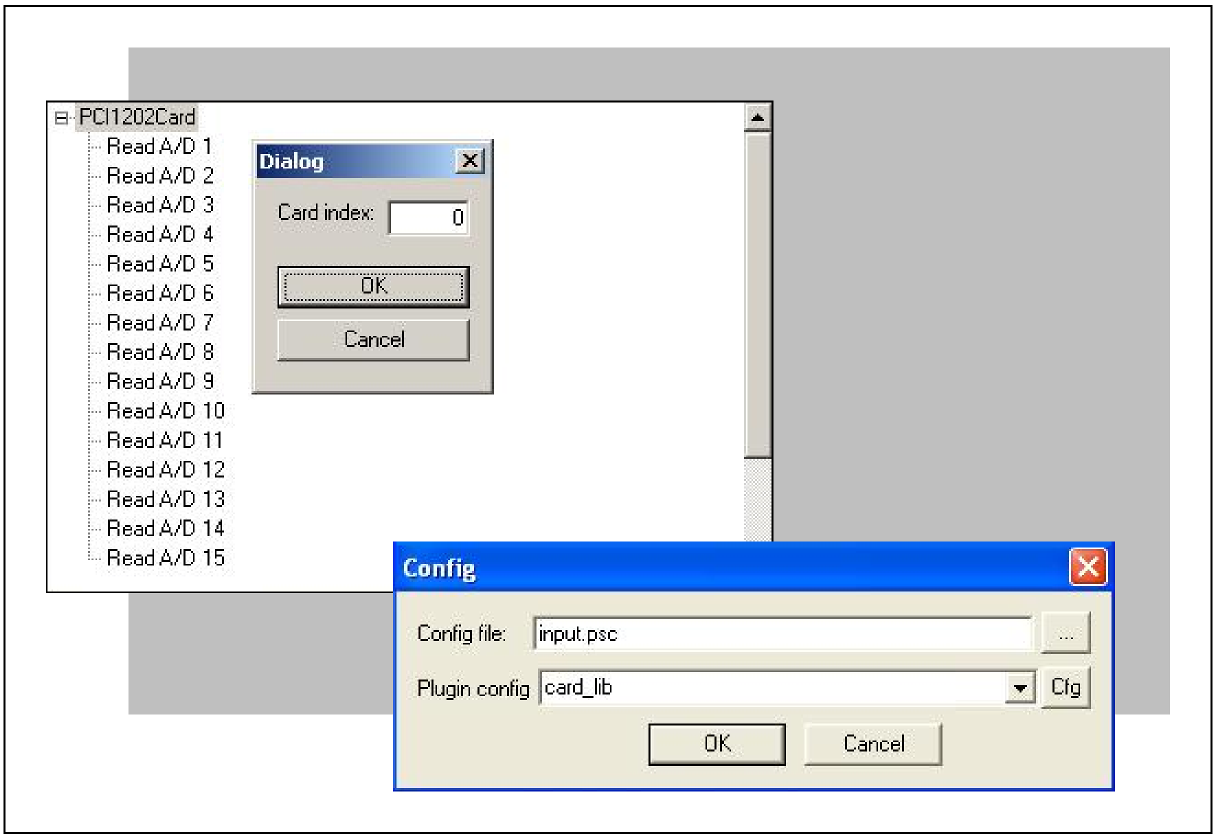


```
(1.10.2015 at 18:06:10) Server started
(1.10.2015 at 18:07:14) User connected from:127.0.0.1 Using ID:1
(1.10.2015 at 18:07:42) User connected from:127.0.0.1 Using ID:2
(1.10.2015 at 18:07:50) User connected from:127.0.0.1 Using ID:3
(1.10.2015 at 18:09:10) User 3 sets device ID:"control_1_i" to value:2871.7
(1.10.2015 at 18:09:10) User 3 sets device ID:"control_1_i" to value:2928.8
(1.10.2015 at 18:09:10) User 3 sets device ID:"control_1_i" to value:3019.2
(1.10.2015 at 18:10:46) Users 3 connection closed
(1.10.2015 at 18:10:46) User connected from:127.0.0.1 Using ID:4
(1.10.2015 at 18:11:24) User connected from:127.0.0.1 Using ID:5
(1.10.2015 at 18:11:30) User connected from:127.0.0.1 Using ID:6
(1.10.2015 at 18:11:35) User connected from:127.0.0.1 Using ID:7
(1.10.2015 at 18:11:37) User 7 sets device ID:"control_1_i" to value:2870.6
(1.10.2015 at 18:11:38) User 7 sets device ID:"control_1_i" to value:2892.4
(1.10.2015 at 18:11:38) User 7 sets device ID:"control_1_i" to value:3070.4
(1.10.2015 at 18:12:33) Server stoped
```

Figure 11 Users' log file containing the timestamped records about interventions performed by the clients

Graphical user interface module

The third module is called GUI (graphical user interface) also determined for the RL designers and administrators to easily configure the MS throughout its operational life. The GUI allows the setup of the MS functioning, connected client and plugins. It provides a set of the dialog boxes with input/output parameters. Each attached plugin has its own set of parameters; hence the GUI is partially dynamic according the RL's rig configuration. There are covered many various parameters, which can be optionally set. The suitable example is shown in Figure 12 with the two dialog boxes to configure plugins. The upper dialog box offers entering the plugin card index. The lower one represents the Config dialog box used for entering the psc control program.


PCI1202Card
Read A/D 1
Read A/D 2
Read A/D 3
Read A/D 4
Read A/D 5
Read A/D 6
Read A/D 7
Read A/D 8
Read A/D 9
Read A/D 10
Read A/D 11
Read A/D 12
Read A/D 13
Read A/D 14
Read A/D 15

Dialog
Card index: 0
OK
Cancel

Config
Config file: input.psc
Plugin config card_lib
OK
Cancel

Figure 12 Dialog boxes allowing the setup of used plugins in the GUI module; the first (upper) enables to set the Card index of the ADDA interface card, the second (lower) requires entering the Config file (psc control program) and Plugin config (reference to the interface card) to communicate with the ISES control board

External modules

The remaining modules are called ldp plugins, which are external by reason of their specific deployment according to types of the signal converters and rigs.

They have the file extension .ldp (dynamic-link library file) to distinguish them as the plugins providing a fixed set of functions for various purposes like mathematical operations, values remapping and clamping. The most frequently used plugin is the ScriptablePlugin2.ldp, and the PCI1202CardPlugin.ldp that underlies at the lowest layer communicating with the converter.

The ScriptablePlugin2.ldp plugin is the strongest and the most difficult of the available plugins. There is possible to achieve with this plugin the equivalent functionality of almost any of the other plugins with an exception of those, which control the specific hardware components. Moreover, there is possible to reach some of the activities, which older plugins cannot perform as is the generation of more complex signals or high-speed experiments. The principle of its operation is based on the psc control program allowing the operations for individual pins located on the ISES control board.

The PCI1202CardPlugin.ldp plugin is used to control the PCI-1202 ADDA interface card. The plugin allows selecting the interface card if a computer has more such the cards by entering the Card index number. This interface card serves the following input and output pins of the ISES control board:

- *Input pins* - For example, the pin “Write D/A 1” allows writing a 12-bit value to the analog output of the interface card. The pin “Write D 1” allows an entry into the 16-bit digital output.
- *Output pins* - For example, the pin “Read A/D 1” can read a 12-bit analog input from the card. The pin “Read D 1” reads a16-bit digital input.

The scheme presenting the ldp plugins frequently used for the MS, related to their communication relationships, is shown in Figure 13.

The further plugins used for various purposes are roughly introduced in the following bullets to present a wide spectrum of the external modules:

1. *ISESCardPlugin.ldp* - The plugin controls the standard interface card that is not supported under Windows NT series.
2. *ISESProCardPlugin.ldp* - The plugin controls the professional interface card that is not supported under Windows NT series.
3. *TestingDevicePlugin.ldp* - This plugin acts as a repository of values that can be used by clients or other plugins to store experiment states.

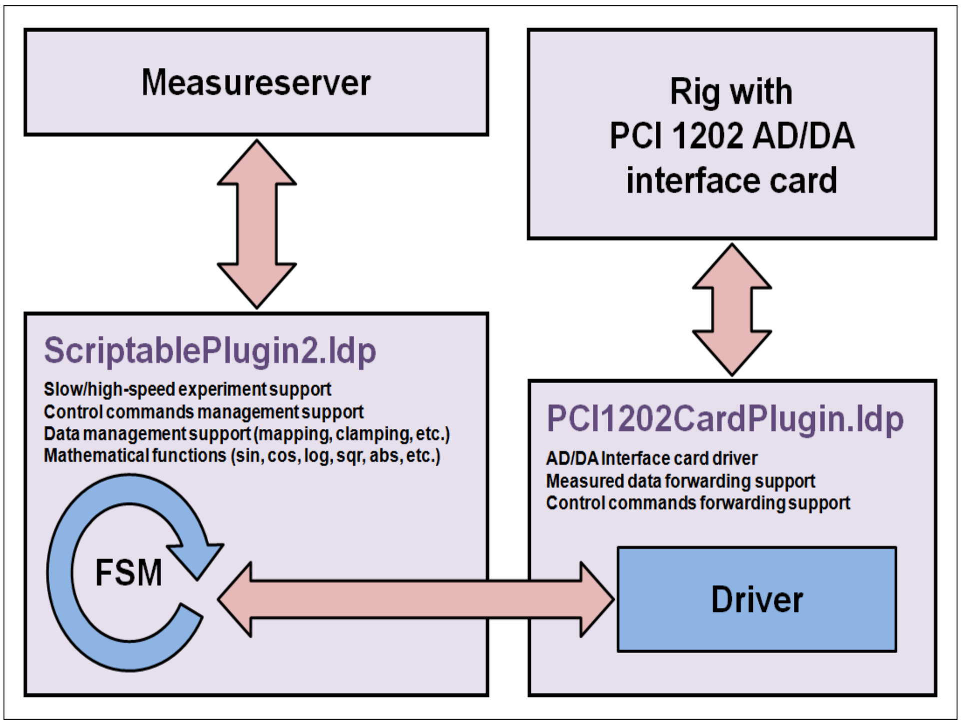

Measureserver
Rig with PCI 1202 AD/DA interface card
ScriptablePlugin2.ldp
Slow/high-speed experiment support
Control commands management support
Data management support (mapping, clamping, etc.)
Mathematical functions (sin, cos, log, sqr, abs, etc.)
FSM
PCI1202CardPlugin.ldp
AD/DA Interface card driver
Measured data forwarding support
Control commands forwarding support
Driver

Figure 13 Scheme of the ldp plugins used with the MS related to their intercommunication and characterization

4. *MixingPlugin.ldp* - Some plugins exploit another plugin for operating. This specific plugin allows such the plugins to use further plugins in that it exports their pins, as if they belong to the mixing plugin and it passes data into plugin, to which appropriate pins underlie.
5. *ConfLogicPlugin.ldp* - Main features of this plugin are focused on the implementation of the experiments logic of the server side. In principle, it is a set of rules describing how to get a value from the output pin or what to do depending on the value written to the input pin.

Control program

The psc control program is stored in a file with the extension psc that contains the complete logic according to which the RL operates. There is also the section loading single devices (pins), which are associated with specific widgets (small functional objects) placed on client’s web pages. This file is coded in a script whose syntax is similar to the C language. The psc control program is described in the next chapters related to other issues. A snippet of the coded experiment containing the body of its logic is shown in Figure 14.

```
version 3.3
name experiment

step_frequency 200
variable output = 0

state generator
{
    variable level = 0.0
    variable frequency = 2.0
    variable amplitude = 1.0
    variable phase_offset = 0.0
    timer generator_timer

    step
    {
        variable output_factor = 0.0;
        output_factor = sin(input_factor * 6.28);
        output = level + output_factor * amplitude;
    }

    pin_read last_value
    {
        result = output;
    }

    pin_write restart
    {
        generator_timer = 0;
    }

    # rest of the coded experiment logic
    # ...
}
```

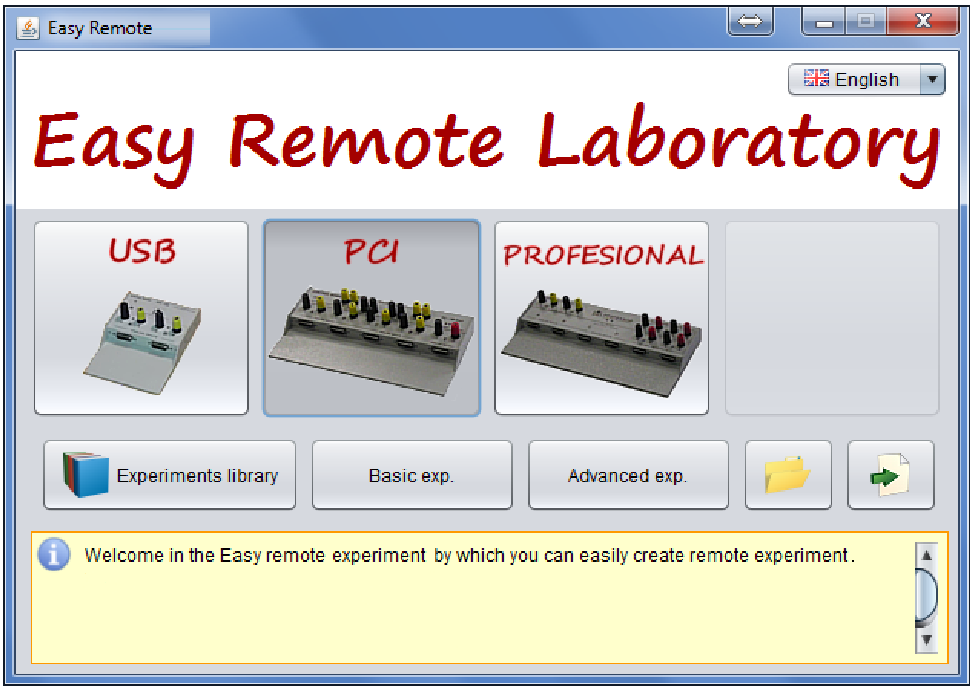
Figure 14 Snippet of the experiment "Sine generator" included in the psc file; the section "state" is started once, the subsection "step" calculates the sine in a loop based on the frequency "step_frequency", the handler "pin_read" and "pin_write" communicate with widgets on the web page

3.4 Easy Remote Laboratory expert system tool

The ER-L is a new user environment exploited to simplify a process of the designing, building and maintaining the RLs by laymen. The expert system tool has been developed since 2012 at Tomas Bata University in Zlín by Michal Krbeček, Ph.D. student. The environment was first based on the Java applets. However, when the Oracle Company started restricting the Java applets usage because of security risks, and the web browsers started prohibiting their displaying on web pages, then a new version of the ER-L environment has been created since 2014. The new version is based on the JavaScript objects used as the progressive technology providing easy and secure implementation of widgets intended for the RL web pages.

The ER-L works as the plug and play controlling program that compiles the screen questionnaire similar to the expert system. It means, the program emulates the decision-making ability of a human expert (experienced user). The ER-L disposes of the graphical development environment creating and exporting the psc file and the RL web page with widgets.

When the ER-L started, the welcome menu with photos of ISES versions appears that shows the basic options as depicted in Figure 15.



Welcome in the Easy remote experiment by which you can easily create remote experiment.

Figure 15 Welcome panel of the ER-L providing its graphical development environment with the experiment components

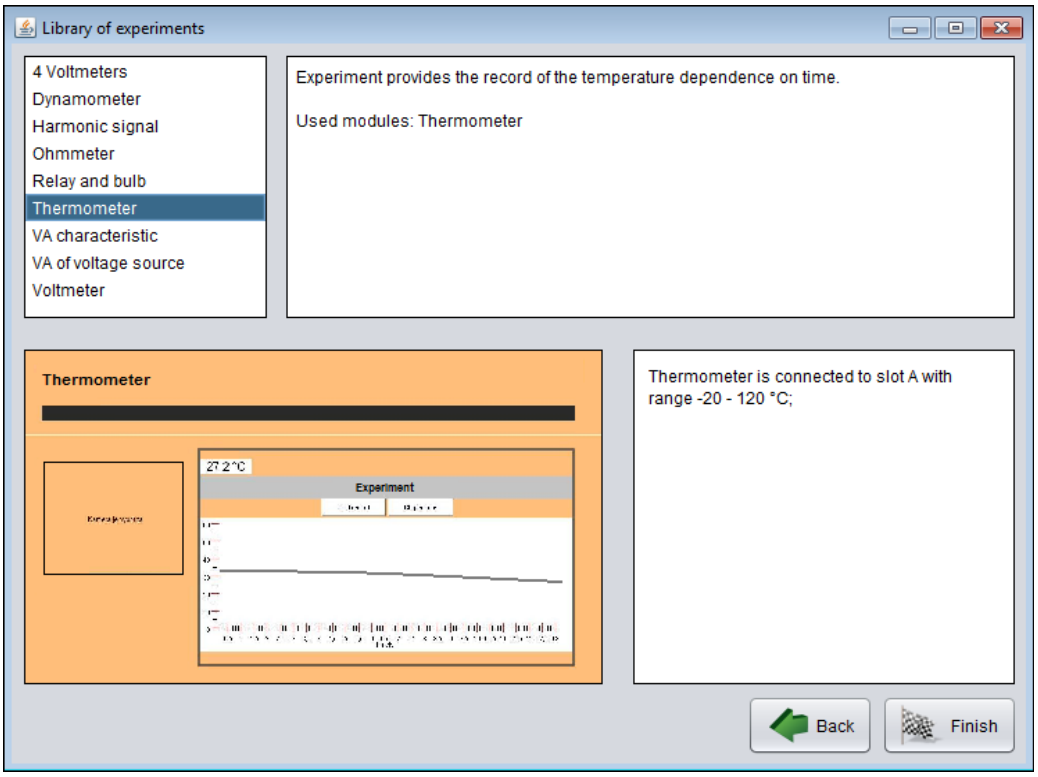
The welcome panel also allows the choice of language to open the saved projects. For practical reasons, especially for designers, the RLs are divided into three categories according to their complexity as listed below [30]:

1. Starting level - The set of precompiled typical RLs,
2. Basic level - The most used RLs with variability and logic,
3. Advanced level - Very complicated RLs with specific devices.

All the RL complexity levels are described in further subsections to offer readers the overview about their options and setup.

Starting level

The simplest way of the RLs building is to exploit the library of common experiments that the user can choose, automatically install and start. The library is accessible by the button “Library of experiments”, located in the welcome panel. The window, as shown in Figure 16, appears to start the RL configuring. In the upper left part of the window, there is a list of available experiments for the choice. When the experiment is chosen from the library, the respective description of the experiment appears in the right panel and beneath the list of hardware modules required to use. At the bottom left, there is a preview of the corresponding website layout. If the user points a mouse to the edge of this screen (not accessible now) the arrows for the next screen appear and the photo of devices and modules are displayed. In the last part, there is a text with the description of the experiment. The final step of the RL starting level compiling, it is only necessary to press the “Finish” button. After the pressing, the program generates all important components of the designed experiment (psc file and web page code) and then returns to the start menu. At this point, the experiment is already operational, and the user can deploy and start it to use. The whole procedure of the RL compiling takes approximately 30s.



Experiment provides the record of the temperature dependence on time.

Used modules: Thermometer

Thermometer is connected to slot A with range -20 - 120 °C;

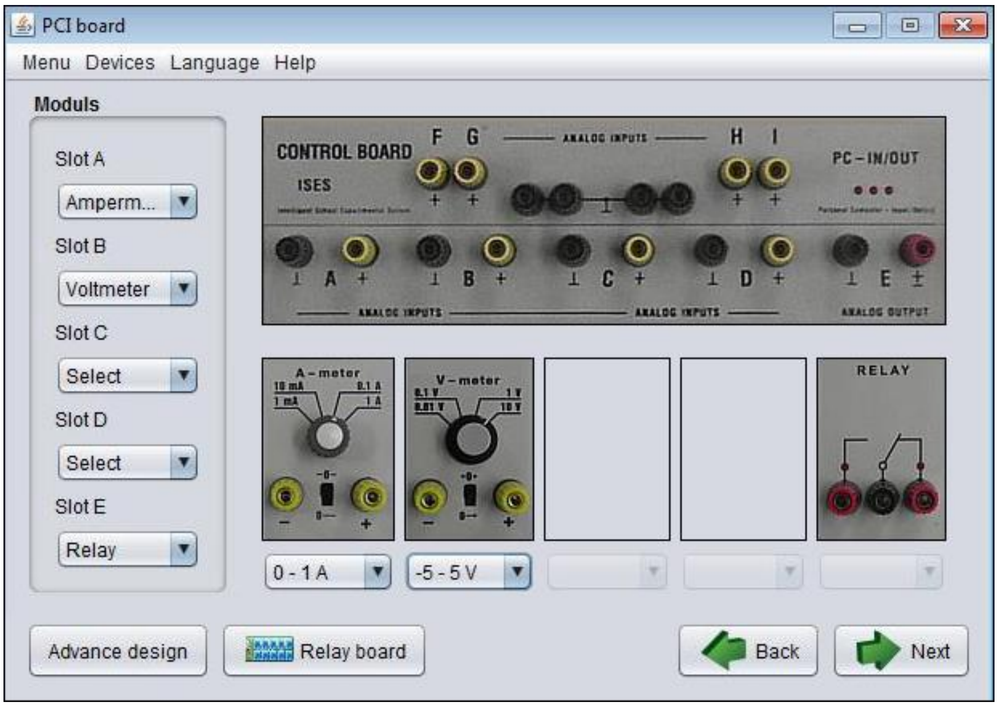
Figure 16 Experiments library window of the RL starting level intended for the experiment selection, its modification and compilation

Basic level

In the basic level of RL compiling, the ISES version has to be duly specified first. The main options are offered to users in the welcome panel. The window for the selection of measuring/controlling modules and their ranges adjustment is subsequently displayed as shown in Figure 17.

The selection of modules is performed by the pull-down menu on the left side of the window. When a module is selected, its photo appears in the appropriate slot of the ISES control board. Its working range is adjusted using the pull-down menu, available below the respective module.

If the experiment is designed to use a relay board, one enables this option by pressing the “Relay board” button. Once pressed, the window appears with a relay board photo by which the user may connect and activate the 2x8 relays. This program then asks for details of the individual settings of the relays. As an example, the user may choose the manual activation with a button placed on the web page, or after meeting any comparative condition.



Slot A: Amperm...
Slot B: Voltmeter
Slot C: Select
Slot D: Select
Slot E: Relay
0 - 1 A
-5 - 5 V

Figure 17 Modules selection window of the RL basic level that provides the intuitive setup for the ISES control board slots

Advanced Level

The advanced level of RL implementing is based on the individual steps of the corresponding flowchart diagram and its transfer of the psc file by using the pre-prepared control blocks from the psc library. In this way, it is possible to assemble very complex control logic for the RLs. This method requires a certain amount of the creativity and logical thinking concerning the sequence of the experiment actions. The advanced level starts on the module selection window by pressing the “Advanced design” toggle button. The process of the advanced design is similar to the basic one, where the modules selection and the web page creation are necessary at first. The advanced level window is then displayed as presented in Figure 18. The window left part offers a tree structure providing the control logic of the experiment. On the right side, there is located a list of the blocks, which can be inserted into the tree structure and the variables list.

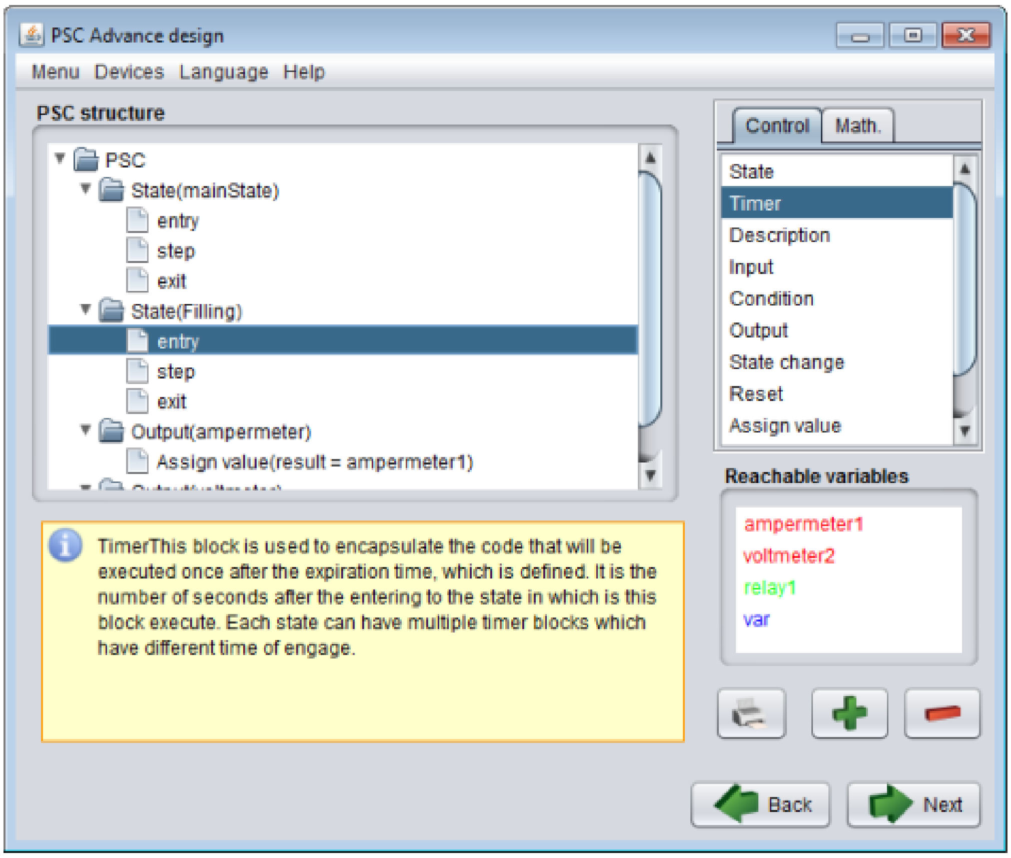

TimerThis block is used to encapsulate the code that will be executed once after the expiration time, which is defined. It is the number of seconds after the entering to the state in which is this block execute. Each state can have multiple timer blocks which have different time of engage.

Figure 18 Blocks selection window of the RL advanced level to design and configure the complex experiment components

The future improvement of the ER-L environment could be inspired by the new integrated development and training system for FSM based approaches called GIFT as a very progressive system introduced by Heinz-Dietrich Wuttke at Ilmenau University of Technology in Germany [31].

3.5 RLMS REMLABNET

The REMLABNET is a platform that integrates and manages the RLs for starting university level and secondary schools. Its building was initiated both from the extensive use and expertise in ISES, the built RLs and the lack of similar systems for secondary schools and universities in Europe. Another added value of REMLABNET is the increased reliability of the delivered RLs. It has been developed since 2014 at Tomas Bata University in Zlín as well. The first version of the REMLABNET platform was based on Java applets exposed on the clients' web pages. When Java applets became restricted to use safely (as mentioned in subchapter 3.4), then a new version of the REMLABNET came, based on JavaScript widgets. The second version deployment started in 2015 including new features. It also cooperates on a federation scheme with the Graasp platform in Lausanne, Switzerland (www.graasp.eu) in the form of direct exposing the RL graphical interfaces on the allied platform and EU RLMS project Go-Lab (www.go-lab-project.eu).

The REMLABNET uses new components, designed for the purpose, as web space management, data warehouse and communication board, uses two level diagnostics and forwards to client's embedded simulations [32] and others. The communication server provides, beside the connection and diagnostics, also envisages services for the teacher's comfort as IP telephony, white board, simulation inclusion, test management and reservation management. For the sake of safety, optimal access to all experiments and economical exploitation the virtualized cloud is used. The schematic arrangement of the units and the communication relationships in REMLABNET is presented in Figure 19.

As mentioned, this platform is consisted of many diversified components, where the six ones are more important, hence they are worth to describe:

1. *Measureserver unit* - This integral component of the platform is described at full length (concept and operation) in subchapter 3.1.1.
2. *Diagnostics server* - This component ensures keeping track of the current status of all experiments connected to the RLMS. Depending on the experiment activity, its breakdown or failure the status will be displayed at the RLMS access portal. The diagnostic system also allows sending commands to the experiment in case of detected faults. From the description of the functionality, it is clear that the direct communication is

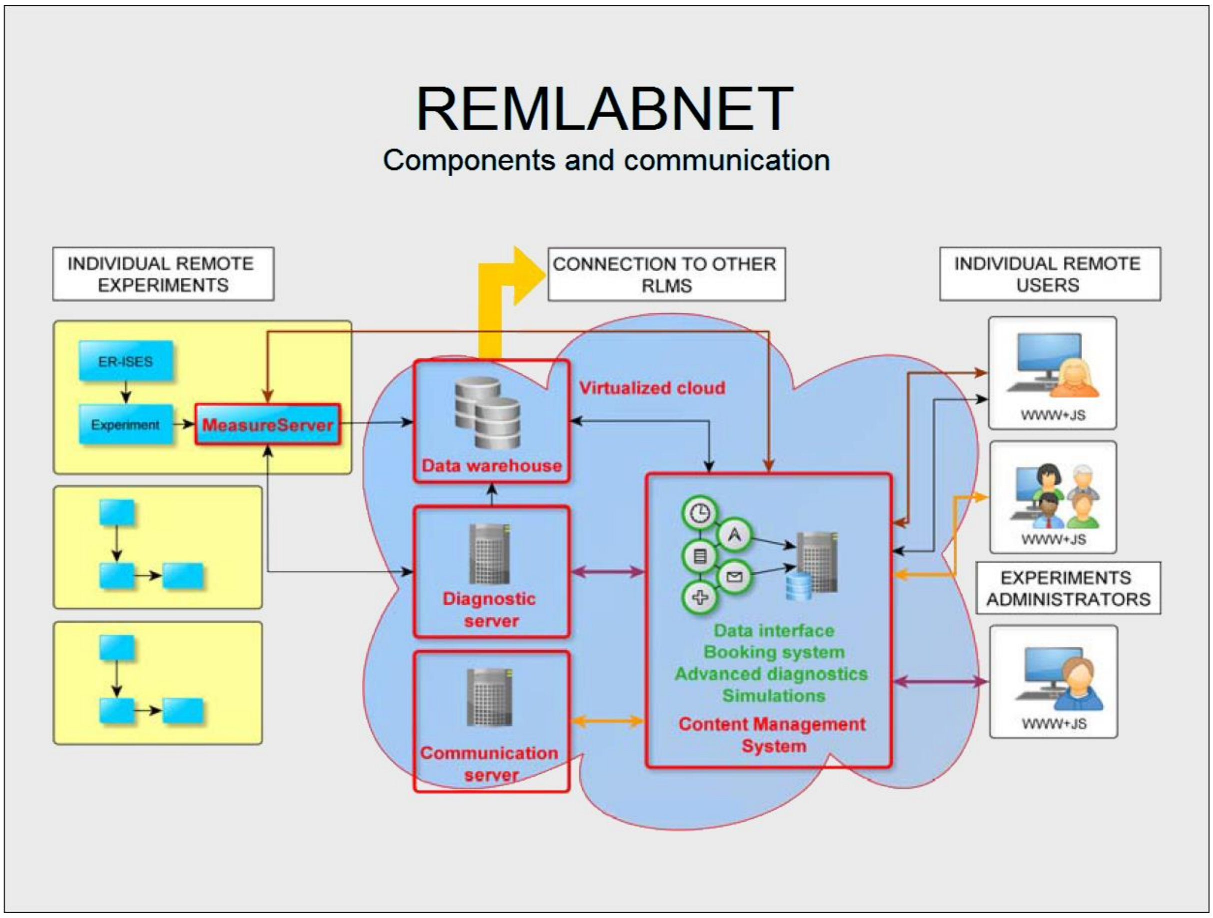


Figure 19 Scheme of the REMLABNET platform covering the ISES RLs with physical experiments, various services and connected clients

needed with the MS of the experiment side because there can be only found the latest information about its availability and status.

3. *Data warehouse* - This is a part of the system used for the data storage and analysis. It is a centralized data storage providing extended analytical services for the MS, Webserver, Imageserver and other components of the RLMS. The data warehouse also includes a number of sophisticated instruments providing data analysis from individual rigs. The analysis may detect and filter existing noise or measurement errors.

4. *Communication server* - This is a subsystem designed for the transmission of information and real-time communication, interaction and collaboration in teaching and learning process with RL. The communication server provides an insertion of the RLs in social networks and allows working with experiments in groups and enabling mutual help among groups. It also serves as the interface for connecting to internal knowledge bases, and as a global knowledge base like the Wikipedia encyclopedia.

5. *Content management system* - The RLMS includes a portal based on the latest HTML5 standards. It ensures portability and compatibility for a wide range of devices such as mobile phones, computers, tablets and many more. The portal meets the demands for an easy work in the laboratory and the work scope with modern techniques. The whole environment is represented in the form of dynamic web pages.
6. *Virtualized cloud* - The whole system runs in a virtual cloud space. This ensure sufficient security, connectivity and 24/7 accessibility.

The REMLABNET portal serves as an access point both for signed users and experiment administrators. For these roles, the special functions are provided. Besides them, other three roles are available. They are defined as a teacher, unsigned user and system administrator. All these roles with their respective functions are shortly described in the following bullets:

1. *Unsigned user* - He has an access to all the experiments' materials and information. Such user can also take measurements on the experiment but only if there is no signed user connected.
2. *Signed user* - He has a priority over an unsigned user in the term of measurements at the experiment. This user can also make a reservation for measurements at the specific time. Measured data coming from all the experiments are saved in the user's account. The attached library provides an access to users' historical measurements. The signed user is also allowed to use all the communication functions.
3. *Teacher* - He provides more options compared to the common user. There is more freedom in the term of the experiment reservation (such teacher is allowed to create more reservations for his lectures). A teacher is also able to administrate the virtual classrooms and assign or remove the control of the experiment running for connected clients.
4. *Experiment administrator* - He is an owner of the experiment. This role provides an access to the diagnostic interface of the experiment where the detailed status of the experiment is available. An administrator is informed by an email message when there is some fault. He is allowed to freely change the experiment information and supporting materials.
5. *System administrator* - The role provides full rights to all the functions, interfaces and content implemented is the system.

3.6 Deficiencies and drawbacks of ISES Measureserver

The ISES RL has some deficiencies and drawbacks hampering a wide use of this system, especially for the RL designers and administrators. Some of them concern MS unit as a major component of RL. This unit had in past several deficiencies as the inferior data archiving, extended clients' activities logging, MS unit & physical modules diagnostics and embedded real-world phenomena simulation. All these deficiencies were historically obstacles for improving the LRs and also for moving to a higher level of design and administration.

First, the absence of high level data storing, prevented to perform detailed analysis in REMLABNET. Second, next drawback related to the absence of the data archiving prevented the data reuse by clients who need to have access to previous measurements. Connected with this, the data transport to the clients is not implemented optimally by the reason of excessive data quantity. Third, it particularly concerns the low transmission speed and its stability.

Fourth, the absence of the clients' activities logging leads to problems that the designers and other data evaluators have no access to the complete statistical data about connected clients and their behavior. They need to know in detail how clients proceed during their experimentations, whether they perform the required measurements correctly or they systematically/occasionally fail.

Fifth, the next drawback of MS was the absence of any diagnostics focused on the MS modules (FSM, CFG, GUI, plugins) and the ISES physical modules (meters, sensors and devices). This feature is especially important for the RL administrators who maintain software components and attached rigs. This diagnostics should help to efficiently check and monitor the RL operation. In case of any failure or sudden crashing, it should immediately choose an appropriate scheme to solve the occurred problem.

The last, sixth drawback, is the absence of the embedded real-world phenomena simulations running concurrently together with the real experiments. Teachers and interested users demand this feature to help understanding examined phenomena by simple and easy understandable means.

The design and implementation of appropriate solutions of enumerated problems in MS of ISES remote laboratories, is the subject and goal of presented doctoral thesis. These solutions should contribute to a better reliability and scalability of the RLs to reach contemporary standards in this area.

4. GOALS OF THE DOCTORAL THESIS

The goals of this doctoral thesis are focused on the software improvements and development related to the MS unit and its components given below:

1. **Measured data archiving to structured data files -** It is responsible for the gathering, filtering and saving measured data and experiment metadata in the specific data format to an xml file, including the files dispatchment to the data warehouse to archive and for the detailed analysis,
2. **Continuous clients' activities monitoring to text files -** It allows logging of all the activities performed by connected clients, when experimenting, and saving to a log file, with an option of the files dispatchment to the data warehouse for the purpose of didactical analysis,
3. **First level diagnostics of the integrated software modules -** It provides the remote laboratory administrators the notifying and warning messages to ensure the Measureserver unit functioning with the aim of avoiding or reducing occasional failures caused by various influences,
4. **Second level diagnostics of the physical hardware modules -** It gives the remote laboratory administrators the features to detect and monitor the physical modules, like meters, sensors, probes and devices, connected to the rig to prevent their failures or disconnections,
5. **Embedded simulation of real-world phenomena and its visualization with the control on the client's web page -** It represents the simulation approach running concurrently and synchronized with the real experiment by the integrated solvers to provide approximate numerical solutions used for a motivation before the real measurement as the introductory step to better acquaint the measured phenomenon.

5. METHODS USED

This chapter deals with the theoretical background needed for understanding particular principles, concepts and ideas used for the implementation of proposed goals in the practical part of this doctoral thesis. There are important fields, related to the controlling the behavior of experiments, which include the input grammar concept and the grammar parsing principle, leading to the data measurement process, data distribution and analysis [33][34][35].

5.1 Finite-state machine classification and concept

There are many ways of controlling the behavior of systems, and the use of state machines is one of the oldest and best known. State machines allow us to think about the “state” of a system at a particular point in time and characterize the behavior of the system based on that state. The use of this controlling technique is not limited to the development of software systems. In fact, the idea of state-based behavior can be traced back to the earliest considerations of physical matter. For example, H_2O can exist in three different states easily observable in nature: solid (ice), liquid (water) and gaseous (steam, fog, clouds). In each of these states, the behavior of H_2O is different. The means of forcing transitions between the three states is also well-defined. Many other natural and artificial systems may also be controlled by the states the system can occupy, the behavior in each of those states, and how the system transitions are between these states, including which states are connected and which are not.

The similar technique can be used to design software systems by identifying what states the system can be in, what inputs or events trigger state transitions, and how the system will behave in each state. In this model, there is seen the execution of the software as a sequence of transitions that move the system through its various states [36]. Concepts and techniques are described in the next subchapters to better understand the FSM models implemented inside of the MS unit as the LR(1) parser and Recursive descent parser.

5.2 Language parsers

A natural language parser is a program that works out the grammatical structure of sentences, for instance, which groups of words go together and which words are the subjects or objects of a verb.

Parsing, or more formally, syntactic analysis, is the process of analyzing a text, made of a sequence of tokens (for example, words), to determine its grammatical structure with respect to a given formal grammar.

In computer technology, the parser is a program that mainly works with some of context-free grammars. It is usually a component part of a compiler that receives input in the form of sequential source program instructions, interactive on-line commands, markup tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or options) that can then be managed by other programming (for example, other components in a compiler). A parser is usually used to check that all the input has been provided that is necessary [37]. The flowchart analyzing a source string by the typical parsing process is illustrated in Figure 20.

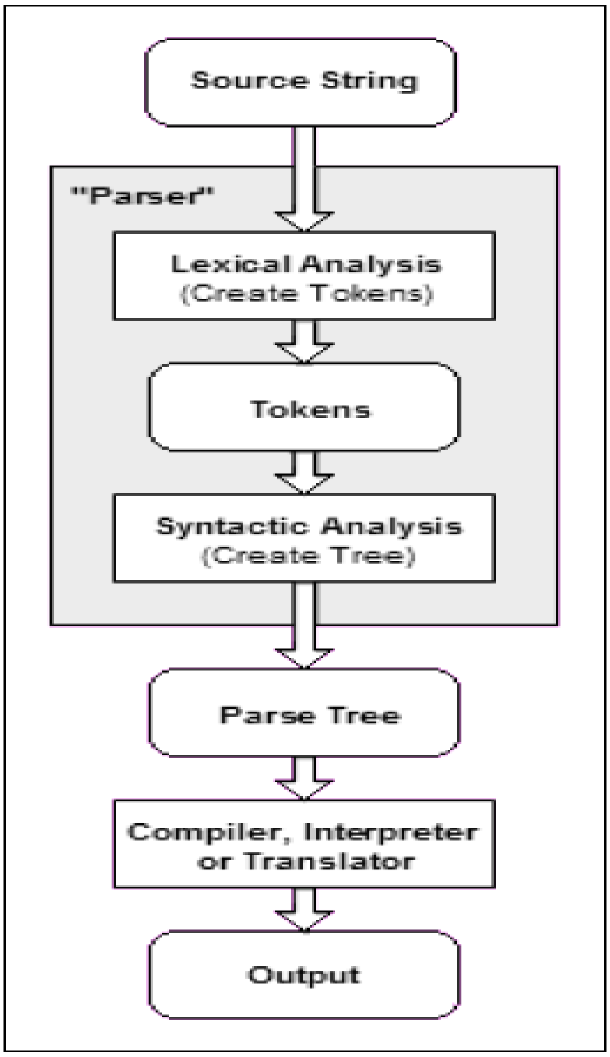


Figure 20 Flowchart of the source string processing in the typical parser

LR(1) parser

The first concept is a canonical LR parser more frequently called the LR(1). It has been designed to use the bottom-up parsing approach. The theoretical background of the bottom-up LR(1) parser is based on the formal mathematics behind this FSM (parsing machine) operation that reads lexical symbols from a source sentence provided as its input, and it proceeds to recognize productions comprising a particular grammar. It means, a table-driven algorithm can be produced for any given grammar (such as a computer programming language), which parses and recognizes valid sentences.

The LR(1) is an LR(k) parser (Left to right, Rightmost derivation parser) defined for k=1 that uses a single lookahead terminal. The LR(k) is a type of shift-reduce parser, as a generalization of existing precedence parsers. Just to mention in brief, the shift-reduce parsing uses two unique steps. These steps are known as shift-step and reduce-step. The LR(k) has the potential of recognizing all deterministic context-free languages and can produce both left and right derivations of statements encountered in the input file. The special attribute (advantage) of this parser is that all the LR(k) parsers (their grammars) with k>1 can be transformed into the LR(1) parser.

The grammar parsed is simply a set of symbols and rules defining the required language. More precisely, the symbols and rules define valid sentences of the grammar. Particular sequences of symbols from the grammar form a sentence, and if that sequence obeys the rules of the grammar, that sentence is said to be a valid sentence with respect to the grammar. If a sequence of symbols from a grammar is fed into LR(1) parser, this machine can determine whether or not the input symbols form a valid sentence of the grammar. If the sequence is valid, the LR(1) parser accepts the input sentence without error, otherwise the parser may detect one or more syntax errors in the input sequence.

The parser operation is based on static state transition tables. These codify the grammar of the language it recognizes and are typically called parsing tables. They are parameterized with a lookahead terminal [38].

Recursive descent parser

As the second functional concept used is the Recursive descent parser that is a kind of the top-down process built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure usually implements one of the productions of the grammar. Thus the created structure of the resulting program closely mirrors that of the grammar it recognizes.

The recursive descent with backtracking is a technique that determines which production to use by trying each production in turn. It is not limited to LL(k) grammars (Left to right, Leftmost derivation grammars), but is not guaranteed to terminate unless the grammar is LL(k). Even when they terminate, parsers that use recursive descent with backtracking may require exponential time.

The top-down parser starts from the root node (start symbol) and matches the input string against the production rules, if matched it replaces them [39].

5.3 Context-free grammars

Context-free grammars are a recursive representation of the context-free languages, which are a larger class of regular languages. These grammars play a substantial role in the compiler technology longer time. They turned the implementation of parsers from an ad-hoc time-consuming implementation task to a routine job that can be performed very quickly.

Context-free grammars are limited in the extent to which they can express all of the requirements of a language. Informally, the reason is that the memory of such a language is limited. The grammar cannot remember the presence of a construct over an arbitrarily long input; this is necessary for a language in which, for example, a name must be declared before it may be referenced. More powerful grammars can express this constraint but they cannot be parsed efficiently. Thus, it is a common strategy to create a relaxed parser to process a context-free grammar that is able to accept a superset of the desired language constructs; later, the unwanted constructs can be filtered out.

5.4 ISES Measureserver unit and its psc control script

The RL functioning has been since the start of ISES RL based on the psc control program and MS. This approach requires creation of the programmable script (executable code running without a compilation) that addresses inputs, outputs and logic of the ISES RL. The script is very condensed and addressing ability rich because it allows programmers to code logical parts, data structures in arbitrary sections in one program (body). It also allows attaching of plugins (drivers) of specific devices indispensable for the RL functioning. The manual design and creation of the psc control program is in principle rather demanding and only a small body of specialists in the field is able to compile and deploy such the script. For this reason, as the alternative approach, the ER-L was recently (2015) introduced to simplify the psc script design, creation and its maintenance by interested non-programmers and teachers. Both the mentioned approaches (manual script creation and ER-L) reach the same results related to the RLs behavior and demanded logical activities.

5.5 Grammar syntax of psc script

The script language grammar is parsed by the Recursive descent parser to build the experiment behavior, data communication and logic. The language developed for the purpose is superficially similar to the C language. Its leading difference compared to the C language is that it is not compiled and functions interpreter-like. This feature is an advantage for the RLs setup and its modification because of avoiding the use of a compiler and its more complex settings. This approach allows better, faster and more effective work.

In this subchapter, the first approach is described only, that is the manual design and creation of the psc control program possessed of two versions. This issue is important to describe as the basis for the work that has been already accomplished concerning the data archiving and diagnostics.

Old psc script version

Users had to write all the script without any syntax errors and save it as a psc file. The creation of the psc script (version 1.0) was very demanding as users had to know all constituent parts of the system and all relationships among its components. It was demanding especially for laymen in the field, who were not experienced in the programming field and for teachers and users. The simple example representing an old version of the psc script is shown in Figure 21.

New psc script version

Since the old script version obstructed the desired dissemination of the RLs to more designers and teachers, the script concept was improved towards the simplified script version. The resulting key improvement was the support for the faster, reliable FSM control and the accessibility of the fast experiments building. The new psc script (version 3.1 and higher) was implemented and encapsulated to the ScriptablePlugin2.ldp plugin.

```
on_each_second {
    if(vardelay >= 1) {
        vardelay = 1;
    }
    else {
        if(vardelay >= 0) {
            vardelay = -1;
        }
        else {
            vardelay = 1;
        }
    }

    if (watchdog_counter > 0.0) {
        watchdog_counter = watchdog_counter - 1;
    }
}

pin_read serie {
    result = ser_tmp;
}

pin_read paralell {
    result = par_tmp;
}

...
```

Figure 21 Old version of the psc script presenting a complicated coding scheme to create data structures and experiment logic

An example of the fast experiment gathering measured data (for example, voltage and current) in the electronic circuit as presented in Figure 22.

```
experiment fast
{
    init
    {
        switch = 1;
    }

    on_sample card_lib
    {
        burst_out_0 = burst_in_0;
        burst_out_1 = burst_in_1;
    }

    finalize
    {
        switch = 0;
    }
}
```

Figure 22 Fast experiment with the "init" and "finalize" blocks switching the relay on/off and the "on_sample" block gathering data provided by two meters attached to the electronic circuit

5.6 Libraries and functionalities of psc script

The MS unit has been designed and developed in the C++ programming language that allows using standard and extended libraries simplifying creation of the application and ensuring its proper operation.

A short introduction of the C++ concept is desirable to have at least the basic overview. The C++ is a multi-paradigm programming language that supports object-oriented programming created by Bjarne Stroustrup in 1983 at Bell Labs. This language is an extension (superset) of C programming and the programs written in the C language can run in the C++ compilers. Generally, a compiler is a computer program that transforms human readable (programming language) source code into another computer language (binary) code.

The C++ is used to create general systems software, drivers for various computer devices, back end servers and specific applications, and also widely used in the creation of video games and simulators.

As mentioned, the C++ supports object-oriented programming. There are defined four major principles of the object-oriented development, namely the inheritance, abstraction, encapsulation and polymorphism.

This universal language is not purely the object-oriented language because object-oriented means to works with objects and classes, but its source code can be written without classes, just using the structured programming.

The data and functions (procedures to manipulate the data) are bundled together as a self-contained unit called an object. A class is an extended concept similar to the structure in the C programming language. This class describes both the properties (data) and behaviors (functions) of objects.

The C++ provides programmers the standard libraries, which were created after many years. They are divided in the following important parts [40]:

1. The C++ core language provides all the building blocks including data types, variables, literals and other.
2. The C++ Standard Library has a rich set of methods for manipulating input/output files and character strings.
3. The Standard Template Library (STL) provides a rich set of template classes to manipulate with data structures.

4. The Microsoft Foundation Classes (MFC) is used for creating Windows applications. It provides an application programming interface (API) to implement all the features expected for the development.

The second set of libraries and functions is intended for own functioning of the MS to start up and run a given RL. This specific set is encapsulated in the ScriptablePlugin2.ldp plugin that is attached to the MS during its startup. As mentioned, this plugin is the strongest and the most difficult by reason of its wide use. These libraries are provided by the Windows operating system called as the Dynamic-link libraries (also known as DLLs) to support implemented expressions, control and mathematical functions coded in the psc script.

Expressions

The expression is a term that can contain any valid combination of explicit numerical constants, variables, functions and operations coded in the prescribed syntactical form of programming language.

Control functions

The control functions are designed to manipulate with input/output data and experiment states/blocks as follows in the example list:

- *write_output* - It generates a new output sample based on the current state of the output variables. This function is active only in the block *init*, *on_sample* or *on_manual_sample* inside the fast experiment.
- *sample (expression)* - It loads a sample value with its index (*sample_index* with the expression) into the input variables. This function is active in the block *on_sample* or *on_manual_sample* inside the fast experiment.
- *rotate (start, end, amount)* - It rotates the temporary variables *t0* - *t99* in the range of the *amount* steps to modify positions.

Mathematical functions

Many functions are available to implement in the psc script to simplify and speed up the RL design and development as listed below:

- *sin(x)* - It returns the sine value in radians,
- *cos(x)* - It returns the cosine value in radians,
- *asin(x)* - It returns the arc sine value in radians in the range *<-1, 1>*,

- *acos(x)* - It returns the arc cosine value in radians in the range *<-1, 1>*,
- *atan2(x, y)* - It returns the arc tangent in radians from *X/Y*,
- *pow(x, y)* - It calculates the expression x^{y} (power),
- *floor(x)* - It rounds the value to the nearest lower integer number,
- *ceil(x)* - It rounds the value to the nearest higher integer number,
- *round(x)* - It rounds the value to the nearest integer number,
- *fraction(x)* - It returns the decimal part of the input value,
- *abs(x)* - It returns the absolute value of the parameter,
- *min(x, minimum)* - It returns the minimum of the two values,
- *max(x, maximum)* - It returns the maximum of two values,
- *remap(x, s0, s1, s2, d0, d1, d2)* - It remaps input values from the defined range *<s0, s1, s2>* to the range *<d0, d1, d2>* with the fact that its parameter *s1* is remapped to *d1*. Input values, which are out of the range, are always clipped to the range *<s0, s2>*,
- *clamp(x, minimum, maximum)* - It returns the value in the demanded range. If defined *minimum > maximum,* the minimum value is returned,
- *cmp(x, treshold, result_a, result_b)* - It returns the parameter *result_a* if defined *x > threshold,* otherwise returns *result_b,*
- *cmp(x, treshold)* - It is the short form for *cmp(x, threshold, 1.0, 0.0),*
- *cmps(x, treshold, result_a, result_b)* - It returns the parameter *result_a* if defined *x >= threshold,* otherwise returns *result_b,*
- *cmps(x, treshold)* - It is the short form for *cmps(x, threshold, 1.0, 0.0).*

5.7 Analysis of Measureserver functioning and data traffic

The MS unit is a Windows application designed and built as a server that is responsible for processing and transferring of measured data and metadata between the rig's physical modules and connected clients.

As introduced, the MS was implemented in the C++ programming language by using the object-oriented programming. Just to remind this specific issue, the object-oriented programming is a paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and the code, in the form of procedures, often known as methods. A feature of objects is that the object's procedures can access and often modify the data fields of the object with which they are associated. The MS was built on the class-based concept that is the most often used object-oriented programming alternative. It means, the objects are instances of classes, which typically determine their type. The MS architecture and relationships among particular implemented objects is shown in Figure 23 that presents the base object as the MFC CObject with all derived objects up to the lowest levels where are located the applicable objects for dialog boxes, communications, processes and parsers.

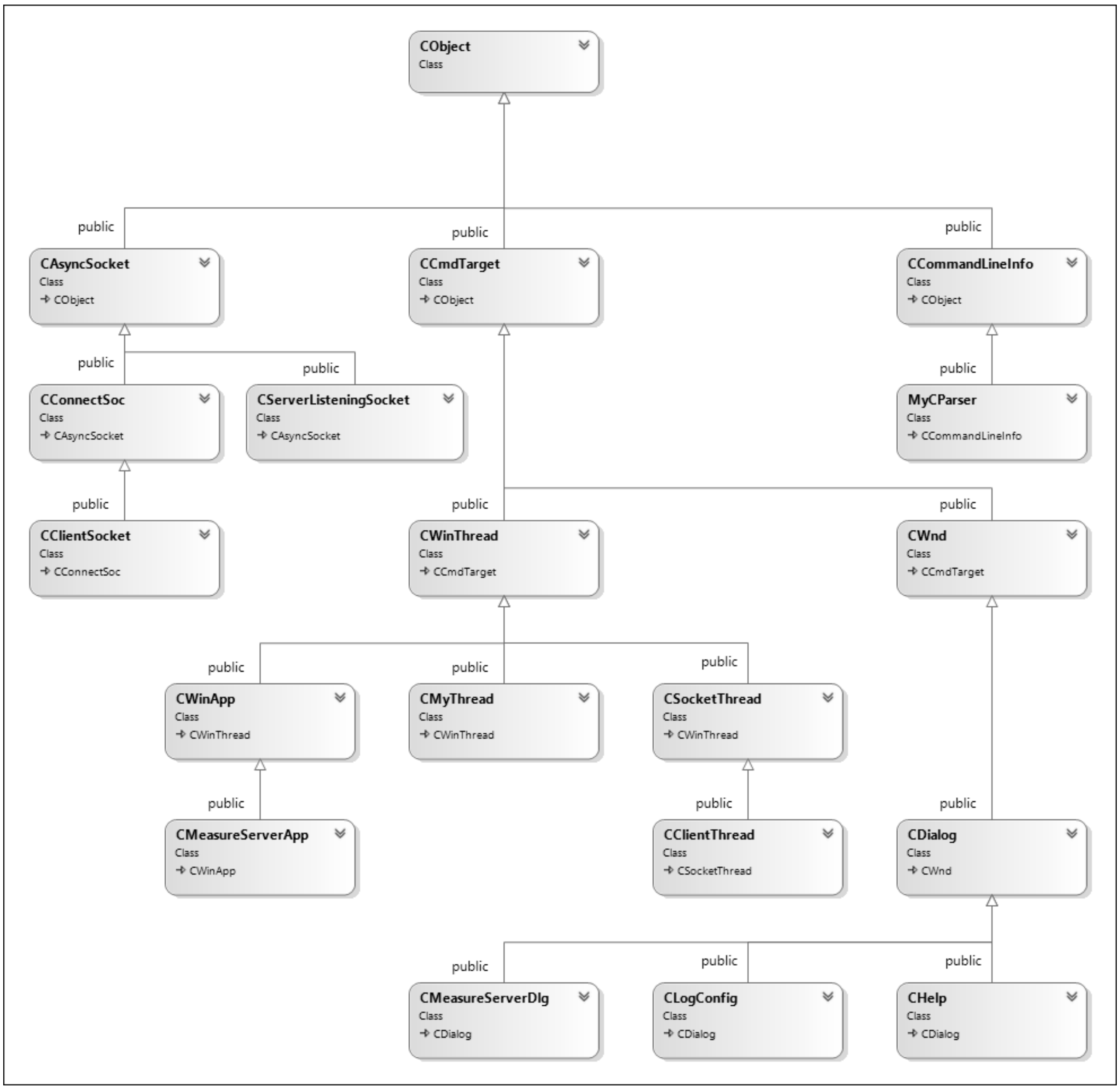

CObject Class
CAsyncSocket Class → CObject
CCmdTarget Class → CObject
CCommandLineInfo Class → CObject
CConnectSoc Class → CAsyncSocket
CServerListeningSocket Class → CAsyncSocket
MyCParser Class → CCommandLineInfo
CClientSocket Class → CConnectSoc
CWinThread Class → CCmdTarget
CWnd Class → CCmdTarget
CWinApp Class → CWinThread
CMyThread Class → CWinThread
CSocketThread Class → CWinThread
CMeasureServerApp Class → CWinApp
CClientThread Class → CSocketThread
CDialog Class → CWnd
CMeasureServerDlg Class → CDialog
CLogConfig Class → CDialog
CHelp Class → CDialog
public

Figure 23 Class diagram of the MS concept consisted of the objects creating its basic functionality; the base object MFC CObject, the derived objects provide main features and the lowest levels bring the dialog boxes, communications, processes and parsers

Before the MS starts working, the unit has to initialize several components ensuring its proper functioning. The components are loaded as the files from various directories and processed by respective parsers. This initialization phase takes place in the following prescribed sequence:

1. The MS loads and parses its configuration file including a reference that is used to attach and start the control plugin in the unit.
2. The MS then configures and initializes the GUI module by the parsed values to set the unit options and its functioning.
3. The control plugin loads the psc file by a reference that is placed in the configuration file and it then parses the coded script.
4. The control plugin then loads and initializes device plugins referenced in the parsed psc file to use them in the experiment.
5. The MS sets the attributes of parsed pins defined in the psc file and creates the pins list that is available in the unit.
6. If required, the MS performs initial calibrations of attached devices based on instructions coded in the psc file.

The next subchapters deal with the MS functioning related to the components initialization and control program operation including data traffic.

5.7.1 Components initialization

The MS must load and initialize all the integral and referenced components before the operation. There are two important components, Measureserver.cfg and ScriptablePlugin2.ldp, which are successively initialized as the first. Further components are initialized optionally according to the experiment scheme.

As the first, the MS loads the Measureserver.cfg file from a disk (that is, it opens the file and reads data). It then parses the file content to obtain predefined parameters and their values the unit requires for the initial setup of its internal modules (data logging and graphical user interface). The parser finally provides the reference (path and filename) to two inevitable files. The first referenced file is the ScriptablePlugin2.ldp plugin used for all the experiments at present. The second reference is for the psc script that is parsed and started later.

The Measureserver.cfg file (presented in Figure 9) is processed by the LR(1) parser (introduced in subchapter 5.2) that is a part of the MS core. Before the description of the LR parsing algorithm, it is important to clarify the constructs, parse trees and bottom-up parsing, related to this kind of the parsing.

Parse trees

The parse trees are a tree-representation of the derivations that is occurred in context-free grammars. Parse trees clearly show how the symbols of a terminal string are grouped into substrings, each of which belongs to the language.

Bottom-up parsing

The Bottom-up parsing is a technique in which the abstract-syntax tree is in the bottom up fashion. The bottom-up name comes from the concept of a parse tree. It discovers and processes the tree starting from the bottom left end and incrementally works its way upwards and rightwards. The tree may be merely implicit in the parser's actions. The opposite technique is the top-down parsing, in which the input's overall structure is decided first before moving down the abstract-syntax tree, leaving the lowest level small details to the last.

LR parsing algorithm

The parsing algorithm is schematically depicted in Figure 24. It typically consists of the five items, 1) input, 2) output, 3) data stack, 4) driver program and 5) parsing table divided into the ACTION and GOTO parts.

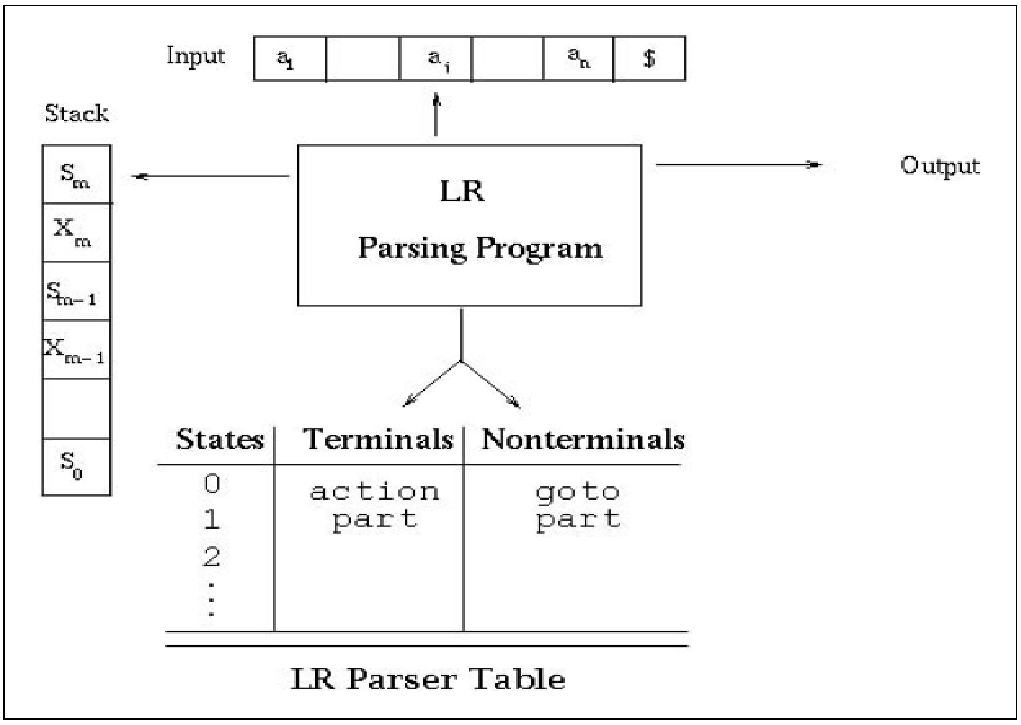


Figure 24 Scheme of the LR parsing algorithm to process input string

This used algorithm involves the below actions:

Bottom-up parsing action

The bottom-up parsing corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top). This method is called as the shift-reduce parsing.

Reduction action

This process is the "reducing" a string *w* to the start symbol of the grammar. At each reduction step, a specific substring matching the body of a production is replaced by the non-terminal at the head of that production. A reduction is the reverse of a step in a derivation.

Handle action

The bottom-up parsing within a left-to-right scan of the input constructs a right most derivation in reverse. The "handle" is a substring that matches the body of a production, and whose reduction represents one step along the reverse of a rightmost derivation, that is, when there is string *w* of a grammar.

Shift-reduce action

The action is a form of the bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the rest of the string to be parsed. The handle appears at the top of the stack just before it is identified as the handle. The *$* is used to mark the bottom of the stack and also the right end of the input. Initially, the stack is empty, and the string *w* is on the input with a stack as *$* and string *w* as *w$*. During a left-to-right scan of the input string, the parser shifts zero or more input symbols onto the stack, until it is ready to reduce a string β of grammar symbols on top of the stack. It then reduces β to the head of the appropriate production. The parser repeats this cycle until it has detected an error or until the stack contains the start symbol and the input is empty.

LR Items

An item set is the list of production rules. An item set has a one-to-one correspondence to a parser state, while the items within the set, together with the next symbol, are used to decide which state transitions and parser action are to be applied. The general form of a LR(1) item is $[A \rightarrow \alpha \bullet \beta, a]$, where $A \rightarrow \alpha\beta$ is a production and *a* is terminal or right *$* end marker. The dot represents how an item is seen in a given state. The second defined component is called the lookahead of the item. The involved lookahead has no effect in an item of the form *[A→α • β, a]* where *β* is not *ε*, but an item of the form *[A→α •, a]* calls for a reduction by *A→α* only if the next symbol is *a*.

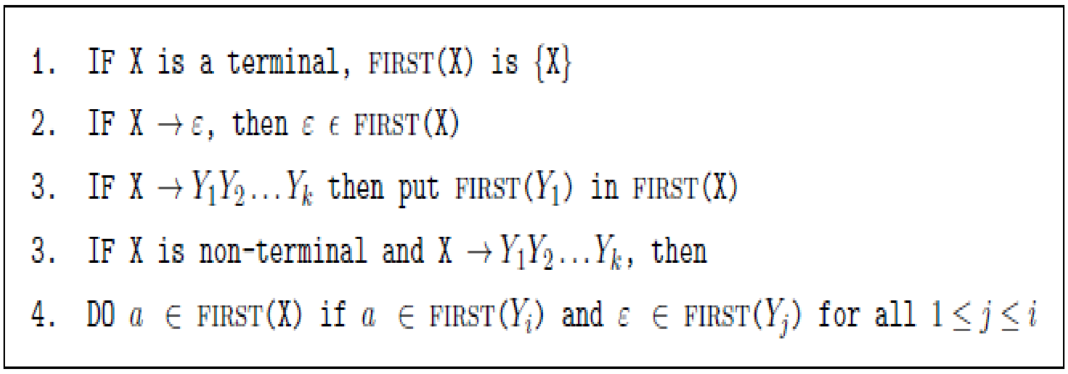
- The • indicates how much of an item is seen at a given state,
- The expression *[A → • XY Z]* indicates the parser is looking for a string that can be derived from *XYZ*,
- The expression *[A → XY • Z]* indicates the parser has seen a string derived from *XY* and is looking for one derivable from *Z*.
- There is a canonical set of the LR(1) items having a set of items:
- derivable from *[S'→ • S, $]* and,
- that can derive a final configuration.

FIRST statement

For a string of grammar symbols *α* is defined the *FIRST(α)* as:

- The set of terminal symbols beginning strings derived from *α*,
- If *L*⇒*ε*, then in valid the expression *ε* ∈ *FIRST(α)*.

The *FIRST(α)* contains the set of tokens valid in the initial position in *α*. The algorithm presenting the building the *FIRST(X)* is listed in Figure 25.



```
1. IF X is a terminal, FIRST(X) is {X}
2. IF X $\rightarrow \varepsilon$, then $\varepsilon \in$ FIRST(X)
3. IF X $\rightarrow Y_1Y_2\ldots Y_k$ then put FIRST($Y_1$) in FIRST(X)
3. IF X is non-terminal and X $\rightarrow Y_1Y_2\ldots Y_k$, then
4. DO $a \in$ FIRST(X) if $a \in$ FIRST($Y_i$) and $\varepsilon \in$ FIRST($Y_j$) for all $1 \leq j \leq i$
```

Figure 25 Example of the algorithm to build the FIRST(X) statement

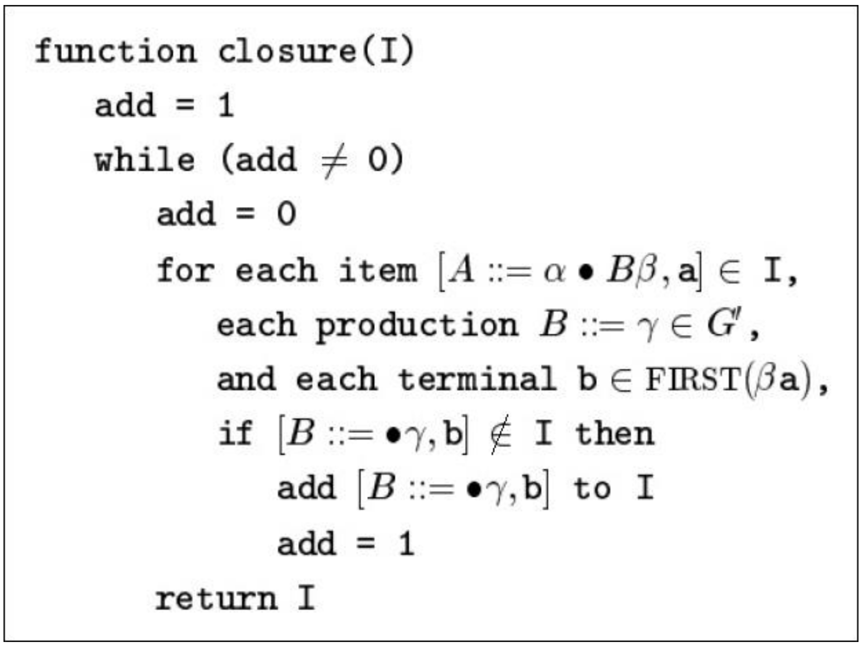
CLOSURE statement

The closure is an operation to construct the set of items through rules:

1. Initially add every item in *I* to closure (I),

2. If the expression $A \to \alpha \bullet B\beta$ is in closure (I) and $B \to \gamma$ is a production, then add $B \to \bullet\, \gamma$ to I if not there,
3. These rules are applied until no more items can be added.

The algorithm computing this statement is clearly explained in Figure 26.



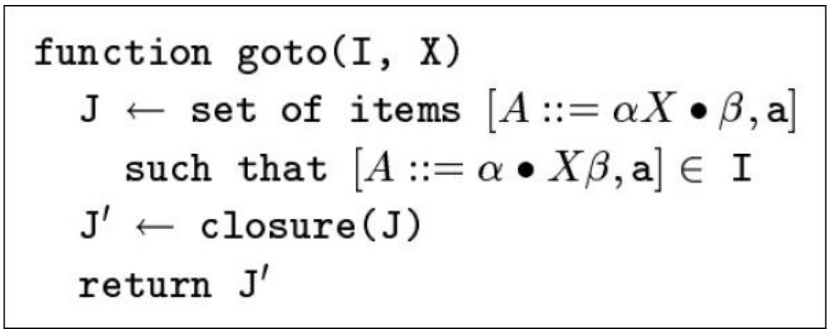
```
function closure(I)
   add = 1
   while (add ≠ 0)
      add = 0
      for each item $[A ::= \alpha \bullet B\beta, \mathrm{a}] \in$ I,
         each production $B ::= \gamma \in G'$,
         and each terminal $\mathrm{b} \in \mathrm{FIRST}(\beta\mathrm{a})$,
         if $[B ::= \bullet\gamma, \mathrm{b}] \notin$ I then
            add $[B ::= \bullet\gamma, \mathrm{b}]$ to I
            add = 1
      return I
```

Figure 26 Algorithm to implement the CLOSURE operation

GOTO statement

Let I be a set of LR(1) items and X be a grammar symbol. Then, the expression *GOTO(I, X)* is the closure of the set of all items *[A → αX • β, a]* defined for the operation such that *[A → αX • β, a]* ∈ *I.*

If I is the set of valid items for some viable prefix γ, then *GOTO(I, X)* is the set of valid items used for the viable prefix γX. The *GOTO(I, X)* represents the state after recognizing X in the state I as explained in Figure 27.



```
function goto(I, X)
   J ← set of items $[A ::= \alpha X \bullet \beta, \mathrm{a}]$
      such that $[A ::= \alpha \bullet X\beta, \mathrm{a}] \in$ I
   J′ ← closure(J)
   return J′
```

Figure 27 Algorithm to implement the GOTO operation

FOLLOW statement

For a non-terminal *A* the *FOLLOW(A)* is defined as:

- The set of used terminals, which appear immediately to the right of *A* in some sentential form.
- A terminal symbol has no *FOLLOW* set.

The method of building this set is described in two points:

1. Place the symbol *$* in the *FOLLOW(S)*, where *S* is the start symbol and *$* is the input right-end-marker.
2. If there is the production $A \rightarrow \alpha B\beta$, then everything in *FIRST(β)* except for ε is placed in *FOLLOW(B)*.

Augmented Grammar

The collection construction of the sets of LR(1) items is started with the defined item *[S' → • S, $]*, where:

1. *S'* is the start symbol of the augmented grammar *G'*,
2. *S* is the start symbol of *G*, and *$* is the right end of string marker.

To compute the collection of the sets of LR(1) items, first a given grammar is augmented, them a respective algorithm is implemented.

Parsing tables

The LR(1) parser uses the ACTION-GOTO table served as a reference to the parser at every move. This table basically represents a FSM that instructs the parser to perform a shift or reduce action when it undergoes a transition from one state to another. The notations used here are as follows:

- *$* represents the end of an input string,
- *sn* tells the parser to make a shift to state I_n ,
- *rk* tells the parser to reduce by rule *k* described in the grammar,
- *acc* indicates that the input string is accepted,
- *j* represents *GOTO(j)* in the table.

With these definitions, there is possible to describe the following method for constructing the LR(1) ACTION-GOTO table [41]:

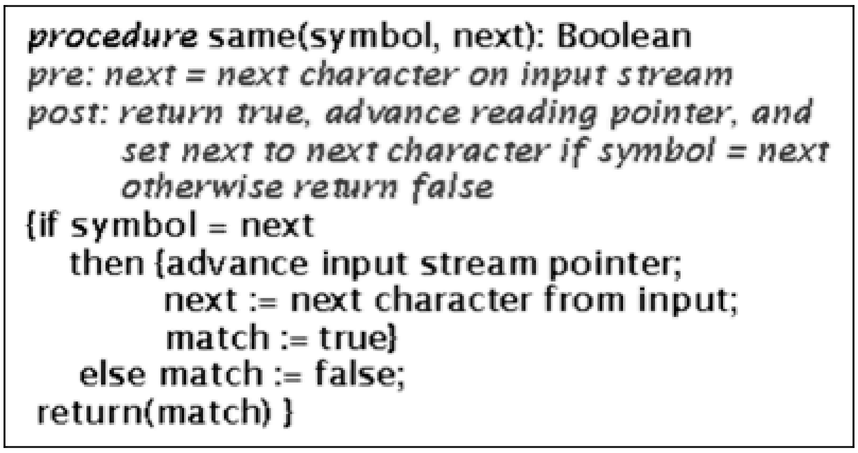
1. Construct the collection of the sets of LR(1) items for G',
2. State i of the parser is constructed from I_i ,
    a) If $[A \rightarrow \alpha \bullet a\beta, b] \in I_i$, and $GOTO(I_i , A) = I_j$, then set $ACTION[i, a]$ to *shift j* or *sj* (*a* must be a terminal),
    b) If $[A \rightarrow \alpha \bullet, a] \in I_i$, then set $ACTION [i, a]$ to do *reduce* $A \rightarrow \alpha$ or *rk* (*k* represents the k^{th} rule of the context-free grammar),
    c) If $[S' \rightarrow S \bullet, \$] \in I_i$, then set $ACTION[i, \$]$ to *acc*,
3. If $GOTO(I_i , A) = I_j$, the set $GOTO[i, a]$ in the table to *j*,
4. All other entries in the ACTION-GOTO table are set to “error”,
5. I_0, the initial state of the parser, is the closure of the item $[S' \rightarrow \bullet S, \$]$. All other states are formed by the above procedure of constructing a FSM to represent the parsing action on each item in I_0 recursively.

After the Measureserver.cfg file parsing, the ScriptablePlugin2.ldp plugin is loaded, attached (its functions) and finally started to parse the psc file to build the experiment logic and data structures. This plugin is also responsible for the communication with other attached plugins, which can be optionally defined as the file references in the psc file. Furthermore, it concerns with the experiment operation related to the control, observation and measurement.

The ScriptablePlugin2.ldp plugin uses the Recursive descent parser residing in its core (introduced in subchapter 5.2) to perform parsing of the control program saved in the pcs file to build the experiment logic.

Recursive descent parser algorithm

The recursive descent parsing is one of the most straightforward forms of parsing technique. This is a top-down process in which the parser attempts to verify that the syntax of the input stream is correct as it is read from left to right. A basic operation necessary for this involves reading characters from the input stream and matching then with terminals from the grammar that describes the syntax of the input. The parsers looks ahead one character and advances the input stream reading pointer when proper matches occur. The procedure that accomplishes the matching and reading process is shown in Figure 28.



```
procedure same(symbol, next): Boolean
pre: next = next character on input stream
post: return true, advance reading pointer, and
        set next to next character if symbol = next
        otherwise return false
{if symbol = next
   then {advance input stream pointer;
           next := next character from input;
           match := true}
    else match := false;
 return(match) }
```

Figure 28 Procedure to match the input symbol

The variable called “next” looks ahead and always provides the next character that will be read from the input stream. This feature is essential for the parsers to be able to predict what is due to arrive as input. There is also defined an error indicator that is returned by the procedure.

What the Recursive descent parser actually does is to perform a depth-first search of the derivation tree for the string being parsed. This principle provides the “descent” portion of the name. The “recursive” portion comes from the parser's form, a collection of recursive procedures.

The following example presents a first portion of the parser’s features. There is defined the simple input grammar:

$E \rightarrow x+T$

$T \rightarrow (E)$

$T \rightarrow x$

and the derivation tree for the expression $x + (x + x)$ as shown in Figure 29.

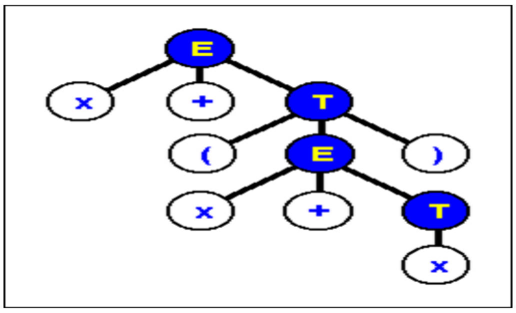
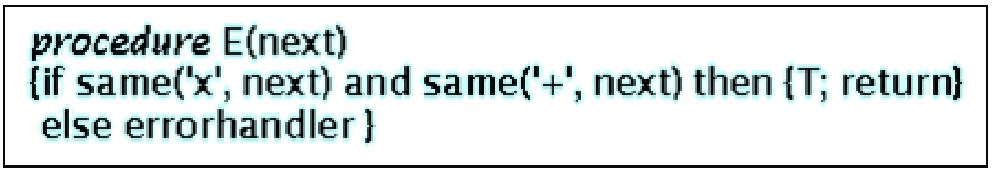


Figure 29 Procedure to accomplish the derivation tree for x + (x + x)

The parser traverses the tree by first calling a respective procedure to recognize an *E*. The procedure reads an *x* and a +, and then it calls a procedure to recognize a *T*. This looks like described clearly in Figure 30.

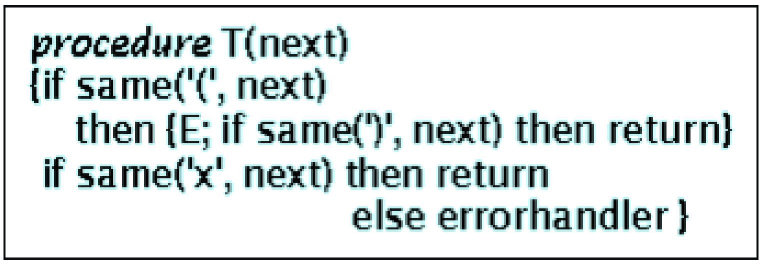


```
procedure E(next)
{if same('x', next) and same('+', next) then {T; return}
 else errorhandler }
```

Figure 30 Procedure to traverse the expression tree

The *errorhandler* is a procedure notifying the user that a syntax error has been made and then possibly terminates execution.

In order to recognize a *T*, the parser must resolve which of the productions to execute. This is not difficult and is done in the procedure in Figure 31.



```
procedure T(next)
{if same('(', next)
    then {E; if same(')', next) then return}
 if same('x', next) then return
                        else errorhandler }
```

Figure 31 Procedure to execute the production

In the above procedure, this parser is able to determine whether *T* has the form *(E)* or *x*. If not, then the error return is called, otherwise the appropriate terminals and non-terminals are recognized.

The parser works with deterministic context-free grammars. They are the subset of context-free grammars, which can be derived from the deterministic pushdown automata (DPDA), and they generate the deterministic context-free languages. The DPDA is a variation of the pushdown automaton that employs a stack. It accepts the deterministic context-free languages.

As an example of the grammar for conditional statements is presented where the symbol *S* is a non-terminal generating statements and *B* is one that generates the following Boolean (true or false) expressions:

$S \rightarrow if\ B\ then\ S;$

$S \rightarrow if\ B\ then\ S\ else\ S;$

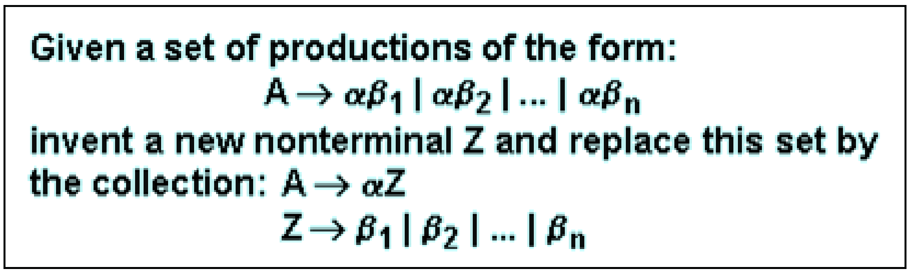
Both of these productions begin the same way, so it is not clear from the first symbols exactly, which production is to be used in recognizing a conditional statement. The solution to this problem is called factoring. The principle of factoring is, the common prefix is recognized (in this case *if B then S*) and then it decides which suffix to recognize as follows:

$S \rightarrow if\ B\ then\ S\ Z$

$Z \rightarrow ;$

$Z \rightarrow else\ S;$

The general factoring algorithm is presented in Figure 32 with α as the common prefix and β_i as the suffixes mentioned above.



Given a set of productions of the form:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n$$

invent a new nonterminal Z and replace this set by the collection: $A \rightarrow \alpha Z$

$$Z \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

Figure 32 General algorithm of the factoring process

The next example, a grammar that generates strings of the form $a^n b^n$ is defined to describe other features of the parser:

$S \rightarrow aSb$

$S \rightarrow ab$

where factoring brings the expressions:

$S \rightarrow aZ$

$Z \rightarrow Sb$

$Z \rightarrow b$

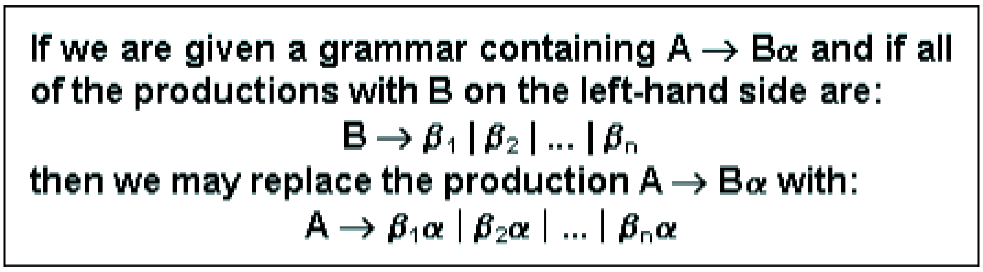
which is the unclear form when applying the *Z*-productions. Substituting for the symbol *S* solves this problem and constructs the required productions:

$S \rightarrow aZ$

$Z \rightarrow aZb$

$Z \rightarrow b$

The general algorithm for the substitution is described in Figure 33. Capital Roman letters are non-terminals and Greek letters represent (possibly empty) the strings of terminals and non-terminals to process.



If we are given a grammar containing $A \to B\alpha$ and if all of the productions with B on the left-hand side are:

$$B \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

then we may replace the production $A \to B\alpha$ with:

$$A \to \beta_1\alpha \mid \beta_2\alpha \mid \dots \mid \beta_n\alpha$$

Figure 33 General algorithm of the substituting process

It simply means if this substitution is performed for a non-terminal in the production, then it must be substituted using all of the instances where it appears on the left-hand side of the production.

Finally, to mention for short, here are three cases, which should not occur in nice grammars along with algorithms for their removal as follows [42]:

1. A chain rule (or singleton production) is one of the form $A \to B$ where both A and B are non-terminals.
2. A non-terminal symbol unreached is one that cannot be generated from the starting symbol.
3. A useless non-terminal is one that generates no strings of terminals.

When the Measureserver.cfg file and the psc script are successfully parsed by respective parsers, the ScriptablePlugin2.ldp plugin initializes and starts all the components for the purpose of the control program operation.

5.7.2 Control program operation

The control program coded in the psc script is a component part that is essential to the MS functioning. Typical RLs are designed to monitor attached meters or probes and to control devices with the aim of obtaining measured data detailing examined phenomena in charts and tables for the analysis.

Basic psc script structure

The standardized structure of a new version of the psc script file consists of six basic obligatory and optional terms (sections) as listed below:

1. Header (required),

2. Definitions of imports (required),
3. Definitions of variables (optional),
4. Definitions of functions (optional),
5. Definitions of state/states (optional),
6. Definition of fast experiment (optional).

Header

The header is declared in the beginning of each psc script and its syntax is obligatory. On the first line, there is specified a version of the psc script. The three versions are available up to now, as 3.1, 3.2 and 3.3.

Under the script version, there is placed the experiment name and the file reference to an attached plugin, usually used the PCI1202CardPlugin.ldp or ISESBlueUsbPlugin.ldp according to the ISES control board type.

Definitions of imports

This is the second part of the psc script used to define inputs and outputs, which the user wants to work with. The ISES control board with the AD/DA interface card has analog and digital inputs and outputs.

Definitions of variables

In this part of the psc script, a user defines global variables, which are valid for the entire script. All variables are declared as a double (real number).

Definitions of functions

A new version of the psc script (version 3.1 and higher) allows the user to define own functions. This is a great simplification because it is not necessary to copy parts of the code that is already repeated. These parts of the code can be neatly encapsulated into the defined function.

Definitions of states

The psc script can be declared as a finite-state machine. The system can change from one state to another. This new approach greatly simplifies the design of the RLs that can be constructed as the sequential tasks.

Definition of fast experiment

The fast experiment feature was implemented in a new version of the psc script to measure/monitor fast events, like the transient phenomena in electronic circuits. The MS is capable to read generated data from respective pins in frequency up to 3 kHz during maximally 10s per one measurement.

Finite-state machine operation

As mentioned, the MS can work as the FSM (introduced in subchapter 5.1) coded in the psc script by using several specifiers and commands.

A new state can be defined by the keyword *state* that is followed by the implementation inside curly brackets. Every state contains several handlers providing activities created for respective purposes. The handler *entry* and *exit* are mandatory only, the rest of them are optional. There are eight standard handlers built to the language with different properties as listed:

1. entry,
2. step,
3. timeout,
4. pin_read,
5. pin_write,
6. on_each_change,
7. on_user_second,
8. exit.

Every handler can contain cycles, conditions, variables and functions. The scheme of all the available handlers is presented in Figure 34.

These handlers are described at some length in the following bullets to understand the concept of the implemented state machine [43]:

- *entry* - If this handler is defined, it should be placed as the first in the sequence. The code written here is performed only once when the RL enters this state. It does not allow the context switching that is an event where the RL changes from a one state to another. This switching is performed by using the keyword *current_state* = *new_state*.

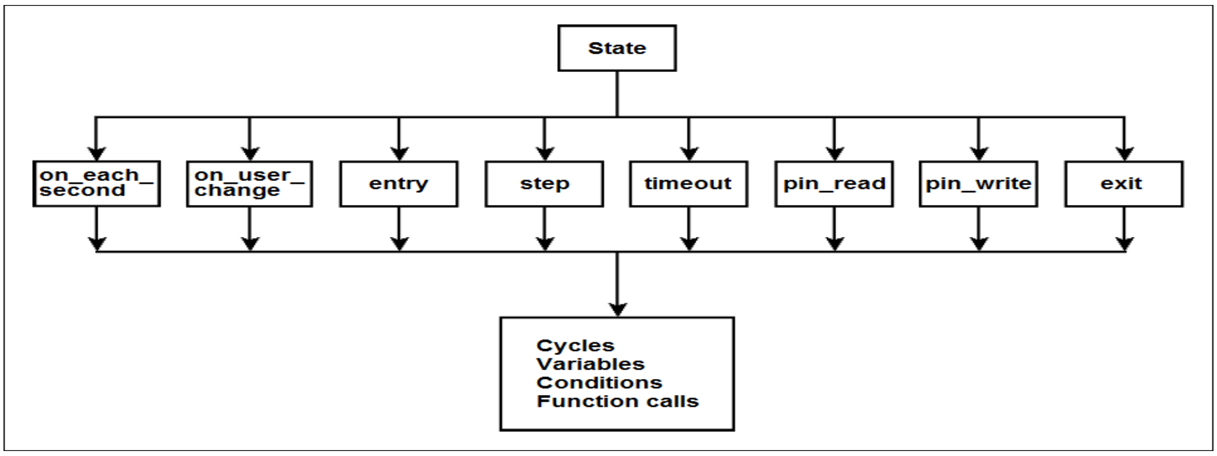
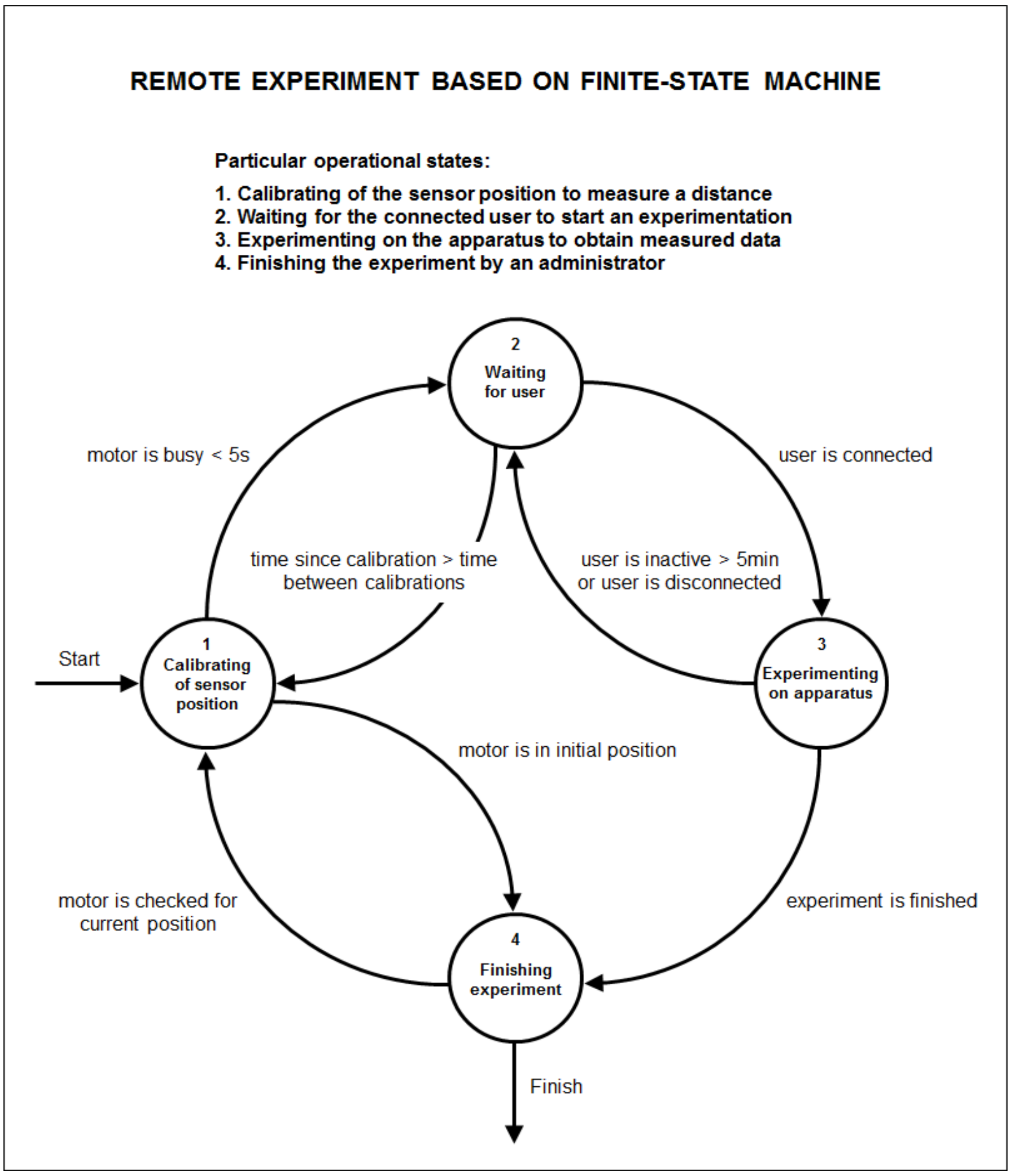


Figure 34 Scheme of the handlers used for the finite-state machine operation

- *step* - This handler is used for a code that is executed repeatedly. The step frequency must be specified in hertz to run the code.
- *timeout* - This handler is used for a code executed after the elapsed time given in seconds A one state allows containing more than one timeout. The handler supports the context switching.
- *pin_read* - This handler provides the function allowing the user to read data from the ISES control board pins. These data can be displayed on the web page in a relevant JavaScript widget. The handler contains its special variable *result* used for reading by this widget. The frequency of handler calling is directly adjusted by the reading widget.
- *pin_write* - This handler is very similar to *pin_read* but it provides a function to write data to the ISES control board pins. A user can control the RL and change its properties through the web page. The handler contains the variable *new_value* to write the data to the pins.
- *on_user_change* - In this handler, the code is executed when the client of the experiment is changed. It is correct to define this handler at the end of each state. If it is defined and if a programmer makes a mistake the RL performs the required code and does not fall in a wrong state.
- *on_each_second* - In this handler, the implemented code is executed each second. This behavior can be achieved by the *step* handler.
- *exit* - The handler executes its code at the end of the state and only once. It does not support the context switching.

The scheme depicted in Figure 35 describes the FSM-based concept used to initialize and run the experiment. Each state is responsible for a specific action to complete (including data exchange, input/output operations, etc.) and to decide a transition (based on conditions) to the next state.



REMOTE EXPERIMENT BASED ON FINITE-STATE MACHINE
Particular operational states:
1. Calibrating of the sensor position to measure a distance
2. Waiting for the connected user to start an experimentation
3. Experimenting on the apparatus to obtain measured data
4. Finishing the experiment by an administrator
2 Waiting for user
motor is busy < 5s
user is connected
time since calibration > time between calibrations
user is inactive > 5min or user is disconnected
Start
1 Calibrating of sensor position
3 Experimenting on apparatus
motor is in initial position
motor is checked for current position
experiment is finished
4 Finishing experiment
Finish

Figure 35 Example of the FSM-based concept used to calibrate the sensor position, to wait for the connected user and to start the experiment operation with its optional finish

The RL designer is allowed to create as many FSM states as required for a completion of the experiment logic. The example shown in Figure 36 presents the coded definitions of particular states (based on the scheme in Figure 35) with using the described specifiers and commands.

```
init
{
     initialization();
     current_state = calibrating;
}

state calibrating;
 {
     step
     {
          if (calibrating_event == true)
          {
               if (motor_busy < 0.5)
               {
                    current_state = waiting;
               }
          }
          else
          {
               motor_position = 0;
               current_state = finishing;
          }
     }
}

state waiting;
{
     on_each_second
     {
          if (waiting < 0.5)
          {
               current_state = experimenting;
          }
          else
          {
               if (time_since_calibration > time_between_calibrations)
               {
                    current_state = calibrating;
               }
          }
     }
}

state finishing;
{
     step
     {
          if (motor_position > 0)
          {
               calibrating_event = false;
               current_state = calibrating;
          }
          else
          {
               finish();
          {
     }
}

state experimenting
{
     step
     {
          experiment();
     }

     on_each_second
     {
          if (waiting > 0.5)
          {
               current_state = waiting;
          }
     }
}
```

Figure 36 Example of the psc script in which is implemented the FSM-based experiment; the specifiers "state" are states, the variable "current_state" makes a transition to a new associated state during the operation

The MS is able to receive or send data in any operational state in which the experiment resides. The communication between the MS and all connected clients (queueing based on their access time, activities or status) is realized by the WebSocket protocol (introduced in subchapter 3.2).

The WebSockets provide a persistent connection between the MS and a client that both involved parties can use to start sending data at any time.

The client establishes a WebSocket connection through a process known as the WebSocket handshake. This process starts with the client sending a regular HTTP request to the MS. The Upgrade header is included in this request that informs the MS that the client wishes to establish a connection.

The MS agrees to the upgrade and communicates this through the Upgrade header in the response as the first necessary step.

Now, the handshake is complete the initial HTTP connection is replaced by a WebSocket connection that uses the same underlying TCP/IP connection. At this point, either party can start sending data to the remote place.

The data transfer goes without incurring the overhead associated with the traditional HTTP requests. The data are transferred through a WebSocket as messages, each of which consists of one or more frames containing the sending data (the payload). In order to ensure the message can be properly reconstructed when it reaches the client each frame is prefixed with 4-12 bytes of data about the payload. Using this frame-based messaging system, it helps to reduce the amount of non-payload data transferred. This approach leads to significant reductions in latency during the communication process.

6. MAIN RESULTS OF THE WORK

In this chapter are presented the results of the envisaged goals of the doctoral thesis in the Chapter 4 using the methods summarized in Chapter 5. This chapter is constituted by the extensive programming work of the internal grant project IGA of the group of Ph.D. students at Tomas Bata University in Zlín and the partial fulfillment of the project SCOPES of the Swiss National Science Foundation and Swiss Agency for Developments and Cooperation.

For the purpose of introducing into results, there is presented the general scheme of the MS unit shown in Figure 37 with new added modules, which are denoted by the red frame. These modules are called DAM, MUD, PMD and EPS described at some length in the next subchapters.

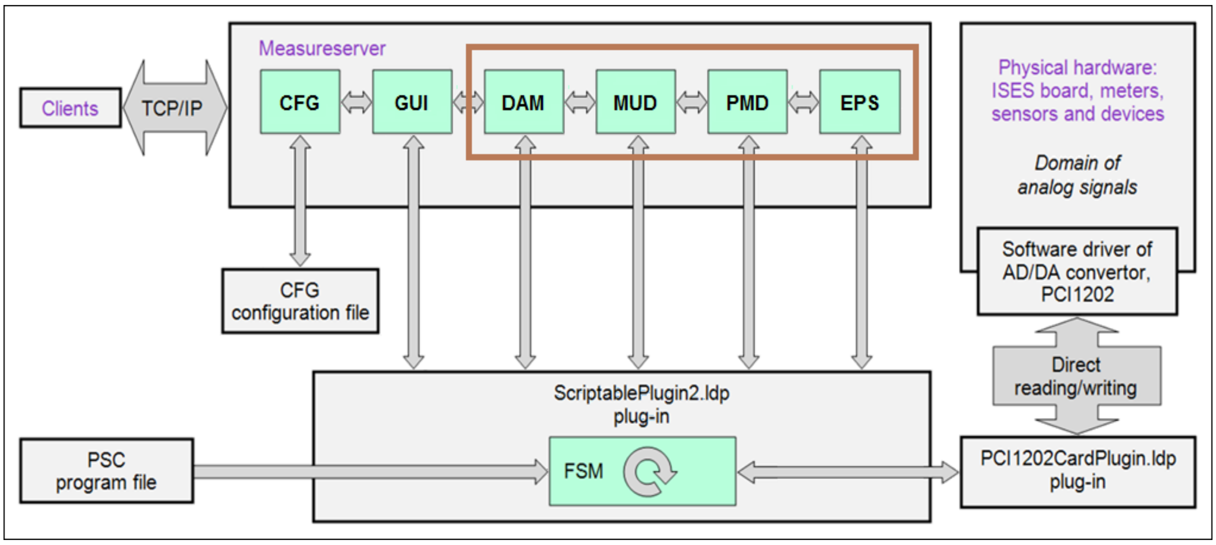

Clients
TCP/IP
Measureserver
CFG
GUI
DAM
MUD
PMD
EPS
Physical hardware: ISES board, meters, sensors and devices
Domain of analog signals
Software driver of AD/DA convertor, PCI1202
CFG configuration file
Direct reading/writing
ScriptablePlugin2.ldp plug-in
FSM
PSC program file
PCI1202CardPlugin.ldp plug-in

Figure 37 Scheme of the MS functional concept that includes the new modules located in the red frame; 1) DAM is data archiving management, 2) MUD is Measureserver unit diagnostics, 3) PMD is physical modules diagnostics, and 4) EPS is embedded real-world phenomena simulation

6.1 Data archiving and logging module

The first task of this doctoral thesis was to design and implement the module called DAM (data archiving management) responsible for gathering, filtering and saving measured data and experiment metadata to an xml file. The module also allows an option to log all performed activities by the connected clients during their experimentations to a log file. The reason to implement this module was to have complete information about the RLs operation and clients whose interventions (their behavior) affect the experimental process.

The generated xml and log files are being sent to the data warehouse where they are analyzed in detail and distributed to other subsystems. The xml files are provided to clients who require measured data coming from typical cases without the real experimenting to reduce time to obtain the measured data.

The log files are useful for RL designers interested in the didactical aspects concerning the clients' behavior for the purpose of RLs improvements.

As the first, more significant part, it deals with the measured data archiving to the XML (eXtensible Markup Language) data format. The XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals emphasize the simplicity, generality, and usability across the Internet. It is a textual data format with a strong support via Unicode (standard for the text encoding, representation and handling) for different human languages. The design focuses on documents, but the language is widely used for working with the data structures such as those used in web services [44], hence it is the optimal data format.

For the DAM implementation, the Microsoft XML Core Services (MSXML) was exploited to speed up its development. The MSXML provides a set of the services allowing developers to design and build high-performance XML based applications. It provides a high level of interoperability with other applications, which adhere to the XML 1.0 standard in realized projects.

The MSXML APIs are represented by the COM (Component Object Model) objects used programmatically in source codes. Developers can program against MSXML objects from C++ or from scripting languages as JScript and VBScript. Like all the COM and MSXML objects are programmatically instantiated by CLSID (globally unique identifier) or programmatic identifier.

The DAM module was coded in C++, directly in the MS source codes by using the MFC wrapper as a part of the MSXML. The wrapper provides a simplified interface to Microsoft's XML DLL. This library makes it easy to write the MFC code that loads an XML document into memory as a tree structure, modifies the XML tree and saves it back to a disk as the file.

The XML data format for archiving is based on the implementation standard called DOM (Document Object Model). The DOM takes the complete XML document and transforms it into a node tree that can be accessed at will by the application. The node tree closely mirrors the document itself and, since the complete document is in memory, random searches are allowed. The next DOM feature is that it guarantees the document is "well-formed" when saving it. This is possible because the details of the syntax are hidden from developers.

The used Microsoft's DOM is contained in the library msxml.dll and is installed as a part of the Internet Explorer 5 and later. Microsoft continually updates their MSXML because of the dynamics of the W3C (World Wide Web Consortium) specifications and they add new features.

In the DAM module, the Microsoft's MSXML 3 was deployed. All the functionality and features are contained in the library msxml3.dll. This DLL has the MFC wrapper class called CXmlIF to make it more manageable for the application developers. The CXmlIF is implemented in the C++ source code files XmlIF.h and XmlIF.cpp [45] included in the MS project.

When the CXmlIF object is constructed (instantiated), its member method `CXmlIF::Initialize()`is called. It is used to initialize the COM library through `CoInitialize()`and to create an instance of the DOM Document object, `IXMLDOMDocument`, that is returned through the interface pointer `m_pDoc` using the COM method `CoCreateInstance()`. There is used the flag, `m_bInitialized`, to determine if the construction is successful and if the document object pointer is valid for the usage.

When all the experiment data are accumulated and inserted to the XML tree structure, the document saving is performed to an xml file. There is called the method `CXmlIF::SaveDocument(const char* szURL)` that saves the content of the DOM object pointed to by `m_pDoc` to the specified file.

The destructor calls `UnInitialize()` to clean up when the CXmlIF object is destroyed (removed from the dynamic memory) before the MS shutdown.

As previously mentioned, the DOM represents a node hierarchy of the XML document. Such the generated document can contain a prolog, body and epilog. The DAM's XML files contain the prolog `<?xml version="1.0"?>` that is a processing instruction node placed at the beginning of the file.

The top node of the document is appropriately named the "document node." The body of the XML document contains all the meat. There is just one node at the top of the body. All other nodes are descendants of this node. In fact, the node at the top of the body is a descendent of the document node.

When the client starts the measurement, data are being gathered (scanned) from the attached metes/probes and devices. The CXmlIF object is created and initialized, then the descending nodes (children) are being created and respective data are being inserted to these nodes. This process starts by calling the method `CXmlIF::CreateChild()` using the node name as the first argument. The XML file formation is listed in Figure 38, focused on the CXmlIF object construction, and all the document variables and nodes declaration before the archiving.

```
CXmlIF Xml;

BOOL XmlInitialized = FALSE;
BOOL ExistingDocument = FALSE;
BOOL ExistingBurst = FALSE;

int ClientID = 0;
std::string ClientSS = "";

MS_XML_NS::IXMLDOMNode *pDocumentNode = NULL;
MS_XML_NS::IXMLDOMNode *pRecordNode = NULL;
MS_XML_NS::IXMLDOMNode *pExperimentNode = NULL;
MS_XML_NS::IXMLDOMNode *pLibraryNode = NULL;
MS_XML_NS::IXMLDOMNode *pDataNode = NULL;
MS_XML_NS::IXMLDOMNode *pLegendNode = NULL;

MS_XML_NS::IXMLDOMAttribute *pAttribute = NULL;

std::map<uint, CLibraryXNode> LibraryXNode;

std::vector<MS_XML_NS::IXMLDOMNode*> LegendNodes;
std::vector<CControlItemXNode> DeviceList;
std::vector<CControlXNode> ControlNodes;
std::vector<CObservationXNode> ObservationNodes;

CControlXNode *pNewControlNode = NULL;
CControlItemXNode *pNewControlItem = NULL;
CControlRowXNode *pNewControlRow = NULL;
CObservationXNode *pNewObservationNode = NULL;
CObservationItemXNode *pNewObservationItemNode = NULL;
CObservationtRowXNode *pNewObservationRowNode = NULL;

CExperiment Experiment;

if (Xml.IsInitialized())
{
    Xml.SetDocumentNode();
    pDocumentNode = Xml.GetDocumentNode();
    Xml.InsertProcessingInst();

    if (pDocumentNode)
    {
        pRecordNode = Xml.CreateChild(pDocumentNode, "record");
        pExperimentNode = Xml.CreateChild(pRecordNode, "experiment");
        pLibraryNode = Xml.CreateChild(pExperimentNode, "library");
        pDataNode = Xml.CreateChild(pExperimentNode, "data");
        pLegendNode = Xml.CreateChild(pDataNode, "legend");
    }
}
```

Figure 38 Implementation of the DAM module that constructs the CXmlIF object with its variables and nodes in the source code file ServerListeningSocket.cpp to create the xml file

The data archiving process is performed in the loop located in the respective sections, according to the RL types (slow and fast experiment), which are coded in the ServerListeningSocket.cpp file. The archiving section used for the fast experiment is listed in Figure 39 that presents a creation of the descending node to which is inserted a new record (row) with the row number, current time and measured data pin read in the respective meter.

```
for (it = itflo->second.Data.begin(); it != itflo->second.Data.end(); it++)
{
    if ((XmlInitialized == true) && (pItemObjectNode == true) && (pItemNode == true))
    {
        RowCount++;

        pNewObservationRowNode = new CObservationtRowXNode();

        Node[0].Format("%d", RowCount);                              //Current row
        Node[1].Format("%.0f", (it2->TimeEntry - BaseTime) / 100.0f); //Current time
        Node[2].Format("%.3f", it2->Value);                          //Measured value

        pNewObservationRowNode->pObservationRowXNode =
            Xml.CreateChild(pItemObjectNode->pObservationItemXNode, "row");

        pAttribute = Xml.CreateAttribute(
            pNewObservationRowNode->pObservationRowXNode, "name", Node[0]));

        pAttribute->Release();

        pAttribute = Xml.CreateAttribute(
            pNewObservationRowNode->pObservationRowXNode, "time", Node[1]));

        pAttribute->Release();

        pAttribute = Xml.CreateAttribute(
            pNewObservationRowNode->pObservationRowXNode, "value", Node[2]));

        pAttribute->Release();

        pItemObjectNode->RowNodes.push_back(*pNewObservationRowNode);

        if (pNewObservationRowNode != NULL)
        {
            delete pNewObservationRowNode;
            pNewObservationRowNode = NULL;
        }
    }
}
```

Figure 39 Implementation of the data archiving process to create the descending node "row" to which is inserted a new record with the row number "name", current time step "time" and measured data pin "value" read the meter

As mentioned, the MS handles specific messages to communicate with clients to monitor and control the RL rig. These messages are conceived for different operational purposes. They are received and sent when the RL starts, initializes all its components and performs the measurement. Each message has a unique identifier distinguished by the MS before producing the corresponding client's action. All the messages are represented by the following command:

- *COMMAND_START_EXPERIMENT* - It obtains details about the slow experiment when it started its running.
- *COMMAND_STOP_EXPERIMENT* - It obtains details about the slow experiment when it stopped its running.
- *COMMAND_READ_EXPERIMENT* - It obtains measured data form the slow experiment when the client demands.
- *COMMAND_BURST_EXPERIMENT* - It obtains measured data form the fast experiment when the client demands.
- *COMMAND_IS_EXPERIMENT_RUNNING* - It obtains current measured data coming from the slow experiment.
- *COMMAND_READ_FAST_LOG* - It obtains the measured data from the high frequency process for a selected device.
- *COMMAND_MEASURE_DEVICE_VALUE* - It gains the value measured from a selected device available in the devices list.
- *COMMAND_SET_DEVICE_VALUE* - It sets the value that controls a selected device available in the devices list.
- *COMMAND_GET_SOURCE_DEVICE_HANDLE* - It obtains the device handlers stated for the reading measured data.
- *COMMAND_GET_DESTINATION_DEVICE_HANDLE* - It obtains the device handlers stated for the writing control values.
- *COMMAND_GET_CONNECTED_CLIENT* - It returns the client details.
- *COMMAND_GET_QUEUE* - It obtains the list of all connected clients.
- *COMMAND_LIST_DEVICES* - It obtains the list of involved devices.
- *COMMAND_LIST_EXPERIMENTS* - It obtains the list of experiments.
- *COMMAND_DELAY_FLUSH* - It activates the data flushing delay.
- *COMMAND_QUERY _POS* - It obtains the position of an active client.
- *COMMAND_READ_LOG* - It reads the history data for a selected device.
- *COMMAND_EXIT* - It terminates the client's connection.

The DAM creates the CXmlIF object when a new client connects to the RL and sends initial messages from the web page. Important details (status and configuration) about the RL are stored to this object. There are details about the connected client, attached physical modules and preset settings to set all widgets displayed on the web page. The UML (Unified Modeling Language) activity diagram depicted in Figure 40 presents the case of sending the client's message *COMMAND_BURST_EXPERIMENT* to perform a measurement on the rig of the fast experiment with the enabled data archiving to the xml file.

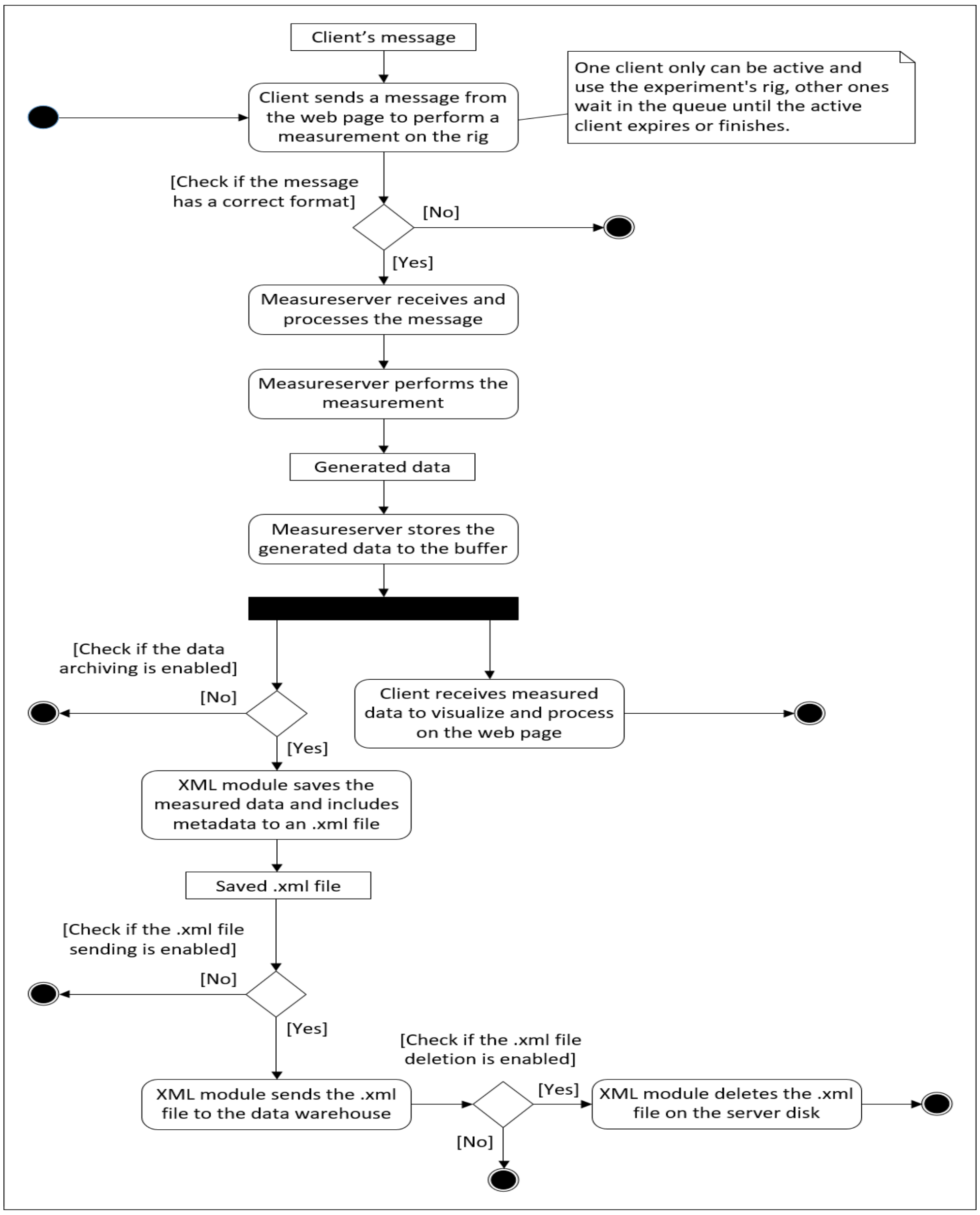
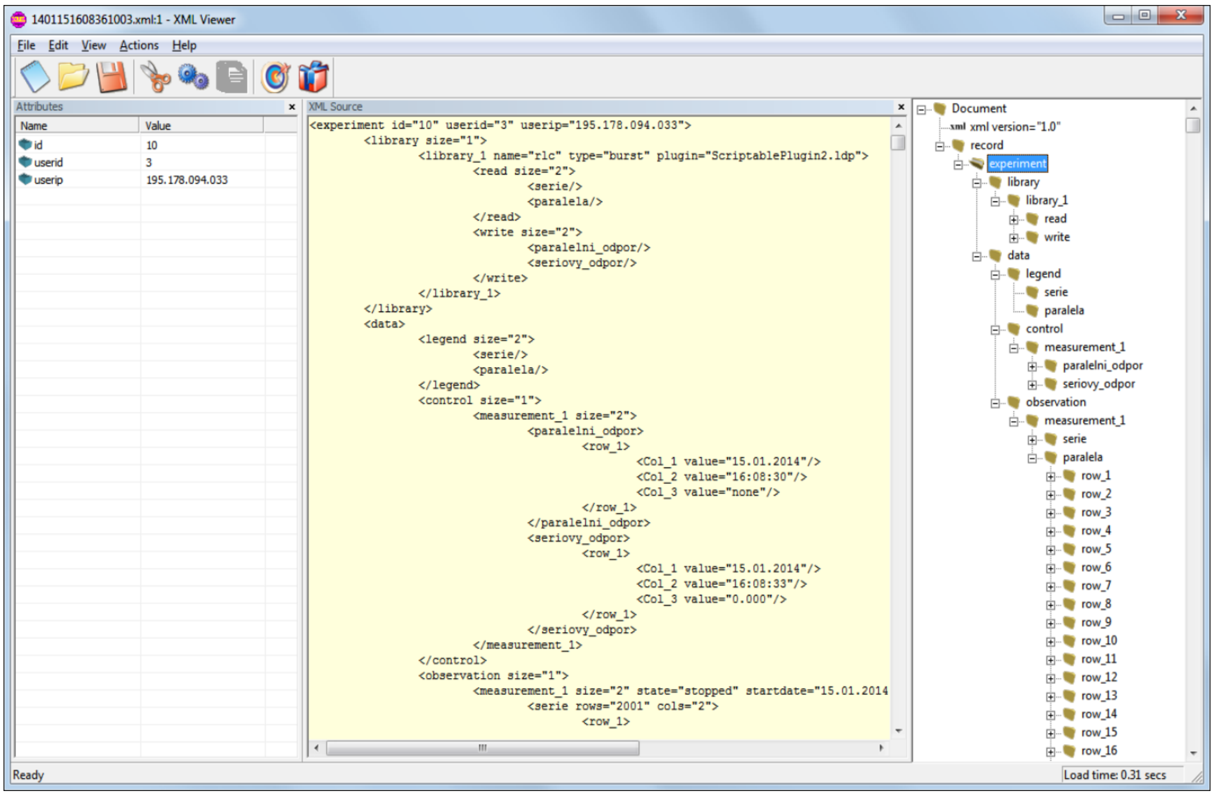

Client's message
Client sends a message from the web page to perform a measurement on the rig
One client only can be active and use the experiment's rig, other ones wait in the queue until the active client expires or finishes.
[Check if the message has a correct format]
[No]
[Yes]
Measureserver receives and processes the message
Measureserver performs the measurement
Generated data
Measureserver stores the generated data to the buffer
[Check if the data archiving is enabled]
[No]
[Yes]
Client receives measured data to visualize and process on the web page
XML module saves the measured data and includes metadata to an .xml file
Saved .xml file
[Check if the .xml file sending is enabled]
[No]
[Yes]
[Check if the .xml file deletion is enabled]
XML module sends the .xml file to the data warehouse
[Yes]
XML module deletes the .xml file on the server disk
[No]

Figure 40 Activity diagram of the archiving process performed when the client's message "COMMAND_BURST_EXPERIMENT" is sent to the MS to save data from the fast experiment to the file

As the example, the 1401151608361003.xml file archiving measured data is presented in Figure 41 that includes the data structure and the respective selected branch attributes. The filename is consisted of the date and time stamp of the performed measurement, client's identifier and sequence identifier.


1401151608361003.xml:1 - XML Viewer
File Edit View Actions Help
Attributes
XML Source
Ready
Load time: 0.31 secs

Name	Value
id	10
userid	3
userip	195.178.094.033

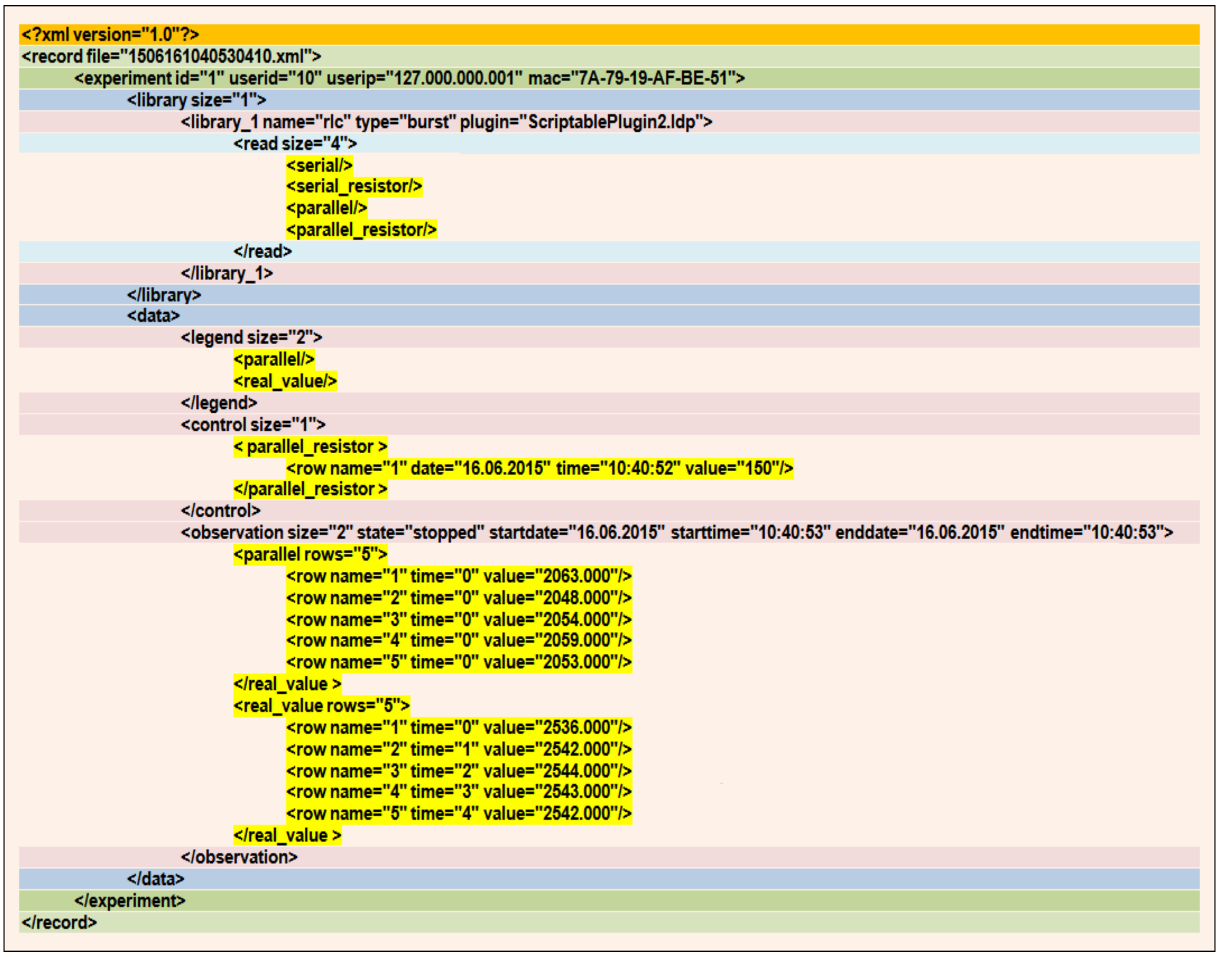
```
<experiment id="10" userid="3" userip="195.178.094.033">
        <library size="1">
                <library_1 name="rlc" type="burst" plugin="ScriptablePlugin2.ldp">
                        <read size="2">
                                <serie/>
                                <paralela/>
                        </read>
                        <write size="2">
                                <paralelni_odpor/>
                                <seriovy_odpor/>
                        </write>
                </library_1>
        </library>
        <data>
                <legend size="2">
                        <serie/>
                        <paralela/>
                </legend>
                <control size="1">
                        <measurement_1 size="2">
                                <paralelni_odpor>
                                        <row_1>
                                                <Col_1 value="15.01.2014"/>
                                                <Col_2 value="16:08:30"/>
                                                <Col_3 value="none"/>
                                        </row_1>
                                </paralelni_odpor>
                                <seriovy_odpor>
                                        <row_1>
                                                <Col_1 value="15.01.2014"/>
                                                <Col_2 value="16:08:33"/>
                                                <Col_3 value="0.000"/>
                                        </row_1>
                                </seriovy_odpor>
                        </measurement_1>
                </control>
                <observation size="1">
                        <measurement_1 size="2" state="stopped" startdate="15.01.2014
                                <serie rows="2001" cols="2">
                                        <row_1>
```

```
Document
  xml version="1.0"
  record
    experiment
      library
        library_1
          read
          write
      data
        legend
          serie
          paralela
        control
          measurement_1
            paralelni_odpor
            seriovy_odpor
        observation
          measurement_1
            serie
            paralela
              row_1
              row_2
              row_3
              row_4
              row_5
              row_6
              row_7
              row_8
              row_9
              row_10
              row_11
              row_12
              row_13
              row_14
              row_15
              row_16
```

Figure 41 Example of the XML file that is generated by the DAM's functions; the file covers the measured data, metadata and control values affecting the measurement. The viewer presents the three panels, the Attributes panel showing attributes of the "experiment" branch, the XML source panel listing the data structure and the branches tree panel

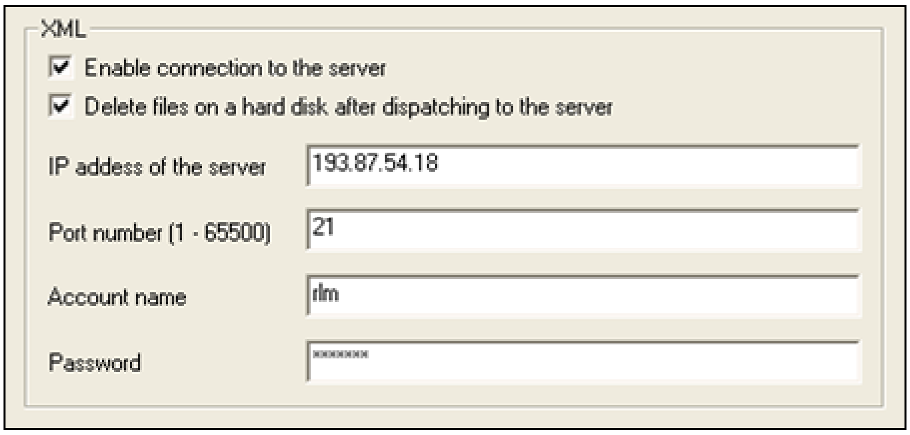
Further example of the archive file is introduced in Figure 42 that exposes a content of the 1506161040530410.xml file presenting the particular branches as the subsection elements. The file content is highlighted by the colored lines to distinguish respective subsections in the data structure.

The DAM module also has the user interface that provides activating of the direct connection to the data warehouse and the file deletion after dispatching. It allows setting the IP address, port number, account name and password to establish the communication with the data warehouse. The user interface is presented in Figure 43 as the XML section residing in the Diagnosis dialog box belonging the MS's GUI used for the general setup.



```
<?xml version="1.0"?>
<record file="1506161040530410.xml">
    <experiment id="1" userid="10" userip="127.000.000.001" mac="7A-79-19-AF-BE-51">
        <library size="1">
            <library_1 name="rlc" type="burst" plugin="ScriptablePlugin2.ldp">
                <read size="4">
                    <serial/>
                    <serial_resistor/>
                    <parallel/>
                    <parallel_resistor/>
                </read>
            </library_1>
        </library>
        <data>
            <legend size="2">
                <parallel/>
                <real_value/>
            </legend>
            <control size="1">
                < parallel_resistor >
                    <row name="1" date="16.06.2015" time="10:40:52" value="150"/>
                </parallel_resistor >
            </control>
            <observation size="2" state="stopped" startdate="16.06.2015" starttime="10:40:53" enddate="16.06.2015" endtime="10:40:53">
                <parallel rows="5">
                    <row name="1" time="0" value="2063.000"/>
                    <row name="2" time="0" value="2048.000"/>
                    <row name="3" time="0" value="2054.000"/>
                    <row name="4" time="0" value="2059.000"/>
                    <row name="5" time="0" value="2053.000"/>
                </real_value >
                <real_value rows="5">
                    <row name="1" time="0" value="2536.000"/>
                    <row name="2" time="1" value="2542.000"/>
                    <row name="3" time="2" value="2544.000"/>
                    <row name="4" time="3" value="2543.000"/>
                    <row name="5" time="4" value="2542.000"/>
                </real_value >
            </observation>
        </data>
    </experiment>
</record>
```

Figure 42 Example of the XML structure that presents the 1506161040530410.xml file to point out the subsection elements. The file comes from the RL "Transients in RLC circuits". The content is highlighted by the colored lines to distinguish particular subsections; the identical colored line pairs indicate subsections at the same level. The yellow elements, "control" and "observation", expose the control value and two measured buffers



XML

☑ Enable connection to the server

☑ Delete files on a hard disk after dispatching to the server

IP address of the server	193.87.54.18
Port number (1 - 65500)	21
Account name	rlm
Password	*******

Figure 43 Example setup of the user interface that presents the XML parameters used for the direct connection to the data warehouse to dispatch xml files

The DAM also provides a component for the clients' activities logging. The component is responsible for continuous writing events reporting activities of the connected clients. The MS assigns a unique identifier for every client to distinguish his activities. These events are being saved to the Users.log file for all the time the client is connected to the system.

The clients' activities logging process is designed to run in several sections of the MS to log respective events. It is deployed in the section where the clients connect/disconnect or change their status. Further logging process is involved in the section for clients' interventions (changing slider positions, button states and other widgets to control the experiment or modify options).

When a new client successfully connects to the MS, the logging component makes a record (date, time, IP address and client's unique identifier) about this event to the Users.log file as shown in Figure 44 that presents this activity through the C++ code residing in the ServerListeningSocket.cpp file.

```
if (ConLog != NULL)
{
    CTime Time = CTime::GetCurrentTime();

    LogFileProtectionLock.Lock();

    fprintf(ConLog,
    "(%d.%d.%d at %d:%d:%d) User connected from:%d.%d.%d.%d Using ID:%u\n",
    Time.GetDay(), Time.GetMonth(), Time.GetYear(),
    Time.GetHour(), Time.GetMinute(), Time.GetSecond(),
    IP[0], IP[1], IP[2], IP[3], USERID);

    fflush(ConLog);

    LogFileProtectionLock.Unlock();
}
```

Figure 44 Section of the clients' activities logging process implemented in the ServerListeningSocket.cpp source code file that is activated when the client connects to the MS to make a record of this event to the Users.log file

Further important activity is the logging process of the clients' interventions performed during the experimentation. These events are especially interesting for the RL designers in the didactical aspects.

When the client changes a value (for example, sets a motor speed or position) that is available on the web page by using the respective slider, all these changes are continuously recorded to the Users.log file. This client's activity is presented in Figure 45, as the C++ code located in the ServerListeningSocket.cpp file, to make one record (date, time, client's unique identifier, device name and new device value) that is finally saved to the file.

```
if (ConLog != NULL)
{
    CTime Time = CTime::GetCurrentTime();

    DeviceProtectionLock.Lock();
    LogFileProtectionLock.Lock();

    fprintf(ConLog,
    "(%d.%d.%d at %d:%d:%d) User %u sets device ID:\"%s\" to value:%f\n",
    Time.GetDay(), Time.GetMonth(), Time.GetYear(),
    Time.GetHour(), Time.GetMinute(), Time.GetSecond(), USERID,
    MasterDialog->m_Devices[DeviceHandle].FullIDName.c_str(), Value);

    fflush(ConLog);

    LogFileProtectionLock.Unlock();
    DeviceProtectionLock.Unlock();
}
```

Figure 45 Section of the clients' activities logging process that is located in the ServerListeningSocket.cpp source code file activated when the client changes a device value to make one event record saved to the Users.log file

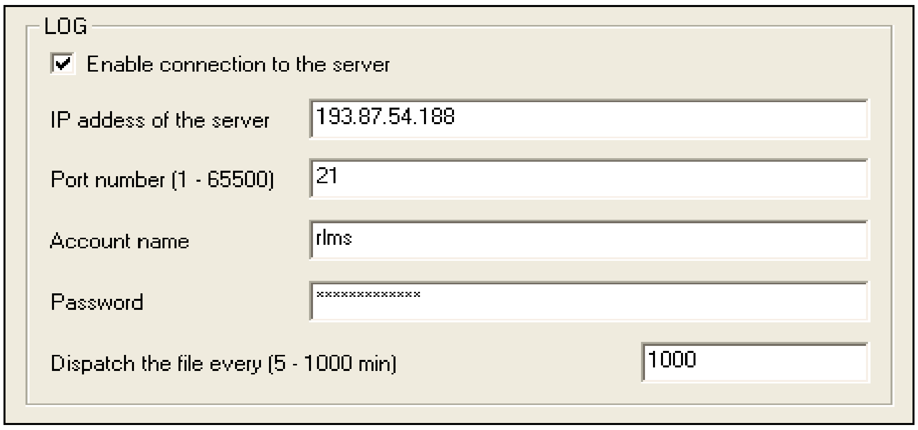
Furthermore, the component for the clients' activities logging is able to record events incoming from the MS functioning like its startup, shutdown, loaded device plugins and data pins allocated in the experiment.

The example record stored in the Users.log file is presented in Figure 46 that exposes the date and time stamp, client's unique identifier assigned when he is connected to distinguish his activities. There are also the records "sets device" indicating the changed parameters for the real and simulated process.

```
(1.9.2016 at 21:30:6) User connected from:195.178.94.33  Using ID:1
(1.9.2016 at 21:30:8) User 1 started burst experiment raw with duration 150. Result is stored as 1
(1.9.2016 at 21:30:9) User 1 read data from experiment 1
(1.9.2016 at 21:30:11) User 1 sets device ID:"rlc_input_serial" to value:30.00
(1.9.2016 at 21:30:14) User 1 started burst experiment raw with duration 150. Result is stored as 2
(1.9.2016 at 21:30:15) User 1 read data from experiment 2
(1.9.2016 at 21:30:40) User 1 sets device ID:"rlc_input_serial" to value:100.00
(1.9.2016 at 21:30:43) User 1 sets device ID:"rlc_input_simulation_L" to value:1.31
(1.9.2016 at 21:31:10) User 1 sets device ID:"rlc_input_simulation_C" to value:1.70
(1.9.2016 at 21:31:15) User 1 sets device ID:"rlc_input_simulation_RL" to value:60.00
(1.9.2016 at 21:32:00) Users 1 connection closed
```

Figure 46 Example record of the clients' activities located in the Users.log file to archive to the data warehouse

The clients' activities logging component has the respective user interface to provide activating of the connection to the data warehouse. Furthermore, it allows setting the IP address, port number, account name and password placed for the communication with the data warehouse. The last included parameter is defined for the dispatching periodicity as introduced in Figure 47.


LOG

☑ Enable connection to the server

IP addess of the server	193.87.54.188
Port number (1 - 65500)	21
Account name	rlms
Password	*************
Dispatch the file every (5 - 1000 min)	1000

Figure 47 Example setup of the user interface that presents the LOG parameters used for configuring of the clients' activities logging component

Summary of the subchapter "6.1 Data archiving and logging module": This subchapter fulfils the goal No. 1 and 2. The remote laboratories produce excessive, even big data, produced by measured ones, which are generated by measuring process and clients' activities data that should be stored for the later recovery and processing. This constitutes the improvement in the intelligent version of the Measureserver unit. The novelty and the scientific approach rest in the use of XML data output for the online analysis.

6.2 First level diagnostics module

The second task of the doctoral thesis was to design and implement the module called MUD (Measureserver unit diagnostics) used as the first level diagnostics. The module provides to RL administrators information to ensure the MS functioning with the aim of avoiding or reducing occasional failures caused by various influences. The MUD also has a self-recovery mechanism to recover the MS functioning after failure events. The typical failure is caused by attached device plugins, which can suddenly stop communicating with their ADDA interface cards or physical modules. The diagnostics is optional that can be disabled to avoid the compatibility conflict with older experiments.

The MUD concept is based on the checking operational sections (blocks) inside the MS core including limited corrections when the evaluating conditions detect failure states in the checked sections. This module is consisting of two internal components working independently in different threads (sequence of the instructions, which can be executed in the application concurrently with other such sequences). These threads work continuously as the background processes, which include many functions to complete specific tasks.

6.2.1 Notifying thread component

The first built component called RlmsNotifyingThread is responsible for the notification process that periodically monitors the MS functioning. If the MS status is detected okay (it works properly), the MUD dispatches the notification message "Measureserver is running" to the diagnostic server (DS). This is a superior server that processes all the messages incoming from the MS and sends back commands to request for the RL status or to recover the operation. In case of the mentioned notification message, the DS identifies this message and exposes as the three lights semaphore placed on every web page of the RLs in the REMLABNET. The semaphore is especially useful for connected clients who can see three colors on it; the red one indicates the RL is unavailable, the orange indicates it is reserved and the green signalizes it is ready to use.

The MUD generates all messages in the same text format before dispatching to uniquely distinguish them in the DS. Each message includes a time stamp and additional identifiers, which are placed in square parenthesis, intended for the parsing purposes; these identifying items are listed below:

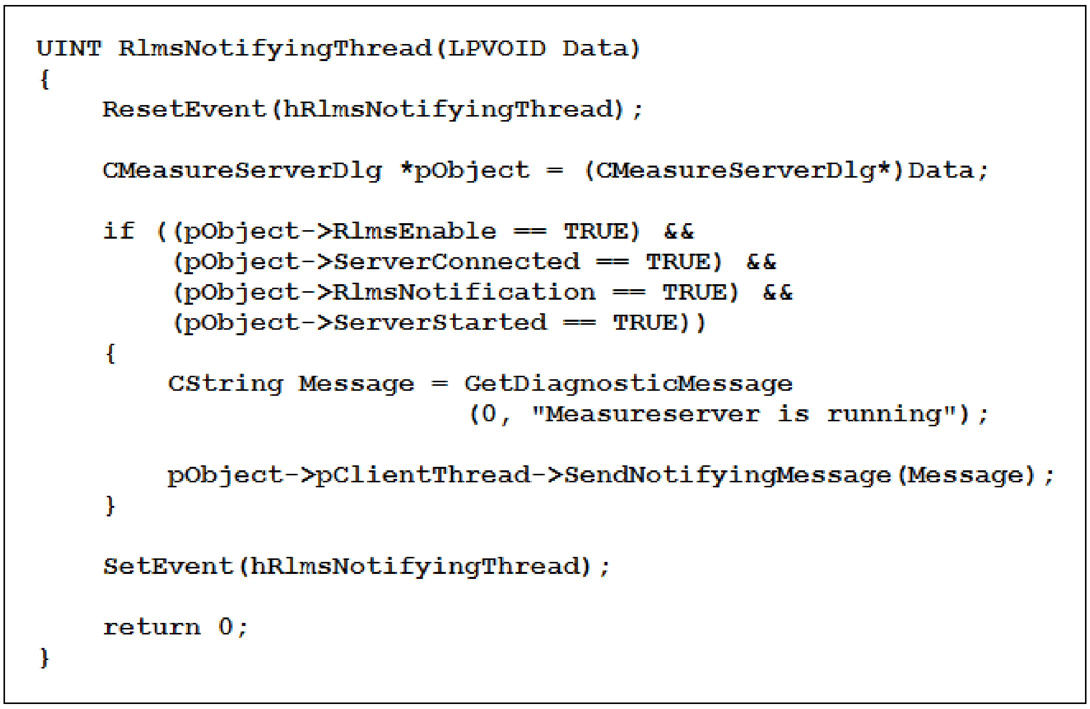
1. [DATE] - current date,
2. [TIME] - current time,
3. [MAC] - network card's MAC (Media Access Control) address,
4. [IP] - computer's IP (Internet Protocol) address,
5. [COMPUTER NAME] - computer name (identifier),
6. [EXPERIMENT NAME] - experiment name (identifier),
7. [MESSAGE CODE] - unique message code (number),
8. [MESSAGE LENGTH] - total message length (number),
9. [MESSAGE TEXT] - action/event occurrence description.

The diagnostic messages are created and formatted as text strings by the function `GetDiagnosticMessage(int Code, CString Text)` that is located in the MeasureServerDlg.cpp file. The message "Measureserver is running" is generated with the appended items and dispatched to the DS where it has the following form intelligible for the RL administrators:

[10.05.2017] [15:20:30] [7A-79-19-25-10-1C] [192.168.1.100] [LAB] [RLC] [0] [114] [Measureserver is running].

This notification message is processed separately by the DS that receives it periodically. If the MS stops dispatching this notification by reason of its crash or disconnection, then the DS qualifies this inactivity and sets a red color on the client's semaphore after some time. The message dispatching is realized by using the CAsyncSocket object to ensure an asynchronous communication with the DS. Shortly to explain, the object's class encapsulates the Windows Socket API providing an object-oriented abstraction to use such sockets in conjunction with the MFC during the application development.

The component represented by the RlmsNotifyingThread function is exposed in Figure 48 that shows periodic dispatching of the notification message.



```
UINT RlmsNotifyingThread(LPVOID Data)
{
    ResetEvent(hRlmsNotifyingThread);

    CMeasureServerDlg *pObject = (CMeasureServerDlg*)Data;

    if ((pObject->RlmsEnable == TRUE) &&
        (pObject->ServerConnected == TRUE) &&
        (pObject->RlmsNotification == TRUE) &&
        (pObject->ServerStarted == TRUE))
    {
        CString Message = GetDiagnosticMessage
                          (0, "Measureserver is running");

        pObject->pClientThread->SendNotifyingMessage(Message);
    }

    SetEvent(hRlmsNotifyingThread);

    return 0;
}
```

Figure 48 Source code of the RlmsNotifyingThread function located in the MeasureServerDlg.cpp file to create and dispatch the notification message "Measureserver is running" to the DS for the analysis

6.2.2 Operating thread component

The second component build in the MUD is called RlmsOperatingThread used for various diagnosing operations in the MS core. This component is implemented as a function running periodically to process all messages.

This specific function is designed for three targets. The first implemented target is responsible for the dispatching of the generated diagnostic messages to the DS for the analysis and the distribution to other subsystems. For this purpose, there is used the MUD’s outbound queue in which the all messages are continuously accumulated before the dispatchment.

The second target is focused on receiving the messages from the DS as the commands to accomplish prescribed actions. There is also used a queue, called as the MUD’s inbound queue, in which are cumulated in sequence all the commands incoming from the DS interface.

The third target deals with the accomplishment of the commands residing in the MUD’s inbound queue to deliver status information to the RL administrators through the DS interface. These commands provide the following actions:

1. *accept* - It returns the message “RLMS connection accepted” to inform about the MS activity whether it runs correctly.
2. *update* - It returns the message “Experiment report updated” with the one of the following messages, “Experiment is idle”, “Experiment is running” or “Experiment is reserved” to inform about the experiment status.
3. *restart* - It performs that the MS restarts itself and then it returns back the message “Measureserver restarted” to inform about the restart.
4. *reboot* - It performs that the computer reboots itself and then it returns back the message “Computer rebooted” to inform about the reboot.
5. *stop* - It performs that the MS stops running to finish the experiment to make the rig’s modules maintenance or recovery.
6. *detect* - It returns the message “Hardware modules detection updated” with the one of the following messages, the message list of attached physical modules, “Hardware modules detection disabled” or “Hardware modules detection not present” to inform about the status and setup of physical modules. This command relates to the second level diagnostics that is discussed at some length in the next subchapter.

Each command can be accompanied by an IP address as the parameter that determines which RL is concerned to obtain status information from, or to perform required actions on it. If there is not entered the IP address, then all the RLs (connected to the DS) accomplish the incoming commands.

When the RL administrator dispatches a requiring command to the RL through the GUI (installed on the DS to manage the experiments), then the MUD receives and processes it. As an example, the formed and dispatched command “update 10.20.54.66” triggers the action shown in Figure 49.

```
if ((Message.Find("update") != -1) && (Message.Find(IPAddress) != -1))
{
    pObject->pClientThread->DispatchTextMessage
       (GetDiagnosticMessage(6, "Experiment report updated"));

    pObject->pClientThread->DispatchTextMessage
        (GetDiagnosticMessage(7, "Experiment is running"));

    if (pObject->RlmsClientVector.size() <= 0)
    {
        pObject->pClientThread->DispatchTextMessage
            (GetDiagnosticMessage(8, "Experiment is idle"));
    }
    else
    {
        pObject->pClientThread->DispatchTextMessage
            (GetDiagnosticMessage(9, "Experiment is reserved"));

        for(int i = 0; i < pObject->RlmsClientVector.size(); i++)
        {
            pObject->pClientThread->DispatchTextMessage
                (GetDiagnosticMessage(1004, pObject->RlmsClientVector.at(i)));
        }
    }
}
```

Figure 49 Source code of the ”update” command that is triggered by the RL administrator from the DS interface to obtain an experiment status; as the first, it returns the message header “Experiment report updated” followed by the one of the messages “Experiment is running”, “Experiment is idle” or “Experiment is reserved”

Further example, the formed and dispatched command “restart 10.20.54.66” triggers the action shown in Figure 50 to perform the MS restart for the purpose of the RL recovery. This command can be triggered manually by the RL administrator or automatically by the DS based on the incoming notification message “Measureserver is running” if it stops notifying for a longer time.

```
if ((Message.Find("restart") != -1) && (Message.Find(IPAddress) != -1))
{
    pObject->KillTimer(pObject->RlmsNotifyingTimerID);
    pObject->KillTimer(pObject->RlmsOperatingTimerID);
    pObject->KillTimer(pObject->FtpDispatchingTimerID);
    pObject->KillTimer(pObject->HwmdsDetectionTimerID);
    pObject->FtpXmlEnable = FALSE;
    pObject->FtpLogEnable = FALSE;

    CString Message = GetDiagnosticMessage
        (3, "Measureserver restarted");

    pObject->RlmsSendQueue.push(Message);

    std::queue<CString>::size_type SendCounter;

    SendCounter = pObject->RlmsSendQueue.size();

    while (SendCounter > 0)
    {
        if (pObject->ServerConnected == TRUE)
        {
            pObject->pClientThread->DispatchTextMessage
                (pObject->RlmsSendQueue.front());

            pObject->RlmsSendQueue.pop();

            SendCounter = pObject->RlmsSendQueue.size();
        }
    }

    TimerProtectionLock.Unlock();

    pObject->ServerStarted = FALSE;

    SetEvent(hRlmsOperatingThread);

    ((CMeasureServerApp*)AfxGetApp())->Restart();

    return 0;
}
```

Figure 50 Source code of the "restart" command triggered by the RL administrator from the DS interface to restart the MS functioning to recover the experiment; it returns the message "Measureserver restarted", then the MS restarts itself

As mentioned, the MUD is consisted of the two components (functions), RlmsNotifyingThread and RlmsOperatingThread, which work as the threads running concurrently in the MS core. They are both started periodically in the member method `VOID CMeasureServerDlg::OnTimer(UINT TimerVal)` in the CMeasureServerDlg object. The UML activity diagram shown in Figure 51 presents the RlmsOperatingThread functioning. There are involved the three sequential operations in this function; the messages dispatching, the commands receiving and the commands accomplishment with notification.

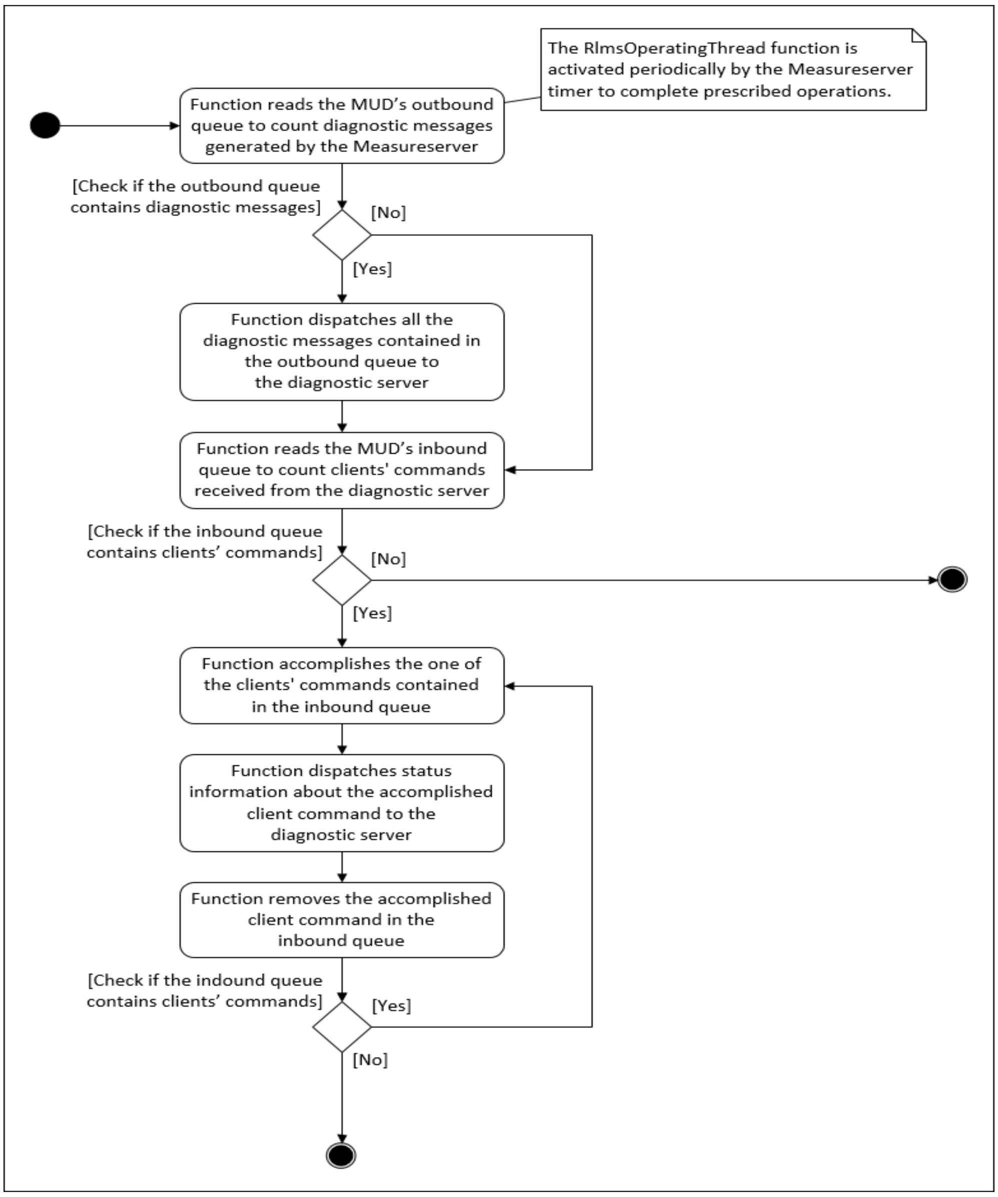
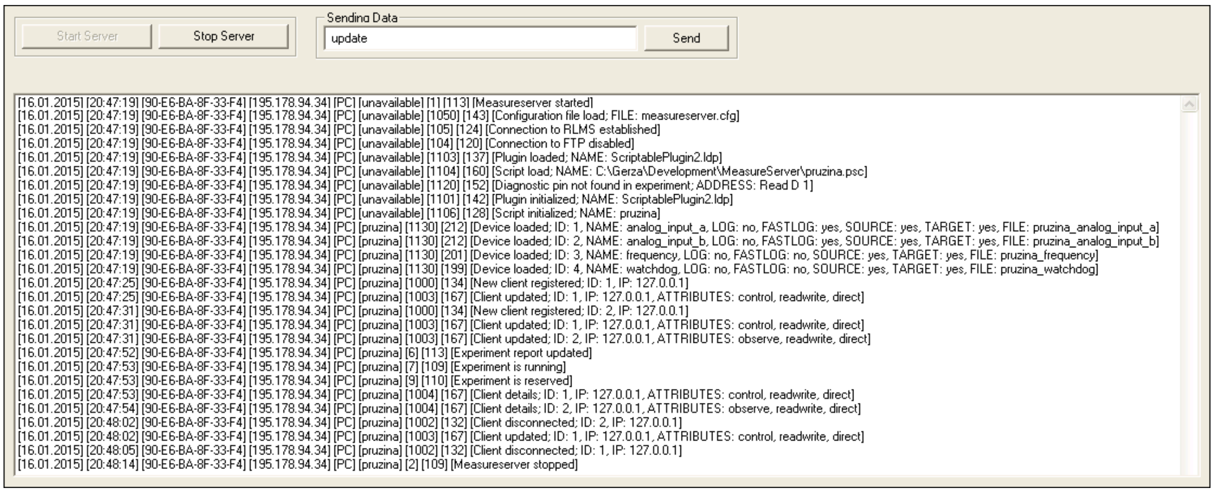

The RlmsOperatingThread function is activated periodically by the Measureserver timer to complete prescribed operations.
Function reads the MUD's outbound queue to count diagnostic messages generated by the Measureserver
[Check if the outbound queue contains diagnostic messages]
[No]
[Yes]
Function dispatches all the diagnostic messages contained in the outbound queue to the diagnostic server
Function reads the MUD's inbound queue to count clients' commands received from the diagnostic server
[Check if the inbound queue contains clients' commands]
[No]
[Yes]
Function accomplishes the one of the clients' commands contained in the inbound queue
Function dispatches status information about the accomplished client command to the diagnostic server
Function removes the accomplished client command in the inbound queue
[Check if the indound queue contains clients' commands]
[Yes]
[No]

Figure 51 Activity diagram of the RlmsOperatingThread component that involves three main operations; the messages dispatching, the commands receiving and the commands accomplishment

The first level diagnostics encapsulates and starts one important function that is an integral part of the second level diagnostics described at full length in the next subchapter. The function is located in the RlmsOperatingThread component to detect and monitor the ISES modules connected and to dispatch messages about status information to the DS for their maintenance.

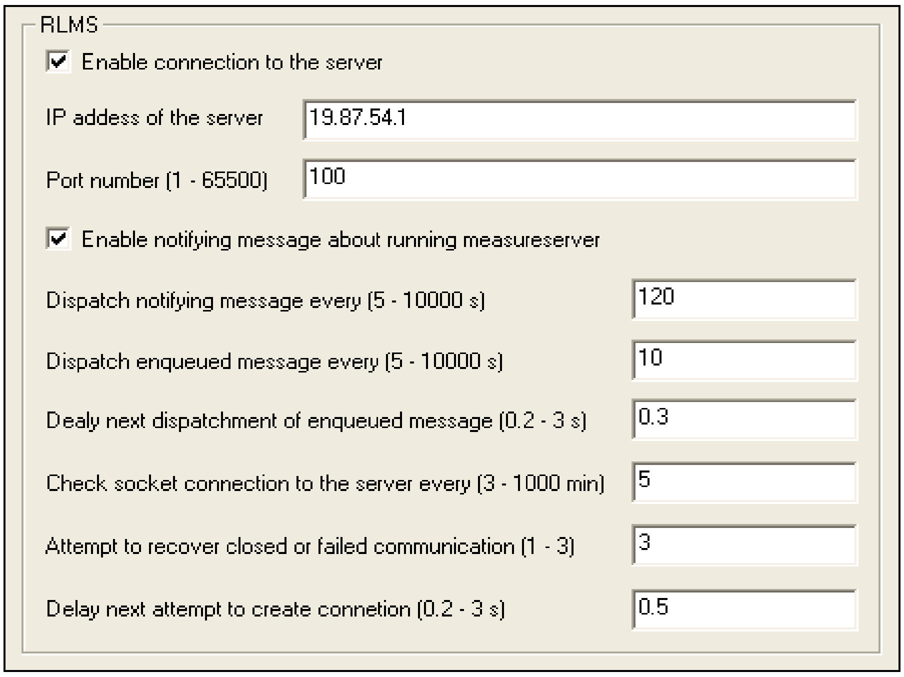
As the example, the "update" command dispatched from the DS to MS unit causes that the MS sends back the list of complex information describing the respective experiment as shown in Figure 52 There is primarily delivered the data and time stamp of the request, the configuration of connected physical modules including attached plugins and registered clients.



```
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1] [113] [Measureserver started]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1050] [143] [Configuration file load; FILE: measureserver.cfg]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [105] [124] [Connection to RLMS established]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [104] [120] [Connection to FTP disabled]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1103] [137] [Plugin loaded; NAME: ScriptablePlugin2.ldp]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1104] [160] [Script load; NAME: C:\Gerza\Development\MeasureServer\pruzina.psc]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1120] [152] [Diagnostic pin not found in experiment; ADDRESS: Read D 1]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1101] [142] [Plugin initialized; NAME: ScriptablePlugin2.ldp]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [unavailable] [1106] [128] [Script initialized; NAME: pruzina]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1130] [212] [Device loaded; ID: 1, NAME: analog_input_a, LOG: no, FASTLOG: yes, SOURCE: yes, TARGET: yes, FILE: pruzina_analog_input_a]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1130] [212] [Device loaded; ID: 2, NAME: analog_input_b, LOG: no, FASTLOG: yes, SOURCE: yes, TARGET: yes, FILE: pruzina_analog_input_b]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1130] [201] [Device loaded; ID: 3, NAME: frequency, LOG: no, FASTLOG: no, SOURCE: yes, TARGET: yes, FILE: pruzina_frequency]
[16.01.2015] [20:47:19] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1130] [199] [Device loaded; ID: 4, NAME: watchdog, LOG: no, FASTLOG: no, SOURCE: yes, TARGET: yes, FILE: pruzina_watchdog]
[16.01.2015] [20:47:25] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1000] [134] [New client registered; ID: 1, IP: 127.0.0.1]
[16.01.2015] [20:47:25] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1003] [167] [Client updated; ID: 1, IP: 127.0.0.1, ATTRIBUTES: control, readwrite, direct]
[16.01.2015] [20:47:31] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1000] [134] [New client registered; ID: 2, IP: 127.0.0.1]
[16.01.2015] [20:47:31] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1003] [167] [Client updated; ID: 1, IP: 127.0.0.1, ATTRIBUTES: control, readwrite, direct]
[16.01.2015] [20:47:31] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1003] [167] [Client updated; ID: 2, IP: 127.0.0.1, ATTRIBUTES: observe, readwrite, direct]
[16.01.2015] [20:47:52] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [6] [113] [Experiment report updated]
[16.01.2015] [20:47:53] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [7] [109] [Experiment is running]
[16.01.2015] [20:47:53] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [9] [110] [Experiment is reserved]
[16.01.2015] [20:47:53] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1004] [167] [Client details; ID: 1, IP: 127.0.0.1, ATTRIBUTES: control, readwrite, direct]
[16.01.2015] [20:47:54] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1004] [167] [Client details; ID: 2, IP: 127.0.0.1, ATTRIBUTES: observe, readwrite, direct]
[16.01.2015] [20:48:02] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1002] [132] [Client disconnected; ID: 2, IP: 127.0.0.1]
[16.01.2015] [20:48:02] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1003] [167] [Client updated; ID: 1, IP: 127.0.0.1, ATTRIBUTES: control, readwrite, direct]
[16.01.2015] [20:48:05] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [1002] [132] [Client disconnected; ID: 1, IP: 127.0.0.1]
[16.01.2015] [20:48:14] [90-E6-BA-8F-33-F4] [195.178.94.34] [PC] [pruzina] [2] [109] [Measureserver stopped]
```

Figure 52 Example of the "update" command dispatched from the DS to MS unit to obtain back the complex information describing the respective experiment

The MUD is allowed to configure by the respective user interface. It provides the RL administrators the specific parameters to set the connection to the DS and to also set the management of notification and faulty messages during the operation as presented on the example settings shown in Figure 53.



RLMS

[✓] Enable connection to the server

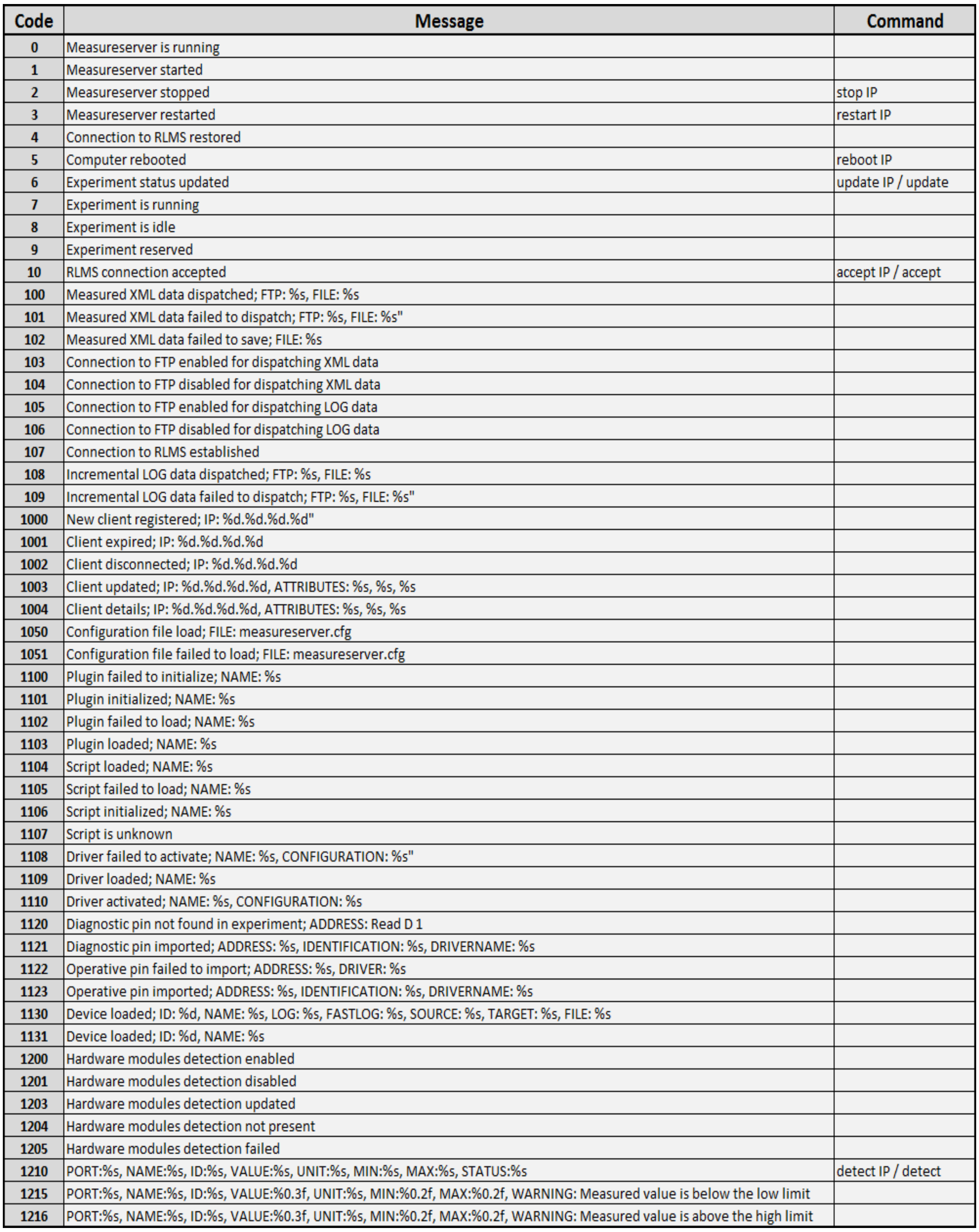
IP addess of the server	19.87.54.1
Port number (1 - 65500)	100

[✓] Enable notifying message about running measureserver

Dispatch notifying message every (5 - 10000 s)	120
Dispatch enqueued message every (5 - 10000 s)	10
Dealy next dispatchment of enqueued message (0.2 - 3 s)	0.3
Check socket connection to the server every (3 - 1000 min)	5
Attempt to recover closed or failed communication (1 - 3)	3
Delay next attempt to create connetion (0.2 - 3 s)	0.5

Figure 53 Example settings of the MUD module available in the MS unit to configure the direct connection to the DS and the messaging process

The notification and faulty messages, which are generated by occurred events, and the involved commands for the management are listed in Figure 54.



Code	Message	Command
0	Measureserver is running	
1	Measureserver started	
2	Measureserver stopped	stop IP
3	Measureserver restarted	restart IP
4	Connection to RLMS restored	
5	Computer rebooted	reboot IP
6	Experiment status updated	update IP / update
7	Experiment is running	
8	Experiment is idle	
9	Experiment reserved	
10	RLMS connection accepted	accept IP / accept
100	Measured XML data dispatched; FTP: %s, FILE: %s	
101	Measured XML data failed to dispatch; FTP: %s, FILE: %s"	
102	Measured XML data failed to save; FILE: %s	
103	Connection to FTP enabled for dispatching XML data	
104	Connection to FTP disabled for dispatching XML data	
105	Connection to FTP enabled for dispatching LOG data	
106	Connection to FTP disabled for dispatching LOG data	
107	Connection to RLMS established	
108	Incremental LOG data dispatched; FTP: %s, FILE: %s	
109	Incremental LOG data failed to dispatch; FTP: %s, FILE: %s"	
1000	New client registered; IP: %d.%d.%d.%d"	
1001	Client expired; IP: %d.%d.%d.%d	
1002	Client disconnected; IP: %d.%d.%d.%d	
1003	Client updated; IP: %d.%d.%d.%d, ATTRIBUTES: %s, %s, %s	
1004	Client details; IP: %d.%d.%d.%d, ATTRIBUTES: %s, %s, %s	
1050	Configuration file load; FILE: measureserver.cfg	
1051	Configuration file failed to load; FILE: measureserver.cfg	
1100	Plugin failed to initialize; NAME: %s	
1101	Plugin initialized; NAME: %s	
1102	Plugin failed to load; NAME: %s	
1103	Plugin loaded; NAME: %s	
1104	Script loaded; NAME: %s	
1105	Script failed to load; NAME: %s	
1106	Script initialized; NAME: %s	
1107	Script is unknown	
1108	Driver failed to activate; NAME: %s, CONFIGURATION: %s"	
1109	Driver loaded; NAME: %s	
1110	Driver activated; NAME: %s, CONFIGURATION: %s	
1120	Diagnostic pin not found in experiment; ADDRESS: Read D 1	
1121	Diagnostic pin imported; ADDRESS: %s, IDENTIFICATION: %s, DRIVERNAME: %s	
1122	Operative pin failed to import; ADDRESS: %s, DRIVER: %s	
1123	Operative pin imported; ADDRESS: %s, IDENTIFICATION: %s, DRIVERNAME: %s	
1130	Device loaded; ID: %d, NAME: %s, LOG: %s, FASTLOG: %s, SOURCE: %s, TARGET: %s, FILE: %s	
1131	Device loaded; ID: %d, NAME: %s	
1200	Hardware modules detection enabled	
1201	Hardware modules detection disabled	
1203	Hardware modules detection updated	
1204	Hardware modules detection not present	
1205	Hardware modules detection failed	
1210	PORT:%s, NAME:%s, ID:%s, VALUE:%s, UNIT:%s, MIN:%s, MAX:%s, STATUS:%s	detect IP / detect
1215	PORT:%s, NAME:%s, ID:%s, VALUE:%0.3f, UNIT:%s, MIN:%0.2f, MAX:%0.2f, WARNING: Measured value is below the low limit	
1216	PORT:%s, NAME:%s, ID:%s, VALUE:%0.3f, UNIT:%s, MIN:%0.2f, MAX:%0.2f, WARNING: Measured value is above the high limit	

Figure 54 Notification and faulty messages generated by the MS unit when events occur and the control commands coming from the DS to use for the management of the unit in case of problems or updates

Summary of the subchapter "6.2 First level diagnostics module": This subchapter deals with the goal No. 3. The diagnostics of remote laboratories is not to be found, to our knowledge, in any published laboratory, irrespective of the fact of its importance. The first level diagnostics serves to inform the client about general function of the respective remote laboratory. It is based on checking the consistency of the function of every plugins and drivers, signaled by the "traffic lights". This constitutes the improvement in the intelligent version of the Measureserver unit. The novelty and the scientific approach rest in the use of real-time checking of the critical points of the units in question.

6.3 Second level diagnostics module

The third task of this doctoral thesis was to design and realize the module called PMD (physical modules diagnostics) used as the second level diagnostics. The module provides the RL administrators the features to detect and monitor the ISES physical modules (meters, sensors, probes and devices) connected to the rig to prevent their failures or disconnections during the measurement.

The physical modules are used to obtain the quantities of electric current and voltage, temperature, speed, sound, etc. They are connected to the ISES control board used as a central board of the RL rig. As the primary aim, the PMD was implemented to detect and determine a kind of the added (attached) module and its configuration, like the number of data channels and sensing range.

The detection principle is based on the periodical testing of input and output ports built in the ISES control board where the physical modules are connected as exchangeable devices. The ADDA conversion is realized by the PCI 1202 interface card via the driver between the MS and the ISES control board. When any physical module is connected, re-connected or removed from the port, then the appropriate technique to update the status is activated.

The physical modules have 15 pins connector (CAN-15) intended to connect channels (A-F) built in the ISES control panel (if used ISES PCI Professional) and every connector disposes of six reserved pins for the detection of the connected physical module. The reserved pins are settled as two triplets numbered 4, 5, 6 and 9, 10, 11. It means, there are six bits used for the detecting purposes. These pins are under the voltage in the range between 0 and +5V. Every triplet is connected to the reserved pin of a 37 pins connector (CAN-37) through the separate resistor ladder with three resistors. The communication between the ISES control board and the RL computer is realized by a cable from the CAN-37 to the PCI 1202 interface card.

The first triplet is defined as the upper part status and the second one is called the lower part status. Current values of both statuses are continuously read by the respective functions on the CAN-37 through the PCI 1202 interface card. These values, represented by the voltage ranging from 0 to +5V, have to be converted to two bytes typed WORD numbers in the range from 2048 to 4095 as listed in Table 2 to show measured voltages on the status pins.

Table 2 Measured voltage on the status pins of the CAN-37 connector and their corresponding numeric representation with the tolerances

Voltage on pin (V)	*Equivalent range of voltage (V)*	*Range of voltage as WORD number*
5,000	*5,000 - 4,167*	*4095 – 3754*
3,333	*4,166 - 2,917*	*3754 - 3242*
2,500	*2,916 - 2,250*	*3242 - 2969*
2,000	*2,249 - 1,844*	*2969 - 2803*
1,689	*1,844 - 1,567*	*2803 - 2689*
1,445	*1,566 - 1,354*	*2689 - 2602*
1,262	*1,353 - 1,192*	*2602 - 2535*
1,121	*1,191 - 0,000*	*2535 - 2047*

The physical modules are constructed in principle to have one detecting pin of the six ones grounded. When this pin indicates +5V, then it is represented as binary 1, and if there is occurred 0V, it is binary 0. If physical module is not connected, all the pins constantly indicate +5V.

The identifiers listed in Table 3 are composed of six bits coming from the pins to detect connected physical modules with sensing ranges and units.

Table 3 Example identifiers required for the detection of the connected physical modules from used six pins of the CAN-15 connector

Identifier	*ISES module*	*Low*	*High*	*Unit*
001001	*capacitometer*	*0*	*10*	*μF*
001011	*voltmeter*	*-5*	*5*	*V*
001110	*manometer*	*0*	*100*	*kPa*

To clarify the identifying process, six used pins of the CAN-37 connector numbered as 4, 5, 9, 10, 11 and 12 corresponds to the bits, which are set, for example, to 0 0 1 1 1 0, what means the ISES manometer is connected.

The obtained (measured) analog values of the respective physical quantity then correspond to the bit combination assigned by the mask part to create the physical module identifier as shown in Table 4.

Table 4 Measured values represented in the bits form and their assignment to the mask parts to create the unique physical modules identifiers

Voltage on pin (V)	*Bit value*	*First part of mask (4, 5, 12)*	*Second part of mask (9, 10, 11)*
5,000	*111*	*11xxx1*	*xx111x*
3,333	*110*	*11xxx0*	*xx011x*
2,500	*101*	*10xxx1*	*xx101x*
2,000	*100*	*10xxx0*	*xx001x*
1,689	*011*	*01xxx1*	*xx110x*
1,445	*010*	*01xxx0*	*xx010x*
1,262	*001*	*00xxx1*	*xx100x*
1,121	*000*	*00xxx0*	*xx000x*

The detection of a physical module is performed separately on every channel according to the following algorithmic steps:

1. *Measurement* - The upcoming signal is measured on the status pins of required channel (A-F) on the CAN-37 connector to obtain the lower and the upper part of the identifying information.
2. *First conversion* - Both the obtained parts are converted (recalculated) to the range from 2048 to 4095 as the WORD numbers.
3. *Second conversion* - Both the WORD numbers are converted to the corresponding three bits (binary digits) form with taking account of predefined tolerances as listed in Table 2.
4. *First assignment* - The two gained binary numbers are assigned to the respective mask as listed in Table 4.
5. *Second assignment* - The built six bits number is finally found in the Module32.map database file to assign and detect the physical module that is attached to the CAN-15 connector.

The PMD includes the detecting functions, which accept the channel numbers as input parameters belonging to the PCI 1202 interface card. They correspond to the appropriate pin on the CAN-37connector as listed in Table 5.

Table 5 Status pins of the physical modules belonging to the CAN-37 connector (values listed in the brackets are the channel numbers accepted by the detection functions as the inputs parameters to generate current status)

Module	Upper part status for pins: 4, 5, 12 on CAN-15	Lower part status for pins: 9, 10, 11 on CAN-15
A	7 (6)	8 (7)
B	9 (8)	10 (9)
C	11 (10)	12 (11)
D	13 (12)	14 (13)
E	15 (14)	16 (15)
F	29 (25)	30 (26)

The PMD is able to provide all the technical parameters of particular physical modules connected to the CAN-15 ports in cooperation with the Module32.map database file. The RL administrators can comfortably monitor the updated list of the physical modules displayed in the concerned GUI containing additional details. If any physical module, defined in the psc file, fails during its operation by the reason of unexpected disconnection or even destruction caused by strokes or vibrations, then the PMD detects this anomaly and dispatches it to the modules list in the GUI and to the DS interface to analyze.

The PMD is consisted of several components deployed on different places of the MS structure and attached plugins. As the main, there is implemented the HwmdsDetectingThread component responsible for the detection process.

When the MS is started up, and the PMD is enabled, afterwards the PMD performs the initial physical modules detection by the member function

```
bool CSinglePluginLibrary::query_detection_info
    (BOOL OnlyDriverDetection,
     BOOL OnlyBoardDetection,
     UINT *pBoardType,
     CString *pConfigName,
     WORD *IdentifiersArray,
     FLOAT *ValuesArray)
```

that returns whether the attached PCI1202CardPlugin.ldp has the detection algorithm implemented in its core. If it is true, then the respective timer is activated to periodically trigger the HwmdsDetectingThread to monitor all the connected physical modules and to generate status information.

6.3.1 Detecting thread component

This component is handled as the function running in a separate tread, and includes the functions directly connected to the PCI1202CardPlugin.ldp to work with the physical modules. As mentioned, it is triggered periodically by a timer located in the timer object together with the MUD's components.

The HwmdsDetectingThread implements the member function

```
bool CMeasureServerDlg::DetectHardwareModules()
```

that calls the query_detection_info function (it is the same function called when the MS starts up). This function belongs to the CSinglePluginLibrary object and directly communicates with the attached ScriptablePlugin2.ldp plugin from which, after conditions checking, it calls the function

```
extern "C" bool __stdcall EXPORT LDPDetectionInfo
    (VOID *const PluginContext,
     BOOL OnlyDriverDetection,
     BOOL OnlyBoardDetection,
     UINT *pBoardType,
     CString *pConfigName,
     WORD *IdentifiersArray,
     FLOAT *ValuesArray)
```

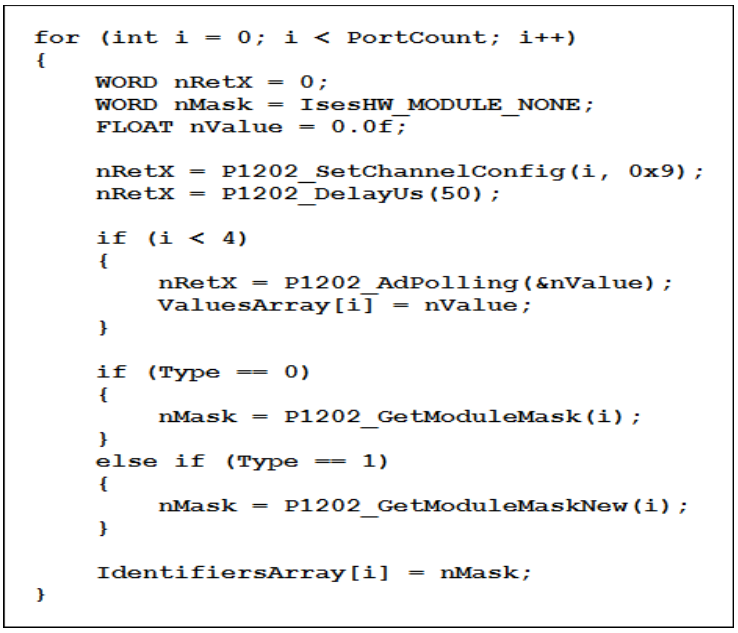
and after several checking operations, it calls the function

```
extern "C" void __stdcall EXPORT LDPDetection
    (VOID *const PluginContext,
     BOOL OnlyBoardDetection,
     UINT *pBoardType,
     WORD *IdentifiersArray,
     FLOAT *ValuesArray)
```

located in the PCI1202CardPlugin.ldp plugin checking the current status of all the CAN-15 ports and updates internal buffers (for five channels A-E, or for six channels A-F). The source code listed in Figure 55 implements the detection algorithm for every port (channel) in the LDPDetection function.

Further important operation is accomplished by the function

```
int CMeasureServerDlg::HardwareModuleActivity
    (WORD &ModuleIdentifier,
     WORD &PrevModuleIdentifier,
     INT Channel)
```



```
for (int i = 0; i < PortCount; i++)
{
    WORD nRetX = 0;
    WORD nMask = IsesHW_MODULE_NONE;
    FLOAT nValue = 0.0f;

    nRetX = P1202_SetChannelConfig(i, 0x9);
    nRetX = P1202_DelayUs(50);

    if (i < 4)
    {
        nRetX = P1202_AdPolling(&nValue);
        ValuesArray[i] = nValue;
    }

    if (Type == 0)
    {
        nMask = P1202_GetModuleMask(i);
    }
    else if (Type == 1)
    {
        nMask = P1202_GetModuleMaskNew(i);
    }

    IdentifiersArray[i] = nMask;
}
```

Figure 55 Source code of the detection algorithm in the LDPDetection function to generate status

that is responsible for a generation of the activity codes from the updated internal buffers, which identify activities on every CAN-15 port as follows:

- *Code 0* - No change,
- *Code 1* - Undefined,
- *Code 2* - Just connected,
- *Code 3* - Just disconnected,
- *Code 4* - Just/still wrong reference,
- *Code 5* - Connected and wrong reference,
- *Code 6* - Disconnected and wrong reference.

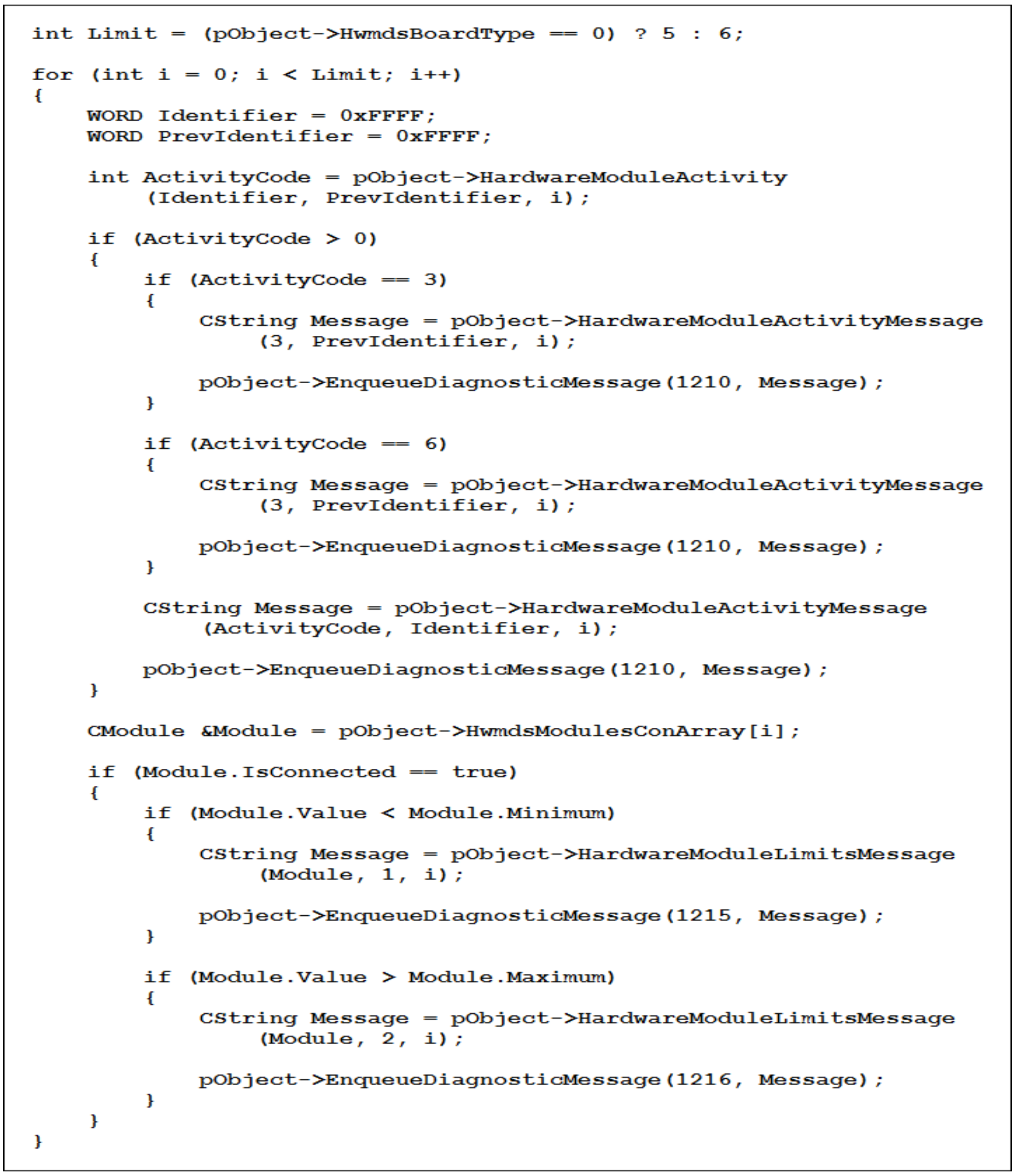
Finally, to complete the detection process, there is called the function

```
CString CMeasureServerDlg::HardwareModuleActivityMessage
    (INT ActivityCode,
     WORD ModuleIdentifier,
     INT Channel)
```

that is constructed to generate and format the diagnostic message reporting about the status of every CAN-15 port after performing one detection cycle. Such the diagnostic message contains whether the port is empty or hosts some physical module. If there is detected a physical module, the diagnostic message provides the engaged port, module name, module identifier, measured unit, measurement range (minimum and maximum) and current status.

Each message indicates one of the status information as: 1) Undefined, 2) Just connected, 3) Just disconnected, 4) Connected with the wrong reference, or 5) Disconnected with the wrong reference, else 6) Detected the empty port.

The source code snippet of the HwmdsDetectingThread component (function) listed in Figure 56 presents the main detecting and reporting process.



```
int Limit = (pObject->HwmdsBoardType == 0) ? 5 : 6;

for (int i = 0; i < Limit; i++)
{
    WORD Identifier = 0xFFFF;
    WORD PrevIdentifier = 0xFFFF;

    int ActivityCode = pObject->HardwareModuleActivity
        (Identifier, PrevIdentifier, i);

    if (ActivityCode > 0)
    {
        if (ActivityCode == 3)
        {
            CString Message = pObject->HardwareModuleActivityMessage
                (3, PrevIdentifier, i);

            pObject->EnqueueDiagnosticMessage(1210, Message);
        }

        if (ActivityCode == 6)
        {
            CString Message = pObject->HardwareModuleActivityMessage
                (3, PrevIdentifier, i);

            pObject->EnqueueDiagnosticMessage(1210, Message);
        }

        CString Message = pObject->HardwareModuleActivityMessage
            (ActivityCode, Identifier, i);

        pObject->EnqueueDiagnosticMessage(1210, Message);
    }

    CModule &Module = pObject->HwmdsModulesConArray[i];

    if (Module.IsConnected == true)
    {
        if (Module.Value < Module.Minimum)
        {
            CString Message = pObject->HardwareModuleLimitsMessage
                (Module, 1, i);

            pObject->EnqueueDiagnosticMessage(1215, Message);
        }

        if (Module.Value > Module.Maximum)
        {
            CString Message = pObject->HardwareModuleLimitsMessage
                (Module, 2, i);

            pObject->EnqueueDiagnosticMessage(1216, Message);
        }
    }
}
```

Figure 56 Source code snippet of the HwmdsDetectingThread component to detect the physical modules and the generation of diagnostic messages

The generated and formatted diagnostic messages are lastly dispatched in sequence to the DS to store in its local database for the analysis.

Except the mentioned periodical detection process, there is also possible to dispatch the command *detect* (optionally accompanied by an IP address) from the RL administrator to obtain the current status list of the CAN-15 ports. This command is processed in the RlmsOperatingThread that calls the function

```
CString CMeasureServerDlg::GetHardwareModuleStatus
    (INT Channel)
```

to provide the diagnostic message for every CAN-15 port that comprises the used port, module name, module identifier, measured unit, measurement range and current value. As an example, when the RL administrator dispatches the command "detect 10.20.54.66", then there is triggered the action of the physical modules detection in the RlmsOperatingThread as shown in Figure 57.

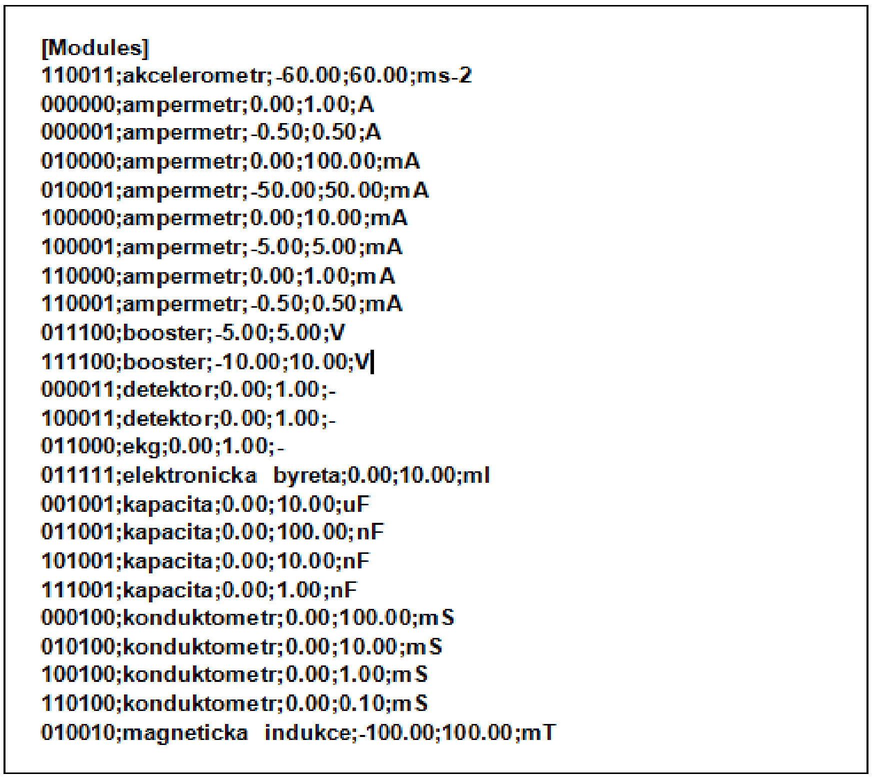
```
if ((Message.Find("detect") != -1) && (Message.Find(IPAddress) != -1))
{
    pObject->KillTimer(pObject->HwmdsDetectionTimerID);

    pObject->pClientThread->DispatchTextMessage
         (GetDiagnosticMessage(1203,
         "Hardware modules detection updated"));

    if (pObject->HwmdsPresent == TRUE)
    {
        if (pObject->HwmdsEnable == TRUE)
        {
            int Limit = (pObject->HwmdsBoardType == 0) ? 5 : 6;

            for (int i = 0; i < Limit; i++)
            {
                pObject->pClientThread->DispatchTextMessage
                     (GetDiagnosticMessage(1210,
                     pObject->GetHardwareModuleStatus(i)));
            }
        }
        else
        {
            pObject->pClientThread->DispatchTextMessage
                 (GetDiagnosticMessage(1201,
                 "Hardware modules detection disabled"));
        }
    }
    else
    {
        pObject->pClientThread->DispatchTextMessage
             (GetDiagnosticMessage(1204,
             "Hardware modules detection not present"));
    }

    pObject->HwmdsSetTimer();
}
```

Figure 57 Source code of the "detect" command triggered by the RL administrator to perform a detection of the connected physical modules and to provide the current status list

The PMD is designed to identify the detected physical modules by using the reference list of all the ISES modules, which can be connected to the system. This concept is important because the LDPDetection function generates and returns the 6-bit identifiers only, which are then associated with more details about detected physical modules, like their entire names, measurement ranges and units. This assigning operation is accomplished (based on the query and the timer) in the components RlmsOperatingThread and HwmdsDetectingThread. These physical modules details (records) are stored in the Modules.txt file that is loaded into the internal buffer inside the MS during its startup. The file currently contains 58 records and it can be extended, if required to add a new physical module. Each stored record has the following items and sequence:

6-bit identifier; module name; minimum; maximum; unit.

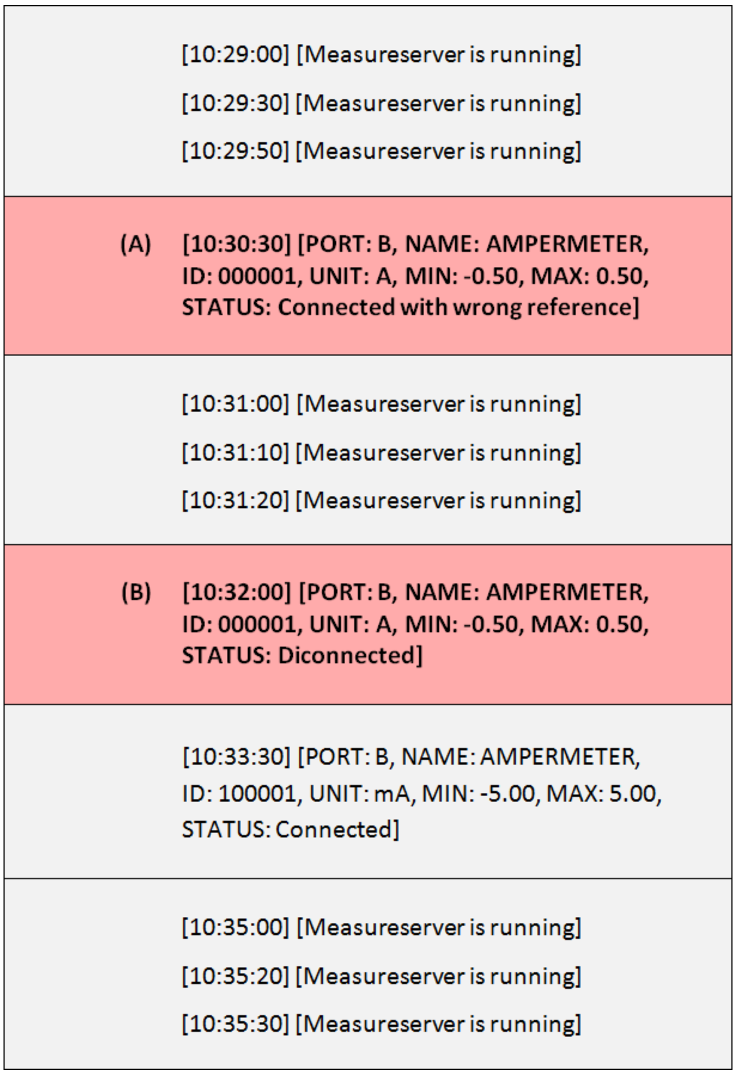
The list presented in Figure 58 shows a piece of the Modules.txt file that is parsed by the MS to make module references for the detection process.



```
[Modules]
110011;akcelerometr;-60.00;60.00;ms-2
000000;ampermetr;0.00;1.00;A
000001;ampermetr;-0.50;0.50;A
010000;ampermetr;0.00;100.00;mA
010001;ampermetr;-50.00;50.00;mA
100000;ampermetr;0.00;10.00;mA
100001;ampermetr;-5.00;5.00;mA
110000;ampermetr;0.00;1.00;mA
110001;ampermetr;-0.50;0.50;mA
011100;booster;-5.00;5.00;V
111100;booster;-10.00;10.00;V|
000011;detektor;0.00;1.00;-
100011;detektor;0.00;1.00;-
011000;ekg;0.00;1.00;-
011111;elektronicka  byreta;0.00;10.00;ml
001001;kapacita;0.00;10.00;uF
011001;kapacita;0.00;100.00;nF
101001;kapacita;0.00;10.00;nF
111001;kapacita;0.00;1.00;nF
000100;konduktometr;0.00;100.00;mS
010100;konduktometr;0.00;10.00;mS
100100;konduktometr;0.00;1.00;mS
110100;konduktometr;0.00;0.10;mS
010010;magneticka  indukce;-100.00;100.00;mT
```

Figure 58 Exemplary piece of the physical modules list stored in the Modules.txt file used for the detection

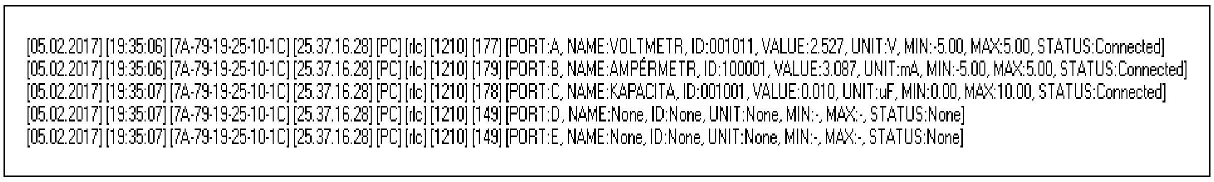
For example, the MUD dispatcher generates and delivers to the DS user interface the periodical notifications related to the MS running and the status messages about the connected physical module as listed in Figure 59.



```
[10:29:00] [Measureserver is running]
[10:29:30] [Measureserver is running]
[10:29:50] [Measureserver is running]

(A)  [10:30:30] [PORT: B, NAME: AMPERMETER,
     ID: 000001, UNIT: A, MIN: -0.50, MAX: 0.50,
     STATUS: Connected with wrong reference]

[10:31:00] [Measureserver is running]
[10:31:10] [Measureserver is running]
[10:31:20] [Measureserver is running]

(B)  [10:32:00] [PORT: B, NAME: AMPERMETER,
     ID: 000001, UNIT: A, MIN: -0.50, MAX: 0.50,
     STATUS: Diconnected]

[10:33:30] [PORT: B, NAME: AMPERMETER,
ID: 100001, UNIT: mA, MIN: -5.00, MAX: 5.00,
STATUS: Connected]

[10:35:00] [Measureserver is running]
[10:35:20] [Measureserver is running]
[10:35:30] [Measureserver is running]
```

Figure 59 Example of the MUD records; correctly running (gray), two failed (red); A) setup changed caused by the range of amperemeter, and B) one amperemeter was disconnected

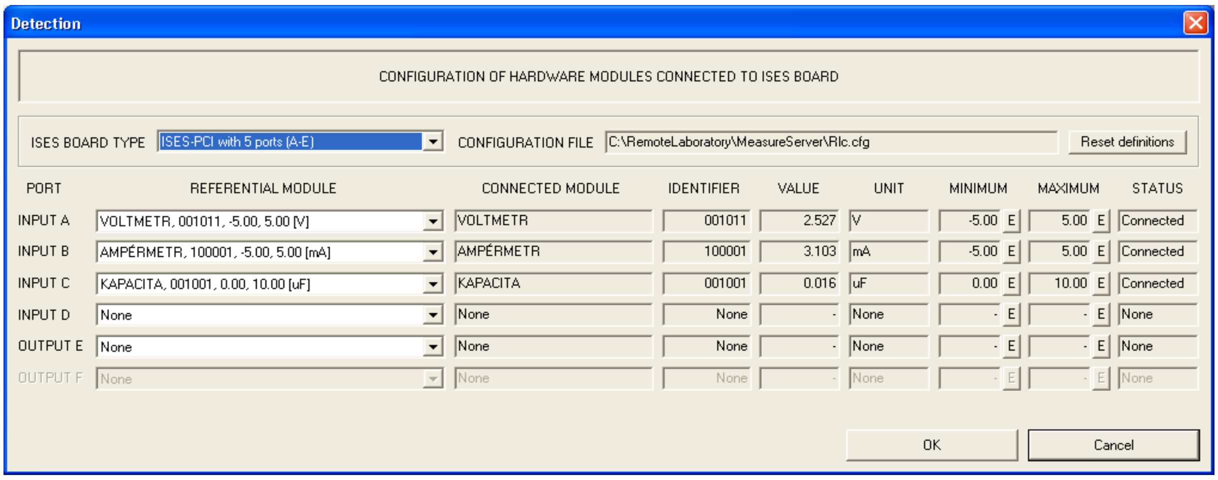
As the next example, if the RL administrator demands the current status of connected physical modules, then he must enter and dispatch the command “detect 10.20.54.66” to the MS unit. After a short time, the MS’s PMD returns the complete list to the DS user interface as shown in Figure 60.



```
[05.02.2017] [19:35:06] [7A-79-19-25-10-1C] [25.37.16.28] [PC] [rlc] [1210] [177] [PORT:A, NAME:VOLTMETR, ID:001011, VALUE:2.527, UNIT:V, MIN:-5.00, MAX:5.00, STATUS:Connected]
[05.02.2017] [19:35:06] [7A-79-19-25-10-1C] [25.37.16.28] [PC] [rlc] [1210] [179] [PORT:B, NAME:AMPÉRMETR, ID:100001, VALUE:3.087, UNIT:mA, MIN:-5.00, MAX:5.00, STATUS:Connected]
[05.02.2017] [19:35:07] [7A-79-19-25-10-1C] [25.37.16.28] [PC] [rlc] [1210] [178] [PORT:C, NAME:KAPACITA, ID:001001, VALUE:0.010, UNIT:uF, MIN:0.00, MAX:10.00, STATUS:Connected]
[05.02.2017] [19:35:07] [7A-79-19-25-10-1C] [25.37.16.28] [PC] [rlc] [1210] [149] [PORT:D, NAME:None, ID:None, UNIT:None, MIN:-, MAX:-, STATUS:None]
[05.02.2017] [19:35:07] [7A-79-19-25-10-1C] [25.37.16.28] [PC] [rlc] [1210] [149] [PORT:E, NAME:None, ID:None, UNIT:None, MIN:-, MAX:-, STATUS:None]
```

Figure 60 Example of the detailed list presenting the current status of connected physical modules to the respective ports on the ISES PCI control board

The MS provides the RL administrator the PMD user interface that allows monitoring the complete status of the CAN-15 ports. There is an option to preset the referential physical modules and to change the measurement ranges. The configuration and changes are stored to the specific diagnostic file. The user interface presenting the exemplary settings is shown in Figure 61.



Detection

CONFIGURATION OF HARDWARE MODULES CONNECTED TO ISES BOARD

ISES BOARD TYPE: ISES-PCI with 5 ports (A-E)

CONFIGURATION FILE: C:\RemoteLaboratory\MeasureServer\Rlc.cfg

PORT	REFERENTIAL MODULE	CONNECTED MODULE	IDENTIFIER	VALUE	UNIT	MINIMUM	MAXIMUM	STATUS
INPUT A	VOLTMETR, 001011, -5.00, 5.00 [V]	VOLTMETR	001011	2.527	V	-5.00	5.00	Connected
INPUT B	AMPÉRMETR, 100001, -5.00, 5.00 [mA]	AMPÉRMETR	100001	3.103	mA	-5.00	5.00	Connected
INPUT C	KAPACITA, 001001, 0.00, 10.00 [uF]	KAPACITA	001001	0.016	uF	0.00	10.00	Connected
INPUT D	None	None	None	-	None	-	-	None
OUTPUT E	None	None	None	-	None	-	-	None
OUTPUT F	None	None	None	-	None	-	-	None

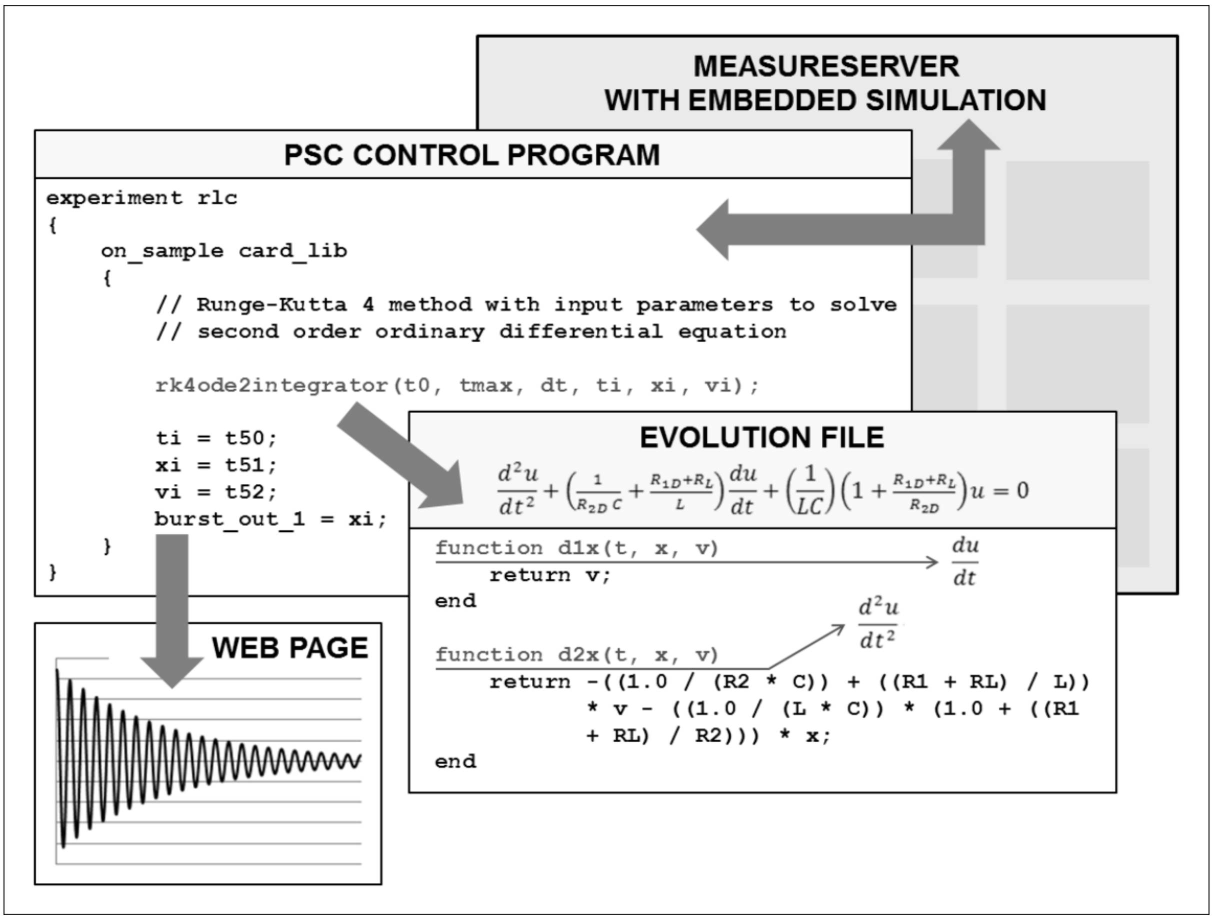
Figure 61 Example settings of the PMD user interface to present the involved referential physical modules and detected information with current status

Summary of the subchapter "6.3 Second level diagnostics module": This subchapter deals with the goal No. 4. The diagnostics of remote laboratories on this level is not to be found, to our knowledge, in any published laboratory, irrespective of the fact of its importance. The second level diagnostics serves to check the conformity of the physical hardware compared with the standard setup. The checking results, when providing negative results, are forwarded to the remote laboratory provider to recover experiment. This constitutes the improvement in the intelligent version of the Measureserver unit. The novelty and the scientific approach rest in the use of guarding all parameters of physical modules as the function, its range, polarity and signal overload.

6.4 Embedded real-world phenomena simulation module

Simulations play an increasingly important role in the way of teaching or doing science. This is especially true in education, where computers are being used more often as a way to make lectures more attractive to users, and to help them achieve deeper understanding of the subject being taught.

This subchapter deals with the design and implementation of the module called EPS (embedded real-world phenomena simulation) as the last task of the doctoral thesis. The EPS components are described for more detail in the next sections to elucidate its concept shown in Figure 62 and functioning.



```
experiment rlc
{
    on_sample card_lib
    {
        // Runge-Kutta 4 method with input parameters to solve
        // second order ordinary differential equation

        rk4ode2integrator(t0, tmax, dt, ti, xi, vi);

        ti = t50;
        xi = t51;
        vi = t52;
        burst_out_1 = xi;
    }
}
```

$$\frac{d^2u}{dt^2}+\left(\frac{1}{R_{2D}C}+\frac{R_{1D}+R_L}{L}\right)\frac{du}{dt}+\left(\frac{1}{LC}\right)\left(1+\frac{R_{1D}+R_L}{R_{2D}}\right)u=0$$

```
function d1x(t, x, v)
    return v;
end

function d2x(t, x, v)
    return -((1.0 / (R2 * C)) + ((R1 + RL) / L))
           * v - ((1.0 / (L * C)) * (1.0 + ((R1
           + RL) / R2))) * x;
end
```

$$\frac{du}{dt}$$

$$\frac{d^2u}{dt^2}$$

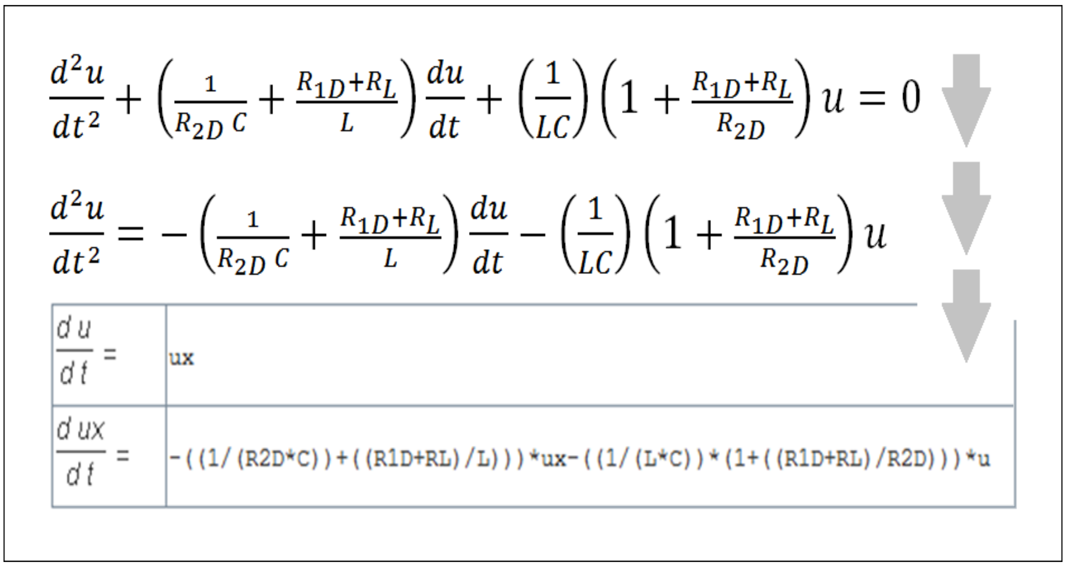
Figure 62 Concept of the EPS module that is implemented in four different components; 1) the MS unit to host this module, 2) psc control file to place the respective integrator, 3) evolution file to calculate the rate, 4) web page on the client's side to visualize the simulation process

The EPS can run concurrently together with the real experiment on the client's web page. It is the optional module that can be disabled, if not needed, in the psc script by using the specific variable.

The MS can use basic mathematical functions, like a sine, cosine, arc sine and arc cosine, which are supplied by the ScriptablePlugin2.ldp to use then for internal calculations or for displaying on the client's web page. The EPS extends the MS's library of mathematical functions. It provides the RL designers advanced numerical function as the solvers to deal with ordinary differential equations (ODEs). There is created the EPS interface to parse and process the first and second order ODE in the analytical form [46] to calculate its solution through the specific iterative functions placed in the psc script.

The EPS interface was designed to exploit the Lua programming language to simply enter (specify) the ODE analytical form coded and saved to the plain text file, called Simulation.lua as the input setting for the solver. The Lua is a lightweight multi-paradigm programming language that is primarily intended for embedded systems and clients. The Lua is cross-platform, since it is written in the ANSI C, and has a relatively simple C API. It is a language for extending software applications to meet the increasing demand for customization at the time. This programming concept provides the basic facilities of most procedural programming languages, but more complicated or domain-specific features were not included; rather, it includes mechanisms for extending the language that allows programmers to implement such features. The Lua designers focus on the improving its speed, portability, extensibility and ease of use [47].

The initial idea of ODEs entering and passing the solver was taken from the Easy Java/JavaScript Simulations (EjsS) environment. At a glance, the EjsS is a free authoring tool written in the Java programming language that helps involved non-programmers to design and create interactive simulations in the Java or Javascript, mainly for the teaching or learning purposes. The EjsS has been developed by Francisco Esquembre and is a part of the Open Source Physics project [48]. As the example, based on this idea, the analytical form of second order ODE can be adjusted (algebraic expression editing) and entered on separate lines to process by the solver as shown in Figure 63.



$$\frac{d^2u}{dt^2} + \left(\frac{1}{R_{2D}\,C} + \frac{R_{1D}+R_L}{L}\right)\frac{du}{dt} + \left(\frac{1}{LC}\right)\left(1 + \frac{R_{1D}+R_L}{R_{2D}}\right)u = 0$$

$$\frac{d^2u}{dt^2} = -\left(\frac{1}{R_{2D}\,C} + \frac{R_{1D}+R_L}{L}\right)\frac{du}{dt} - \left(\frac{1}{LC}\right)\left(1 + \frac{R_{1D}+R_L}{R_{2D}}\right)u$$

$\frac{d\,u}{d\,t} =$	`ux`
$\frac{d\,ux}{d\,t} =$	`-((1/(R2D*C))+((R1D+RL)/L)))*ux-((1/(L*C))*(1+((R1D+RL)/R2D)))*u`

Figure 63 Adjustment and entry of the second order ODE to a readable form processed by the EjsS as the basic idea used for entering such the mathematical expressions to the EPS interface

The Simulation.lua file contains input parameters (coefficients), which are declared in the upper part as local variables (including their initializations) with unique 8-bit identifiers to distinguish particular coefficients by the EPS handling them from the MS core. These coefficients affect the ODE solution during the simulation. Further definitions represent the first and second derivative of a function of a real variable (rate) built as the following functions:

1. `function d1x(t, x, v)` - returns the value of first derivative,
2. `function d2x(t, x, v)` - returns the value of second derivative.

These two functions are involved at the most in the file because the EPS is able to handle and solve first and second order ODEs only. It means, when the second order ODE is entered and validated, the d1x function is called as the first to return the value of the first derivative. This resulting value can be inserted to the d2x function as a part of the rate (right side expression) to calculate and return the value of the second derivative. Both the functions are called in a loop based on the maximum simulation time and time step defined in the psc script to obtain the solution (evolution) for the given time. The example, based on the introduced idea from the EjsS environment, presents the analytical form of the second order ODE (Figure 63) that is manually coded in the Simulation.lua file constituted as the evolution component shown in Figure 64.

```
local R1 = 0            -- #10000000
local R2 = 100000000    -- #11000000
local RL = 60           -- #11100000
local L  = 0.910000     -- #11110000
local C  = 0.000001     -- #11111000

function d1x(t, x, v)

    return v;

end

function d2x(t, x, v)

    return -((1.0 / (R2 * C)) + ((R1 + RL) / L)) * v -
            ((1.0 / (L * C)) * (1.0 + ((R1 + RL) / R2))) * x;

end
```

Figure 64 Example of the Simulation.lua file that contains the declared and initialized coefficients R_1, R_2, R_L, L and C, and the function dx1 and dx2 taking the second order ODE to calculate the rate

The used second order ODE (Figure 63) is constructed from the Kirchhoff's voltage law. The file header section contains a declaration of the required coefficients (R_1, R_2, R_L, L and C) accompanied by their unique 8-bit identifiers (for example, #10000000), which are engaged as the actuating instructions used by the EPS algorithms to set/change demanded values in respective variables. The dx1 function is used to return the real value of the variable *v* after its differentiation. Afterwards, the dx2 function performs a differentiation of the variable *x* and the variable *v* (again, to reach the second derivative) and then returns the current damped voltage at time *t* incremented in the psc script.

The EPS handles the Simulation.lua file in four objects to read declared coefficients and to initialize and run the dx1 and dx2 functions. The objects represent the numerical integration functions (integrators) used to implement the simulation process by manual coding in the psc script; these objects are:

1. ScriptMEODE1Integrator,
2. ScriptMEODE2Integrator,
3. ScriptRK4ODE1Integrator,
4. ScriptRK4ODE2Integrator.

The listed objects are divided into two groups based on the applied numerical methods called Modified Euler and Fourth Order Runge-Kutta. They are used to solve the first and second order ODEs. The mentioned order is indicated by the number 1 or 2 contained in the object's name (ODE1 and ODE2). These two groups including their numerical methods and function identifiers applicable in the psc script are described in the following subchapters.

6.4.1 Modified Euler integrator

It is the ScriptMEODExIntegrator component that represents two integrators using the Modified Euler method. This is not so precise method (less suitable for the simulation) but it was implemented because of the didactical purposes.

The Modified Euler algorithm

This method provides the approximate numerical solution of the initial value problem $\frac{dy}{dx} = f(x, y)$ subject to the initial condition $y(x_0) = y_0$ with the step size h that is obtained from the algorithm

$$y_{n+1} = y_n + \frac{1}{2}h[\,f(x_n, y_n) + f\big(x_n + h,\, y_n + hf(x_n, y_n)\big)\,] \qquad (1)$$

for $n = 1, 2, \ldots$, where $x_n = x_0 + nh$.

The estimate of y_{n+1} is performed by the two-stage algorithm implemented as predictor-corrector; first step is the predictor

$$y^*_{n+1} = y_n + hf(x_n, y_n) \tag{2}$$

and second step is the corrector

$$y_{n+1} = y_n + \frac{1}{2}h[\,f(x_n,\, y_n) + f(x_{n+1}, y^*_{n+1})\,]\,. \tag{3}$$

The first step gives the estimate of y' at x_n . This action is used to predict the first guess y^*_{n+1} that is calculated to obtain an estimate of the average y' over the interval h. The second step then corrects the estimate of y_{n+1} [49].

The Modified Euler integrators are exposed in the psc script as functions with parameters; the first function represents the first order ODE integrator

```
void meode1integrator
    (DOUBLE t0,
     DOUBLE tmax,
     DOUBLE dt,
     DOUBLE ti,
     DOUBLE xi,
     DOUBLE &tf,
     DOUBLE &xf)
```

and the second function implements the second order ODE integrator

```
void meode2integrator
    (DOUBLE t0,
     DOUBLE tmax,
     DOUBLE dt,
     DOUBLE ti,
     DOUBLE xi,
     DOUBLE vi,
     DOUBLE &tf,
     DOUBLE &xf,
     DOUBLE &vf).
```

The ScriptMEODE2Integrator object that allows using the meode2integrator function in the psc script is introduced in Figure 65. There is exposed the source code snippet of the Modified Euler algorithm as the predictor and corrector.

```
double h, t;

h = dt;
t = ti + h;

//Predictor
xf = xi + d1x(ti, xi, vi) * (tf - ti);
vf = vi + d2x(ti, xi, vi) * (tf - ti);

//Corrector
xf = xi + (d1x(ti, xi, vi) + d1x(ti, xf, vf)) * 0.5 * (tf - ti);
vf = vi + (d2x(ti, xi, vi) + d2x(ti, xf, vf)) * 0.5 * (tf - ti);
```

Figure 65 Source code of the ScriptMEODE2Integrator object that implements the Modified Euler algorithm to obtain the solution; the second order ODE is entered in the function dx1 and dx2 in the Simulation.lua file, the variables

6.4.2 Runge-Kutta integrator

It is the ScriptRK4ODExIntegrator component that represents two integrators using the Fourth Order Runge-Kutta method (also called Runge-Kutta 4). This method produces a better solution in fewer steps. The numerical method is used deliberately because it is suitable for the simulation process.

The Runge-Kutta 4 algorithm

This method gives the approximate numerical solution of the initial value problem $\frac{dy}{dx} = f(x,y)$ subject to the initial condition $y(x_0) = y_0$ with the step length h that is obtained from the following algorithm divided into three steps, with $x_n = x_0 + nh$ and $x_n = x_0 + nh$.

First step: Calculate

$$x_{1n} = hf(x_n, y_n) \tag{4}$$

$$x_{2n} = hf\left(x_n + \frac{1}{2}h,\ y_n + \frac{1}{2}k_{1n}\right) \tag{5}$$

$$x_{3n} = hf\left(x_n + \frac{1}{2}h,\ y_n + \frac{1}{2}k_{2n}\right) \tag{6}$$

$$x_{4n} = hf(x_n + h,\ y_n + k_{3n}). \tag{7}$$

Second step: Calculate

$$d_n = \frac{1}{6}(k_{1n} + 2k_{2k} + 2k_{3n} + k_{4n}). \tag{8}$$

Third step: The numerical approximation y_{n+1} of the solution $y = y(x_{n+1})$ is given by the expression

$$y_{n+1} = y_n + d_n, \tag{9}$$

for $n = 1, 2, \ldots$ [50].

The Runge-Kutta 4 integrators are also exposed in the psc script as functions with parameters; the first function constitutes the first order ODE integrator

```
void rk4ode1integrator
    (DOUBLE t0,
     DOUBLE tmax,
     DOUBLE dt,
     DOUBLE ti,
     DOUBLE xi,
     DOUBLE &tf,
     DOUBLE &xf)
```

and the second function provides the second order ODE integrator

```
void rk4ode1integrator
    (DOUBLE t0,
     DOUBLE tmax,
     DOUBLE dt,
     DOUBLE ti,
     DOUBLE xi,
     DOUBLE vi,
     DOUBLE &tf,
     DOUBLE &xf,
     DOUBLE &vf).
```

Both the integrators are widely used because they yield satisfactory results as mentioned. The ScriptRK4ODE2Integrator object that implements and exposes the rk4ode2integrator function in the psc script is presented in Figure 66. There is showed the source code snippet of the Runge-Kutta 4 algorithm for second order ODE to calculate the approximate numerical solution.

There is important to add the information about executing the d1x and d2x functions, which are bound to their namesakes in the Simulation.lua file used by every integrator. As cleared up, both the functions (Figure 64) contain the ODE analytical form to calculate the rate at time *t*, and return respective results to calling functions. As an example to understand the process of calling and executing the dx1 function (for the dx2 function, it is the same principle), located in the Simulation.lua file, is performed in the ScriptRK4ODE2Integrator object by the rk4ode1integrator function as shown in Figure 67.

```
double d1x(double t, double x, double v) const
{
    //Function name
    lua_getglobal(LuaObject, "d1x");

    //First function argument
    lua_pushnumber(LuaObject, t);

    //Second function argument
    lua_pushnumber(LuaObject, x);

    //Third function argument
    lua_pushnumber(LuaObject, v);

    if (lua_pcall(LuaObject, 3, 1, 0) != 0)
    {
        DisabledRK4ODE2Integrator = true;
        return 0;
    }

    double result = (double)lua_tonumber(LuaObject, -1);

    lua_pop(LuaObject, 1);

    return result;
}
```

Figure 66 Source code of the dx1 function that is implemented in the ScriptRK4ODE2Integrator object to call the Lua functions for pushing particular parameters to the dx1 function and calling it in the Simulation.lua file with the rate return

```
double h, t, k1x, k2x, k3x, k4x, k1v, k2v, k3v, k4v;

h = dt;
t = ti + h;

k1x = h * d1x(t - t0, xi, vi);
k1v = h * d2x(t - t0, xi, vi);

k2x = h * d1x((t - t0) + h / 2.0, xi + k1x / 2.0, vi + k1v / 2.0);
k2v = h * d2x((t - t0) + h / 2.0, xi + k1x / 2.0, vi + k1v / 2.0);

k3x = h * d1x((t - t0) + h / 2.0, xi + k2x / 2.0, vi + k2v / 2.0);
k3v = h * d2x((t - t0) + h / 2.0, xi + k2x / 2.0, vi + k2v / 2.0);

k4x = h * d1x((t - t0) + h, xi + k3x, vi + k3v);
k4v = h * d2x((t - t0) + h, xi + k3x, vi + k3v);

xf = xi + (k1x + 2.0 * (k2x + k3x) + k4x) / 6.0;
vf = vi + (k1v + 2.0 * (k2v + k3v) + k4v) / 6.0;
```

Figure 67 Source code of the ScriptRK4ODE2Integrator object that gives shape to the Runge-Kutta 4 algorithm to obtain the solution; the second order ODE is entered in the function dx1 and dx2 in the Simulation.lua file, the variables t0, tmax, dt, ti, xi, vi, xf and vf come from the rk4ode1integrator function

6.4.3 Integrator parameters component

The EPS also handles the Simulation.lua file for the purpose of modifying declared coefficients in its header section by using the ScriptIntegratorParameter object that is aimed at exposing the function

```
void integratorparameter
    (DOBULE id,
     DOUBLE number)
```

that is called from the psc script to set a new value to the coefficient with the assigned unique identifier (for example, #10000000) formed as 8-bit number. The example shown in Figure 68 presents the algorithm to find the input unique identifier in the Simulation.lua file. If the identifier is found, then its value is replaced by the new one in respective number format.

```
std::string Identifier = Format("#%d"), (int)id);

bool Detected = false;

for (int i = 0; i < Lines.size(); i++)
{
    std::size_t Found = Lines[i].find(Identifier);

    if (Found != std::string::npos)
    {
        std::size_t position = Lines[i].find("=");
        std::string variable = Lines[i].substr(0, position + 1);

        double Integer = 0.0;
        double Fraction = 0.0;

        Fraction = modf(number, &Integer);

        if (Fraction > 0)
        {
            Lines[i] = Format("%s %lf -- %s\n", variable.c_str(),
                                  number, Identifier.c_str());
        }
        else
        {
            Lines[i] = Format("%s %0.0lf -- %s\n", variable.c_str(),
                                  number, Identifier.c_str());
        }

        Detected = true;
    }
}
```

Figure 68 Source code of the ScriptIntegratorParameter object that implements the algorithm aimed at finding the input unique identifier in the Simulation.lua file and at replacing its value by the new one if the identifier is found

The RL designer can use the described functions in the psc script to create the simulation running together with the real process. The example code introduced in Figure 69 presents the psc script snippet that points to the integratorparameter and rk4ode1integrator functions implemented in respective sections (pin_write and experiment). The coded structure constitutes the fast experiment including three internal blocks (init, on_sample and finalize). The simulation fragments are marked by the red text to read better the source code.

```
pin_write input_sim_L
{
    sim_L = new_value;
    integratorparameter(11110000, simu_L);
}

pin_write input_sim_C
{
    sim_C = new_value;
    integratorparameter(11111000, sim_C);
}

experiment fast
{
    init
    {
        switch = 1;
    }

    on_sample card_lib
    {
        #Measured voltage
        burst_out_0 = burst_in_0;

        #Simulated voltage
        rk4ode2integrator(sim_t0, sim_tmax, sim_dt, sim_ti, sim_xi, sim_vi);

        sim_ti = t50;
        sim_xi = t51;
        sim_vi = t52;

        burst_out_1 = sim_xi;
    }

    finalize
    {
        switch = 0;
    }
}
```

Figure 69 Source code of the psc script that presents the simulation; the "pin_write" function modifies the variables named "sim_L" and "sim_C" in the psc script and the variables identified as "11110000" and "11111000" in the Simulation.lua file to set them by the integratorparameter function. The "experiment" section is the body of the fast RL to measure voltage by the attached volt-meter and to simulate the voltage by the rk4ode1integrator function with parameters

The results coming from the simulation process are visualized on the client's web page by specific widgets like charts and data tables. There are also available the widgets to set and change coefficients involved in the simulation by using buttons and sliders. The example presenting a snippet of the source code in JavaScript is listed in Figure 70 to implement the widgets on the web page.

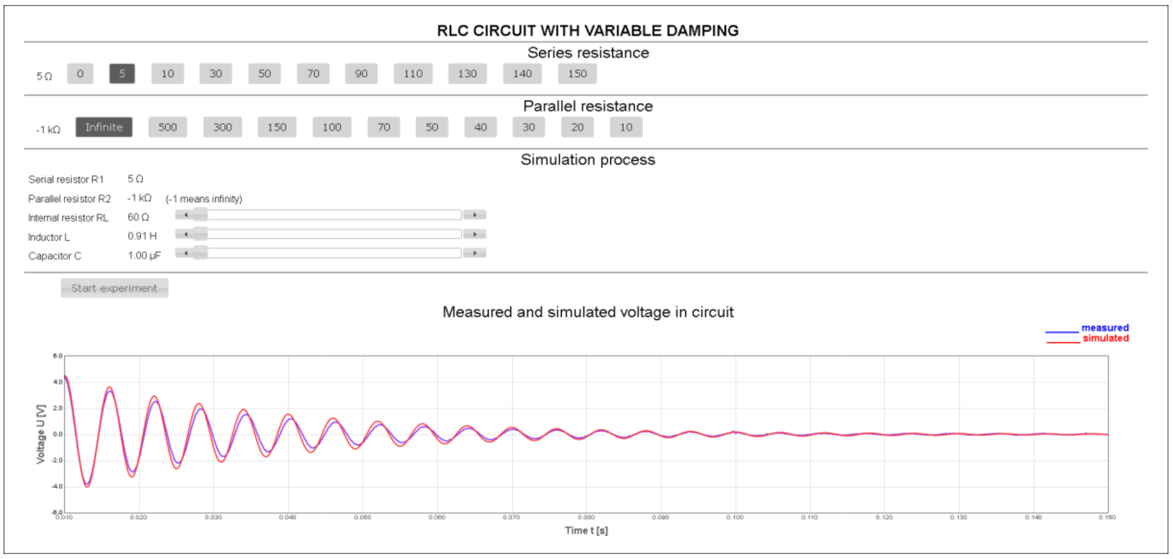
```
$("#display_simulation_L").VLValueDisplay({
      Devices: [  { deviceName: "rlc_output_simulation_L" } ],
      Refresh: 500,
      Format: "0.00 H"
});
```

```
$("#slider_simulation_L").VLSlider({
      devices: [ { deviceName: "rlc_input_simulation_L" } ],
      min: 0.91,
      max: 5.0,
      select_step: 0.1,
      button_step: 0.1,
      refresh: 200,
});
```

```
$("#experiment_graph").VLChart({
      devices: [{
             deviceName: "burst_1",
             loadMethod: "Experiment",
             slot: "experimentId",
             sMin: -4.5,
             sMax: 4.5,
             dMin: -4.5,
             dMax: 4.5
      }],
      ...
});
```

Figure 70 Example of the source code snippet in JavaScript implementing three widgets (red identifiers with their devices): 1) "display_simulation_L" displays a current value of the inductor, 2) "slider_simulation_L" sets a new value of the inductor, 3) "experiment_graph" plots the simulated voltage in the chart

As the example, the visualization of the embedded simulation, provided by the EPS module, is presented on the web page as shown in Figure 71.


RLC CIRCUIT WITH VARIABLE DAMPING

Series resistance

5 Ω [0] [✓ 5] [10] [30] [50] [70] [90] [110] [130] [140] [150]

Parallel resistance

-1 kΩ [✓ Infinite] [500] [300] [150] [100] [70] [50] [40] [30] [20] [10]

Simulation process

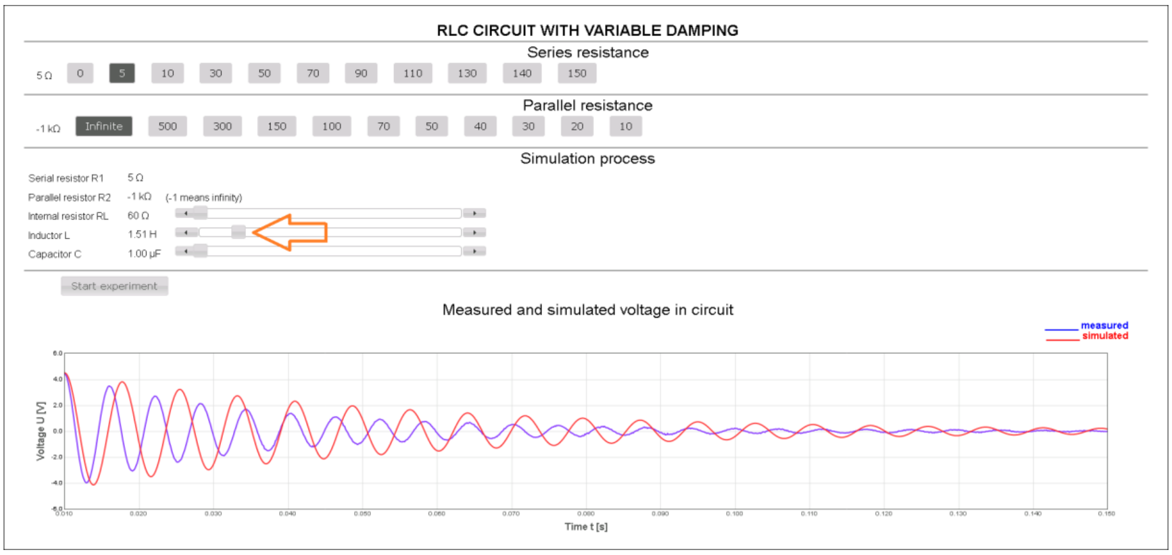
Serial resistor R1	5 Ω	
Parallel resistor R2	-1 kΩ	(-1 means infinity)
Internal resistor RL	60 Ω	
Inductor L	0.91 H	
Capacitor C	1.00 µF	

Start experiment

Measured and simulated voltage in circuit

Figure 71 Example of the visualized embedded simulation provided by the EPS module that is integrated into the MS unit; it presents the RL "The electric and electromagnetic phenomena in the RLC circuit with the variable damping" where two colored curves are displayed; 1) blue: measured voltage, and 2) red: simulated voltage; both the approaches have the same values of the input parameters R_1, R_2, R_L, L and C; curves are similar

The next example represents the introduced RL, but it slightly differs in the setup of the simulated process as displayed in Figure 72.


RLC CIRCUIT WITH VARIABLE DAMPING

Series resistance: 5 Ω

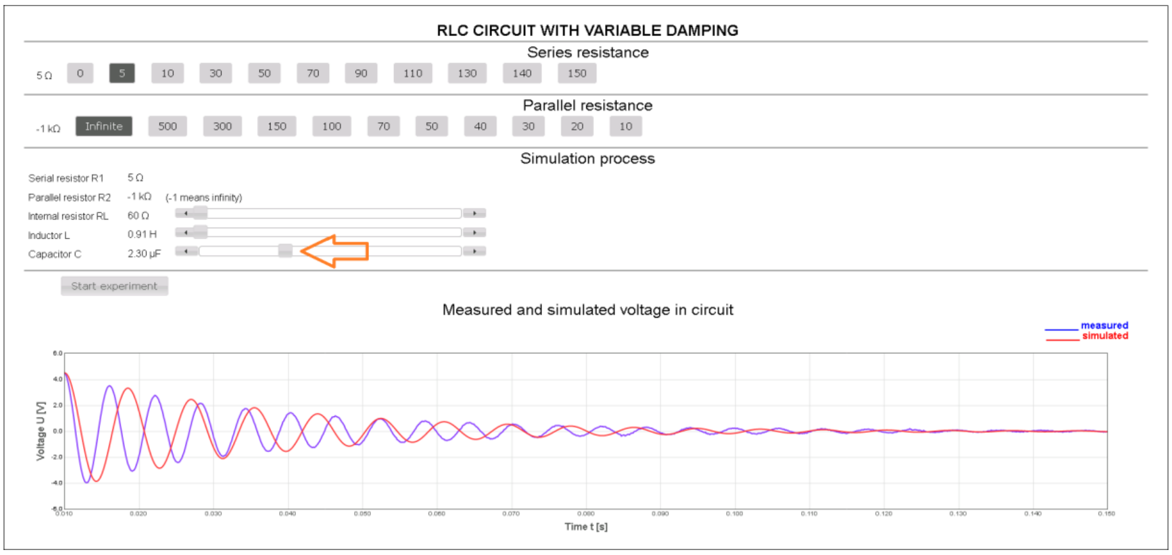
Parallel resistance: -1 kΩ

Simulation process

Serial resistor R1	5 Ω	
Parallel resistor R2	-1 kΩ	(-1 means infinity)
Internal resistor RL	60 Ω	
Inductor L	1.51 H	
Capacitor C	1.00 µF	

Figure 72 Example of the visualized embedded simulation; the L coefficient differs as the orange arrow points out; the curves have unmatched behavior

The last example concerning the mentioned RL demonstrates another coefficient adjustment of the simulated process as shown in Figure 73.


RLC CIRCUIT WITH VARIABLE DAMPING

Series resistance: 5 Ω

Parallel resistance: -1 kΩ

Simulation process

Serial resistor R1	5 Ω	
Parallel resistor R2	-1 kΩ	(-1 means infinity)
Internal resistor RL	60 Ω	
Inductor L	0.91 H	
Capacitor C	2.30 µF	

Figure 73 Example of the visualized embedded simulation; the C coefficient differs as the orange arrow points out; the curves have unmatched behavior

The simulation functions can also be deployed in the slow RLs inside the sections characteristic for this kind of the experiment concept. Furthermore, the simulated data are coupled with the measured data and archived to xml files, and finally dispatched to the data warehouse for detailed analyses.

Summary of the subchapter "6.4 Embedded simulation module of real-world phenomena": This subchapter deals with the goal No. 5. The multi-parameter simulation synchronized with the real measured phenomenon is a tool of utmost importance, not to be found in published remote laboratories. For the purpose, the solver, including ordinary differential equations of basic mathematical function has to be built. Clients may compare the measured results with theoretical equations and adjust their parameters to learn their impact on final results, and to find their correct set. The simulation may be used as a motivation tool independently before real measurement as an introductory step for the acquaintance with measured phenomenon. This improvement is a major contribution to the intelligent version of the Measureserver unit resting in the fact of the necessity of sharing priorities of two processes. The novelty and the scientific approach rest in the use of the new embedding architecture of two concurrently running programs in one Measureserver unit. The synchronization is achieved by a specific function to trigger the computation.

7. CONTRIBUTIONS OF THE WORK

The doctoral thesis tries to improve the ISES remote laboratory usability in education and dissemination of scientific knowledge, all that contributes to reliability, robustness, scalability, measured and control data storage and its processing, diagnostics and tools. For this purpose, we formulate the following contributions for the remote laboratories community and researchers.

The first contribution for the community is the implementation of new software modules, built in the Measureserver unit, deployed as the data archiving, clients' activities logging, and two level diagnostics to monitor and recover physical hardware components. These modules provide required functionalities for the REMLABNET platform.

The second contribution to the researchers in the branch is the possibility to have a feedback for remote laboratory operating and eventual improvement. This functionality is provided by the detailed analysis of the clients' activities, based on the data, logged during their experimentations.

The third contribution is to the branch of advanced tools. The embedded real-world phenomena simulation, integrated and synchronized with running remote laboratories, in the Measureserver unit. Simulation, as a simplified version of real experiment maybe used in various combinations with a real experiment, as the standalone, asynchronous and synchronous application, depending on the level of clients. In the simplification, the clients get insight into the basis of the observed phenomenon, in the most progressive case; the client may use the parameters of physics laws to fit the measurements.

The fourth contribution, aiming at the future, are modules important even more when the REMLABNET platform should host a large number of the remote laboratories focused on natural sciences and engineering fields.

CONCLUSIONS

The presented work is focused on the design and development of the software components related to the Measureserver unit. In the frame of work on the doctoral thesis a new Measureserver unit called Intelligent Measureserver was developed to improve stability, reliability and maintenance of the ISES remote laboratories. The implementation also helped the ISES remote laboratories to turn into high level to compete with other similar laboratories worldwide.

The following conclusions concerning the new software components in the Intelligent Measureserver unit are formulated:

1. Fulfillment of the doctoral thesis goals as documented in the individual chapters' summaries of the chapters,
2. Creation of the data archiving management module, responsible for gathering, filtering and saving measured data and metadata to an xml file, together with the option to log clients' activities to a log file,
3. Creation of the Measureserver first level diagnostics, providing the administrators warning and notifying information about the remote laboratories functioning with the recovery features,
4. Creation of the Measureserver second level diagnostics, providing the administrators to detect, monitor and maintain the ISES meters, sensors, probes or devices connected to the remote laboratory with the regards to their failures during the measurement and adjustment process,
5. Creation of the embedded real-world phenomena simulation, operating concurrently with the real remote laboratory with using the solvers to generate approximate solution of the corresponding differential equations, including continuous results visualized on the client's web page.

REFERENCES

[1] MAITI, Ananda, Andrew D. MAXWELL and Alexander A. KIST. Features, Trends and Characteristics of Remote Access Laboratory Management Systems. In: International Journal of Online Engineering (iJOE) [online]. 2014, 10(2), p. 30-37 [cit. 2017-01-07]. DOI: 10.3991/ijoe.v10i2.3221. ISSN 1861-2121. Available: http://online-journals.org/index.php/i-joe/article/view/3221

[2] ŽÁKOVÁ, Katarína. Two ways of inverted pendulum remote control. In: The 6th WSEAS International Conference on Education and Educational Technology, Venice, Italy, November 21-23, 2007, El. Comp. Eng., p. 139-144 [cit. 2017-01-07]. Available: http://www.wseas.us/e-library/conferences/2007venice/papers/570-625.pdf

[3] BISTÁK, Pavol. Matlab and Java Based Virtual and Remote laboratories for Control Engineering. In: The 17th Mediterranean Conference on Control and Automation, Thessaloniki, Jun 24-26, 2009, Vols. 1-3, p.1439-1444 [cit. 2017-01-07]. Available: http://ieeexplore.ieee.org/abstract/document/5164749

[4] VÁLKOVÁ, Lenka and František SCHAUER. Remote interactive real experiment in electrochemistry as exemplified on the experiment Electrochemical cell. In: The 6th Int. Conference on Emerging e-learning Technologies and Applications, Slovakia, September 11-13, 2008 [cit. 2017-01-07]. Available: http://www.iceta.sk/archive/2008

[5] VÁLKOVÁ, Lenka, František SCHAUER and Miroslava OŽVOLDOVÁ. Electrochemical cell characterization - is it start of remote experiments in chemistry education? In: The International Conference REV 2009, Bridgeport (USA), June 22-25, 2009, International Association of Online Engineering, 2009, ISBN 978-3-89958-480-6, p. 326-331 [cit. 2017-01-07]. Available: http://www.utb.cz/file/14174_1_1/download

[6] GARCÍA-ZUBÍA, Javier, Pablo ORDUNA, Ignacio ANGULO, Unai HERNANDEZ, Olga DZIABENKO, Diego LOPEZ-IPINA and Luis RODRIGUEZ-GIL. Application and user perceptions of using the WebLab-Deusto-PLD in technical education. In: First Global Online Laboratory Consortium Remote Laboratories Workshop [online]. IEEE, 2011, p. 1-6 [cit. 2017-04-07]. DOI:

10.1109/GOLC.2011.6086780. ISBN 978-1-4577-1943-1. Available: http://ieeexplore.ieee.org/document/6086780

[7] HARDISON, James L., Kimberly DELONG, Philip H. BAILEY and V. Judson HARWARD. Deploying interactive remote labs using the iLab Shared Architecture. In: 38th Annual Frontiers in Education Conference [online]. IEEE, 2008, S2A-1-S2A-6 [cit. 2017-04-07]. DOI: 10.1109/FIE.2008.4720536. ISBN 978-1-4244-1969-2. Available: http://ieeexplore.ieee.org/document/4720536

[8] LOWE, D., S. MURRAY, E. LINDSAY and D. LIU. Evolving Remote Laboratory Architectures to Leverage Emerging Internet Technologies. In: IEEE Transactions on Learning Technologies [online]. 2009, 2(4), p. 289-294 [cit. 2017-04-07]. DOI: 10.1109/TLT.2009.33. ISSN 1939-1382. Available: http://ieeexplore.ieee.org/document/5210092

[9] TAWFIK, M., E. SANCRISTOBAL, S. MARTIN, et al. Virtual Instrument Systems in Reality (VISIR) for Remote Wiring and Measurement of Electronic Circuits on Breadboard. In: IEEE Transactions on Learning Technologies [online]. 2013, 6(1), p. 60-72 [cit. 2017-04-07]. DOI: 10.1109/TLT.2012.20. ISSN 1939-1382. Available: http://ieeexplore.ieee.org/document/6305453

[10] GOMES, Luís and Javier GARCÍA-ZUBÍA. Advances on remote laboratories and e-learning experiences. Bilbao, Spain: University of Deusto, 2007, 310 p. ISBN 978-84-9830-662-0. Available: https://www.weblab.deusto.es/Advances_on_remote_labs.pdf

[11] GREEN, S. L. and N. M. ANID. Training K-12 teachers in STEM education: A multi-disciplinary approach. In: IEEE Integrated STEM Education Conference (ISEC) [online]. IEEE, 2013, p. 1-4 [cit. 2017-04-07]. DOI: 10.1109/ISECon.2013.6525206. ISBN 978-1-4673-5624-4. Available: http://ieeexplore.ieee.org/document/6525206

[12] BUTIME, J, R. BESIGA, A. BWONYO, V. NAKANWAGI, T. TOGBOA and A. KATUMBA. Design of online Digital Electronics laboratories based on the NI ELVIS II platform. In: 9th International Conference on Remote Engineering and Virtual Instrumentation [online]. IEEE, 2012, p. 1-3 [cit. 2017-04-07]. DOI: 10.1109/REV.2012.6293098. ISBN 978-1-4673-2542-4. Available: http://ieeexplore.ieee.org/document/6293098

[13] GARCÍA-ZUBÍA, Javier and Gustavo R. ALVES. Using Remote Labs in Education. Bilbao, Spain: University of Deusto, 2011, 465 p. ISBN 978-84-9830-398-8. Available: http://www.deusto-publicaciones.es/deusto/pdfs/otraspub/otraspub01.pdf

[14] ORDUNA, Pablo. Transitive and scalable federation model for remote laboratories. Bilbao, Spain, 2013, 242 p. Doctoral thesis. University of Deusto. Supervisor Javier García-Zubía. Available: http://www.weblab.deusto.es/pub/dissertation_pablo.pdf

[15] RICHTER, Thomas, Yvonne TETOUR and David BOEHRINGER. Library of Labs - A European Project on the Dissemination of Remote Experiments and Virtual Laboratories. In: IEEE International Symposium on Multimedia [online]. IEEE, 2011, p. 543-548 [cit. 2017-04-07]. DOI: 10.1109/ISM.2011.96. ISBN 978-1-4577-2015-4. Available: http://ieeexplore.ieee.org/document/6123404

[16] COSTA, David, Gustavo ALVES, Paulo FERREIRA and Juarez SILVA. Remote Labs Accessible through 3D environments A Case Study with Open Wonderland. In: 8th International Conference on Remote Engineering and Virtual Instrumentation [online], Romania: IAOE, 2011, p. 191-197 [cit. 2017-04-07]. Available: www.tinyurl.com/zbyc898

[17] GILLET, Denis, Anh VU NGUYEN NGOC and Yassin REKIK. Collaborative Web-Based Experimentation in Flexible Engineering Education. In: IEEE Transactions on Education [online]. Vol. 48, No. 4. University of Leicester, Leicester, United Kingdom: IEEE, 2005, p. 696-704 [cit. 2017-04-07]. ISSN 0018-9359. Available: http://www.cs.le.ac.uk/people/avnn1/papers/GilletNR-IEEETransEdu05.pdf

[18] FRANTIŠEK, Lustig. : Intelligent School Experimental System - ISES. E-Laboratory Project [online]. Prague, Czech Republic: Faculty of Mathematics and Physics, Charles University in Prague, 2017 [cit. 2017-04-07]. Available: http://www.ises.info/index.php/en/systemises

[19] SCHAUER František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA and Petra ŠPILÁKOVÁ. REMLABNET - Open Remote Laboratory Management System for e-Experiments.

REV, Porto, Portugal, 2014. ISBN 978-1-4799-2024-2. Available: http://ieeexplore.ieee.org/document/6784273

[20] SCHAUER František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA and Petra ŠPILÁKOVÁ. REMLABNET II - Open Remote Laboratory Management System for University and Secondary Schools Research Based Teaching. In: 12th International Conference on Remote Engineering and Virtual Instrumentation [online]. Bangkok, Thailand: IEEE, 2015, p. 109-112 [cit. 2016-09-26]. DOI: 10.1109/REV.2015.7087273. ISBN: 978-1-4799-7839-7. Available: http://ieeexplore.ieee.org/document/7087273

[21] SCHAUER, František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA, Petra ŠPILÁKOVÁ and Lukáš TKÁČ. REMLABNET III - Federated Remote Laboratory Management System for University and Secondary Schools. In: 13th International Conference on Remote Engineering and Virtual Instrumentation [online]. Madrid, Spain: IEEE, 2016, p. 232-235 [cit. 2016-09-26]. DOI: 10.1109/REV.2016.7444471. ISBN 978-1-4673-8245-8. Available: http://ieeexplore.ieee.org/document/7444471

[22] KRBEČEK, Michal, František SCHAUER and Karel VLČEK. Communication Requirements of Laboratory Management System. In: Latest Trends on Systems - Volume II: Proceedings of the 18th International Conference on Systems [online]. Vol. 2. Santorini Island, Greece, 2014, p. 686-691 [cit. 2017-04-07]. ISBN 978-1-61804-244-6. Available: http://www.europment.org/library/2014/santorini/bypaper/SYSTEMS/SYSTEMS2-56.pdf

[23] HAMID, R. and S. MOHAMMED. Remote access laboratory system for material technology laboratory work. In: Proceedings of the 7th WSEAS International Conference on Engineering Education [online]. Corfu Island, Greece: WSEAS, 2010, p. 311-316 [cit. 2017-04-07]. ISBN 978-960-474-202-8. Available: http://dl.acm.org/citation.cfm?id=1864241

[24] KOUKIANAKIS, L. G. and J. G. GLENTZES. A virtual lab and e-learning system for renewable energy sources. In: WSEAS Transactions on Computers [online]. Vol. 5, No. 2. Greece: WSEAS, 2006, p. 337-341 [cit. 2017-04-07]. ISSN 1109-

2750. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.579.9961&rep=rep1&type=pdf

[25] KELLER, Robert. Finite-State Machines. Finite Automata and Formal Languages [online]. Claremont, California, USA: Harvey Mudd College, 2009, p. 471-545 [cit. 2017-04-07]. Available: http://www.cse.chalmers.se/~coquand/AUTOMATA/book.pdf

[26] BRADEN, R. Requirements for Internet Hosts - Communication Layers. Internet Standard [online]. Marina del Rey, California, USA: Internet Engineering Task Force, 1989 [cit. 2017-04-07]. Available: https://tools.ietf.org/html/rfc1122

[27] BRADEN, R. Requirements for Internet Hosts - Application and Support. Internet Standard [online]. Marina del Rey, California, USA: Internet Engineering Task Force, 1989 [cit. 2017-04-07]. Available: https://tools.ietf.org/html/rfc1123

[28] LUSTIG, František, Jiří DVOŘÁK, Pavel KURIŠČÁK and Pavel BROM. Building your own real remote experiment controlled by a mobile or touch enabled device. Information and communication Technology in Education [online]. Rožnov pod Radhoštěm: University of Ostrava, 2015, p. 88-95 [cit. 2017-04-07]. Available: http://www.ises.info/old-site/clanky_pdf/Lustig_ICTE_2015.pdf

[29] WEST, Matt. An Introduction to WebSockets. What You Can Learn in 15 Minutes A Day [online]. Treehouse Island, 2013 [cit. 2017-04-07]. Available: http://blog.teamtreehouse.com/an-introduction-to-websockets

[30] KRBEČEK, Michal, František SCHAUER and František LUSTIG. Easy Remote ISES - Development Environment Remote Experiments. In: Innovations 2013: World Innovations in Engineering Education and Research [online]. USA: Potomac, 2013, p. 81-100 [cit. 2017-04-07]. ISBN 978-0-9818868-4-8. ISSN 1553-9911. Available: http://www.ises.info/old-site/clanky_pdf/Easy_Remote_ISES_2013.pdf

[31] HENKE, Karsten, Tobias FÄTH, René HUTSCHENREUTER and Heinz-Dietrich WUTTKE. GIFT - An Integrated Development and Training System for Finite State Machine Based Approaches. In: 14th International Conference on Remote Engineering and Virtual Instrumentation [online].

New York, USA: IAOE, 2017, p. 125-139 [cit. 2017-04-07]. Available: https://www.tu-ilmenau.de/en/integrated-communication-systems-group/publications/?topic_id=&publication_id=461

[32] GERŽA, Michal, František SCHAUER and Petr DOSTÁL. Embedded Simulations in Real Remote Experiments for ISES e-Laboratory. In: 9th EUROSIM Congress on Modelling and Simulation [online]. Oulu, Finland: IEEE, 2016, p. 653-658 [cit. 2016-09-26]. DOI: 10.1109/EUROSIM.2016.66. ISBN 978-1-5090-4119-0. Available: http://eurosim2016.automaatioseura.fi/images/sas/Full-Program-11-Sept-Net.pdf

[33] GERŽA, Michal, František SCHAUER and Karel VLČEK. Communication Principles Between Client and Physical Hardware of ISES Remote Laboratory. AMCSE, Varna, Bulgaria, 2014. ISBN 978-1-61804-246-0. Available: http://www.inase.org/library/2014/varna/bypaper/AMCSE/AMCSE-03.pdf

[34] GERŽA, Michal, František SCHAUER and Karel VLČEK. Advanced Communication Diagnostics in ISES Remote Experiment. International Journal of Communications [online]. North Atlantic University Union, 2015, p. 43-52 [cit. 2015-10-12]. ISSN 1998-4480. Available: http://www.naun.org/main/NAUN/communications/2015/a142001-337.pdf

[35] GERŽA, Michal, František SCHAUER and Ivan ZELINKA. Artificial Intelligence in ISES Measureserver for Remote Experiment Control. NOSTRADAMUS, Ostrava, Česká Republika, 2014. ISBN 978-3-319-07401-6. ISSN 2194-5357. Available: https://link.springer.com/chapter/10.1007/978-3-319-07401-6_41

[36] R. WRIGHT, David. Finite State Machines. CSC 216 Course Website [online]. USA: NC State University, 2005 [cit. 2017-04-07]. Available: http://www4.ncsu.edu/~drwrigh3/docs/courses/csc216/fsm-notes.pdf

[37] DICK, Grune and Ceriel J.H. JACOBS. Parsing and its applications, a practical guide. Chichester, England: Ellis Horwood, 1990. ISBN 0-13-651431-6. Available: https://dickgrune.com/Books/PTAPG_1st_Edition

[38] TRIBBLE, David. LR(k) Parsing Theory. Practical LR(k) Parser Construction [online]. Plano, Texas, USA, 2004 [cit. 2017-04-07]. Available: http://david.tribble.com/text/lrk_parsing.html

[39] BUCK, Jamis. Writing a Simple Recursive Descent Parser. The Buckblog [online]. Smithfield, Utah, USA, 2015 [cit. 2017-04-07]. Available: http://weblog.jamisbuck.org/2015/7/30/writing-a-simple-recursive-descent-parser.html

[40] GHOST, Gautam, Gauray ROY and Vikash SINGH. C++ Introduction. C++ Tutorials [online]. w3schools.in [cit. 2017-04-07]. Available: http://www.w3schools.in/cplusplus-tutorial/intro

[41] VISHWANATH, Siddharth and Shivendra KUMAR. LR(1) Parsers: Theory and Implementation [online]. IKanpur, Iindia: Indian Institute of Technology, 2012 [cit. 2017-06-29]. Available: https://www.academia.edu/4188731/LR_1_Parsers_Theory_and_Implementation

[42] NORVELL, Theodore. Parsing Expressions by Recursive Descent. Informal Publications [online]. Canada: Memorial University of Newfoundland, 1999 [cit. 2017-06-29]. Available: https://www.engr.mun.ca/~theo/Misc/exp_parsing.htm

[43] KOVÁŘ, Martin. Diagnostics and Elucidation of the Environment Web ISES Control Kit for Remote Experiments Control. Zlín, Czech Republic, 2012, 78 s. Bachelor Thesis. Tomas Bata University in Zlín. Available: http://digilib.k.utb.cz/handle/10563/22906

[44] RAY, Erik. Creating Self-Describing Data - Learning XML [online]. Sebastopol, USA: O'Reilly & Associates, 2001, 368 p. [cit. 2017-04-07]. ISBN 0-59600-046-4. Available: https://doc.lagout.org/programmation/Java/OReilly%20Learning%20XML.pdf

[45] CAMPIONE, Ben. An MFC Wrapper for MSXML. Dr. Dobb's - The world of software development [online]. San Francisco, California, USA: UBM, 2001 [cit. 2017-04-07]. Available: http://www.drdobbs.com/an-mfc-wrapper-for-msxml/184416288

[46] NAGY, Gabriel. Ordinary Differential Equations [online]. East Lansing, MI, USA: Mathematics Department, Michigan State

University, 2017 [cit. 2017-07-16]. Available: http://users.math.msu.edu/users/gnagy/teaching/ode.pdf

[47] IERUSALIMSCHY, Roberto, Waldemar CELES and Luiz FIGUEIREDO. Lua Documentation: Reference manual. The Programming Language [online]. Rio de Janeiro, Brazil: LabLua, 2017 [cit. 2017-06-25]. Available: https://www.lu[48]a.org/docs.html

[48] ESQUEMBRE, Francisco. EjsWiki Documentation. About Easy Java/Javascript Simulations [online]. Murcia, Spain: University of Murcia, 2017 [cit. 2017-06-25]. Available: http://www.um.es/fem/EjsWiki/Main/Documentation

[49] KRISHNAN, Mangala, Kushal SEN and Bhaskar RAMAMURTHI. Numerical Solution of Ordinary Differential Equations: Modified Euler Method. National Programme on Technology Enhanced Learning [online]. Chennai, India: NPTEL, 2017 [cit. 2017-06-25]. Available: http://nptel.ac.in/courses/111107063/3

[50] KRISHNAN, Mangala, Kushal SEN and Bhaskar RAMAMURTHI. Numerical Solution of Ordinary Differential Equations: Fourth Order Runge Kutta Methods. National Programme on Technology Enhanced Learning [online]. Chennai, India: NPTEL, 2017 [cit. 2017-06-25]. Available: http://nptel.ac.in/courses/111107063/5

PUBLICATION ACTIVITIES OF THE AUTHOR

1. GERŽA, Michal and Pavel POKORNÝ. A Visualisation of the Results of a Thermoforming Process Simulation in the Plastics Industry. In: 21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision: WSCG2013 - Poster Proceedings. Plzeň: Západočeská univerzita v Plzni, Katedra informatiky a výpočetní techniky, 2013. ISBN 978-80-86943-76-3.

2. GERŽA, Michal, František SCHAUER and Roman JAŠEK. Security of ISES Measureserver Module for Remote Experiments against Malign Attacks. International Journal of Online Engineering, Wien, Austria, 2014. ISSN 1868-1646.

3. GERŽA, Michal, František SCHAUER and Karel VLČEK. Communication Principles between Client and Physical Hardware of ISES Remote Laboratory. AMCSE, Varna, Bulgaria, 2014. ISBN 978-1-61804-246-0.

4. GERŽA, Michal, František SCHAUER and Ivan ZELINKA. Artificial Intelligence in ISES Measureserver for Remote Experiment Control. NOSTRADAMUS, Ostrava, Česká Republika, 2014. ISSN 2194-5357.

5. SCHAUER František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA and Petra ŠPILÁKOVÁ. REMLABNET - Open Remote Laboratory Management System for e-Experiments. REV, Porto, Portugal, 2014. ISBN 978-1-4799-2024-2.

6. GERŽA, Michal and František SCHAUER. Advanced Modules Diagnostics in ISES Remote Laboratories. In: The 10th International Conference on Computer Science & Education. Fitzwilliam College, Cambridge University, UK, 2015. ISBN 978-1-4799-6599-1.

7. SCHAUER František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA and Petra ŠPILÁKOVÁ. REMLABNET II - Open Remote Laboratory Management System for University and Secondary Schools Research Based Teaching. In: 12th International Conference on Remote Engineering and Virtual Instrumentation [online]. Bangkok, Thailand: IEEE, 2015, p. 109-112 [cit. 2016-09-26]. DOI: 10.1109/REV.2015.7087273. ISBN: 978-1-4799-7839-7.

8. GERŽA, Michal, František SCHAUER and Karel VLČEK. Advanced Communication Diagnostics in ISES Remote Experiment. International Journal of Communications [online]. North Atlantic University Union, 2015, p. 43-52 [cit. 2015-10-12]. ISSN 1998-4480.

9. GERŽA, Michal and František SCHAUER. Intelligent Processing of Experimental Data in ISES Remote Laboratory. International Journal of Online Engineering [online]. Wien, Austria, 2016, (Vol 12, No 03), p. 58-63 [cit. 2016-09-26]. DOI: 10.3991/ijoe.v12i03.5538. ISSN 1861-2121.

10. GERŽA, Michal, František SCHAUER and Petr DOSTÁL. Embedded Simulations in Real Remote Experiments for ISES e-Laboratory. In: 9th EUROSIM Congress on Modelling and Simulation [online]. Oulu, Finland: IEEE, 2016, p. 653-658 [cit. 2016-09-26]. DOI: 10.1109/EUROSIM.2016.66. ISBN 978-1-5090-4119-0.

11. SCHAUER, František, Michal KRBEČEK, Pavel BEŇO, Michal GERŽA, Lukáš PÁLKA, Petra ŠPILÁKOVÁ and Lukáš TKÁČ. REMLABNET III - Federated Remote Laboratory Management System for University and Secondary Schools. In: 13th International Conference on Remote Engineering and Virtual Instrumentation [online]. Madrid, Spain: IEEE, 2016, p. 232-235 [cit. 2016-09-26]. DOI: 10.1109/REV.2016.7444471. ISBN 978-1-4673-8245-8.

12. SCHAUER, František, Michal GERŽA, Michal KRBEČEK, Pavel BEŇO, Lukáš PÁLKA, Petra ŠPILÁKOVÁ, Tomáš KOMENDA, Miroslava OŽVOLDOVÁ, Žaneta GERHATOVÁ and Lukáš TKÁČ. REMLABNET IV - LTI Federated Remote Laboratory Management System with Embedded Simulations. In: 14th International Conference on Remote Engineering and Virtual Instrumentation [online]. New York, USA: IEEE, 2017, p. 340-349 [cit. 2017-06-19].

13. SCHAUER, František, Michal GERŽA, Michal KRBEČEK and Miroslava OŽVOLDOVÁ. Remote Experiment Wave Laboratory for Wave Phenomena Teaching. In: 14th International Conference on Remote Engineering and Virtual Instrumentation [online]. New York, USA: IEEE, 2017, p. 350-356 [cit. 2017-06-19].

CURRICULUM VITAE

Michal Gerža, MSc.

Address:	Jílová 4571 760 05 Zlín Czech Republic
Phone:	+420 603 732 198
Email:	michal.gerza@email.cz

Personal data:

Date of birth:	22. November 1974
Marital status:	Married
Mother tongue:	Czech language

Education:

2007 – to this day	Thomas Bata University in Zlín, Faculty of Informatics, Postgraduate study (Ph.D.), Field of study: Engineering Informatics
2007 – 2012	Thomas Bata University in Zlín, Faculty of Informatics, Bachelor and graduate study, Field of study: Engineering Informatics
1989 – 1993	Secondary school of Electrotechnic in Vsetín Field of study: Mechanical engineering

Professional experiences:

2010 – to this day	Accuform, Zlín, Programmer
2009 – 2010	Edhouse, Ltd., Zlín, Programmer
1993 – 2009	Glass Service, Inc., Vsetín, Programmer

Language:

- English
- Russian

Skills:

- Flexibility
- Independence
- Responsibility
- Communicableness
- Interest in communication with people
- Leading with human resources
- Project management
- Computer skills

Projects:

- SCOPES project (Swiss National Science Foundation and Swiss Agency for Developments and Cooperation) aimed at the remote laboratories community

Hobby:

- Fitness
- Tourism
- Psychology
- Computer technology

Other knowledge:

- Programing languages: C, C++, C#, Java, JavaScript, Visual Basic
- Web applications: PHP, CSS, XML, XSL, XAML, HTML
- Database systems: MSSQL, MySQL, PostgreSQL
- Computer systems: Windows, Linux