Technical University of Valencia
Department of Computer Systems and Computation

EL TRABAJO FIN DE MASTER

THE MASTER'S THESIS

Diseño y Desarrollo de un software para el cronometraje de eventos deportivos de alta participación no motorizados.

Design and Implementation of the software for the timing purposes of the high participation non-motorized sports events.

Paweł Kubicz

Supervisor:
Vicente Pelechano Ferragud, PhD

Valencia, 2014

ACKNOWLEDGMENTS

I would like to acknowledge all the people that contributes to my practical as well as theoretical work and supports me during my hard and exhaustive work.

First I would like to express my gratitude to my supervisor Vicente Pelechano Ferragud, PhD for his advice and supervision of my work.

Then I would like to thanks to a company Cronochip and its boss Abraham Serra for great cooperation and engaging me into the project.

Finally, I gratefully acknowledge my family and my girlfriend that gives me great support and motivated me to the work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURE

ABBREVIATIONS

AIDC - Automatic Identification and Data Capture

API - Application Programming Interface

CLR - Common Language Runtime

CUBICSOFT – a software described in this thesis for timing control

DLL - Dynamic-Link Library

FCL - .NET Framework Class Library

GUI - Graphical User Interface

IDE - Integrated Development Environment

JSON - JavaScript Object Notation

MSIL - Microsoft Intermediate Language

NTP - Network Time Protocol

RFID - Radio-Frequency Identification

RDBMS - Relational Database Management System

SVN – Subversion

TIMINGSENSE – a name of new timing system

UHF - Ultra High Frequency

XML - Extensible Markup Language

ABSTRACT

The master's thesis consist of the design and development of the software for the timing purposes of the high participation non-motorized sports events (athletic races, cycling, triathlon, etc.). Given software should be intuitive and easy to use by end users (timekeepers). The software must fully manage all aspects and features of the race, the athlete registrations, results calculation, reporting etc. A timing process is done by RFID technology. The company Cronochip, where the given project is being developed, is developing its own timing system, equipment and chips using RFID UHF second generation technology for time control. One of the main objectives of the project is that the software can communicate with a timing equipment to receive chips readings (participants are carrying chips), monitor the status of equipment, recover old chips readings, etc. Likewise, the company is also working on developing an online platform that will host online registration, results in real-time interaction with social networks, etc. The timing software developed in this thesis should be integrated with the online platform to import participants and configuration of a races and to publish the results so that the whole process becomes easier for the timekeeper. Presented software, which name is CubicSoft, is a motherboard for the newly formed sport timing system, TimingSense, which manages all functionality and communication between timing equipment and online platform.

RESUMEN

El trabajo fin de máster se lleva a cabo el diseño y desarrollo de un software para el cronometraje de eventos deportivos, tales como carreras de atletismo, ciclismo y triatlón. Este software debe ser intuitivo y fácil de manejar por los usuarios finales (cronometradores). El software debe gestionar de forma completa todos los aspectos y características de la prueba, de las inscripciones de atletas, el cálculo de resultados, la generación de informes, la gestión de incidencias, etc. El cronometraje se realizará haciendo uso de la tecnología rfid. La empresa Cronochip, donde se realiza el proyecto, está desarrollando sus propios equipos y chips utilizando la tecnología rfid de segunda generación para el control de tiempos. Uno de los objetivos esenciales del proyecto es que el software pueda comunicarse con los equipos de cronometraje para poder recibir las lecturas de chips (que llevarán los participantes), monitorizar el estado de los equipos, recuperar lecturas antiguas, etc. Así mismo, también la empresa está trabajando en el desarrollo de una plataforma online que albergará inscripciones online, resultados, resultados en tiempo real, interacción con redes sociales, etc. El software de cronometraje desarrollado en esta tesina debe integrarse con la plataforma online para poder importar inscripciones y definiciones de pruebas, así como para poder publicar los resultados de forma que todo el proceso resulte sencillo para el cronometrador.

1. INTRODUCTION

1.1. GENERAL INTRODUCTION

Nowadays where a running is a hot topic and a lot of people is training some sports, there are organized a competitions where is a need to control time of each of participants especially in sports like athletic races, cycling, triathlon, etc. Mostly each sport event need time controlling. In this case is a need of use a sophisticated tools for timing purposes. A timing system is a very powerful and complex instrument.

My work is dedicated to the Cronochip company located in Pobla de Vallbona and is concentrated generally on creating a part of timing system of the high participation non-motorized sports events. The field of sport timing is fairly developed. Nowadays there are some companies in the world which are specialized in this area. The main part of the timing system is a time control software which creation process is presented in given master's thesis. There are some programs available on the market that can manage a time measuring. These applications are complex and offer various services to manage a time measuring like manage all aspects and features of the race, calculates result, generate reports, etc. However most of them have a lot of options which are not used by a casual user (timekeepers) and are very difficult to manipulate. Normally there is a need of special training to learn how to use a given software. Apart from these programs are expensive, usually each timing system has its own dedicated software to manage a time measuring.

As mentioned above a new timing system, which name is TimingSense, is being created. TimingSense works with the technology radio-frequency identification, RFID, ultra high frequency, UHF, the second generation. Therefore there was designed and implemented unique software dedicated to cooperate with TimingSense system to manage a time measuring during non-motorized sport competitions. The presented program deliver all indispensable functions and services for future timekeepers, which are described in detail later in this thesis, starting from race preparation, result calculation and ending on sending result to online platform and report generation.

1.2. MOTIVATION

The thing that motivates me the most, was the importance of the RFID technology in our everyday life and being a part of a newly emerging timing system. Nowadays RFID is a technology which is applicable in many areas of everyday life like credit cards, tickets, advanced systems, etc. and every year the demand on this technology is increasing. Personally, I was always interested in the RFID technology. It is inspiring me and this project give me a real opportunity to work with the RFID technology. I was faced with opportunity to create a part of a timing system which can be used in future by many people and somehow help develop the idea of company Cronochip where I was implementing my project and where I can give my own contribution to the project. Consequently, I decided to devote my master's dissertation for purposes of the Cronochip located in Pobla de Vallbona. In cooperation of Polytechnic University of Valencia and company Cronochip I have been implementing the software for sport event timing purposes. What motivates me the most was the fact that my hard work will be used by other people (timekeepers) in their everyday work all over the world.

2. BACKGROUND

The chapter of theoretical background presents the general requirements imposed on the project, main technologies used in project, principle of the RFID and description of a main components used in project. At the end of the chapter there is a revision of already existing solutions on the market, existing software to manage a time measuring.

2.1 GENERAL REQUIREMENTS

First aspect of my work was to gathered necessary information and make investigation of the sport timing environment. On this basis the project was designed and then implemented.

Application, which is named CubicSoft, had to meet six basic requirements given by the company Cronochip:

- devotes to work with timekeepers
- has user-friendly interface
- provides a simple management of time measuring
- provides an effective result calculation
- generates report
- be well-documented

Before the work over project have been started there was a need to choose correct technologies that will satisfy the requirements of TimingSense system and will not force final users to unnecessary installation of additional software. The first step was to investigate the sport timing environment. After short examination and consults the standard configuration of the workstation was determined. CubicSoft will be designed to work with Microsoft Windows operative system.

Finally after gathering basic information the correct technologies could be selected. Because of operating systems it was decided to use .NET Framework 4 which is a integrated component of Windows 7 or later, there is no need to install extern software.[20] The next technology selected to be deployed was programming language C# 4.0. As it is the most powerful and the most common .NET language.[21] My skills and great knowledge of this programming language led me to this choice. Moreover there was some doubts in choosing the correct data storage system. In case of database technology selection there was considered

two options Microsoft SQL and MySQL. After long deliberation and analysis of pros and cons the second option was chosen, MySQL technology from the Oracle company.[3] The last aspect is a communication between CubicSoft and, TimingSense equipment or TimingSense online platform. To interchange information between TimingSense system over TCP/IP it is used JSON format to transmit a data object. All above technologies are describes in detail with exact explication of every aspect in the next subsection.

2.2. DEPLOYED TECHNOLOGIES

This section describes four main technologies that were involved in my project. They are the core of my work and without them it will be impossible to create this application. Frankly speaking, they consist mostly of Microsoft proprietary technologies starting from programming environment .NET Framework 4, programming language C# 4.0 and development environment Microsoft Visual Studio 2010 and ending on Oracle technology, database system which name is MySQL.

2.2.1. .NET FRAMEWORK

The core technology used to create the CubicSoft software was the .NET Framework 4.0. The Microsoft .NET Framework is a software framework dedicated to Microsoft Windows operating systems, but not only. The .NET Framework consist of components needed to develop, deploy, and execute Web applications, Windows applications, Web services, Windows services, and Console applications. The .NET Framework has a three-level hierarchy which is in constant improvement:[3]

- CLR
- .NET Framework class library
- development technologies like ADO.NET, ASP.NET, Windows Forms, Windows Presentation Foundation, Windows Communication Foundation, Windows Workflow Foundation, Windows Card Space and LINQ

The .NET Framework development technologies and class libraries are increasing with every new release of the framework starting from the first .NET 2.0. Now the newest .NET framework is 4.5 which adds some new functionalities like asynchronies model to the .NET framework hierarchy. Above picture presents the .NET Framework 4.0 hierarchy.

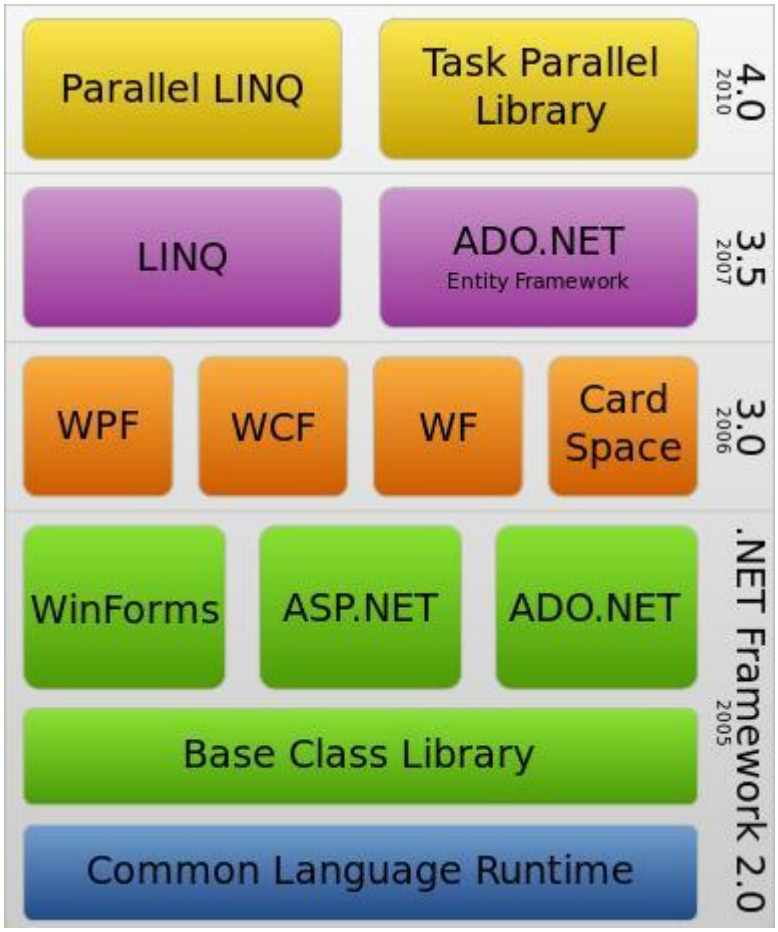

Figure 1. The .NET Framework 4.0 hierarchy[23]

The Common Language Runtime, known as CLR, is the base of the .NET Framework applications. The CLR is what actually loads, verifies, and executes development technologies. The CLR is responsible for managing memory, code execution, thread execution, compilation, code safety verification, etc. The fundamental principle of .NET runtime is a code management. Code that has its execution managed by the CLR is known as managed code otherwise when code that do not run under the control of the CLR is said to be unmanaged as showed on the picture below. Mostly all technologies available with .NET use manage code, specific is ASP.NET technology.

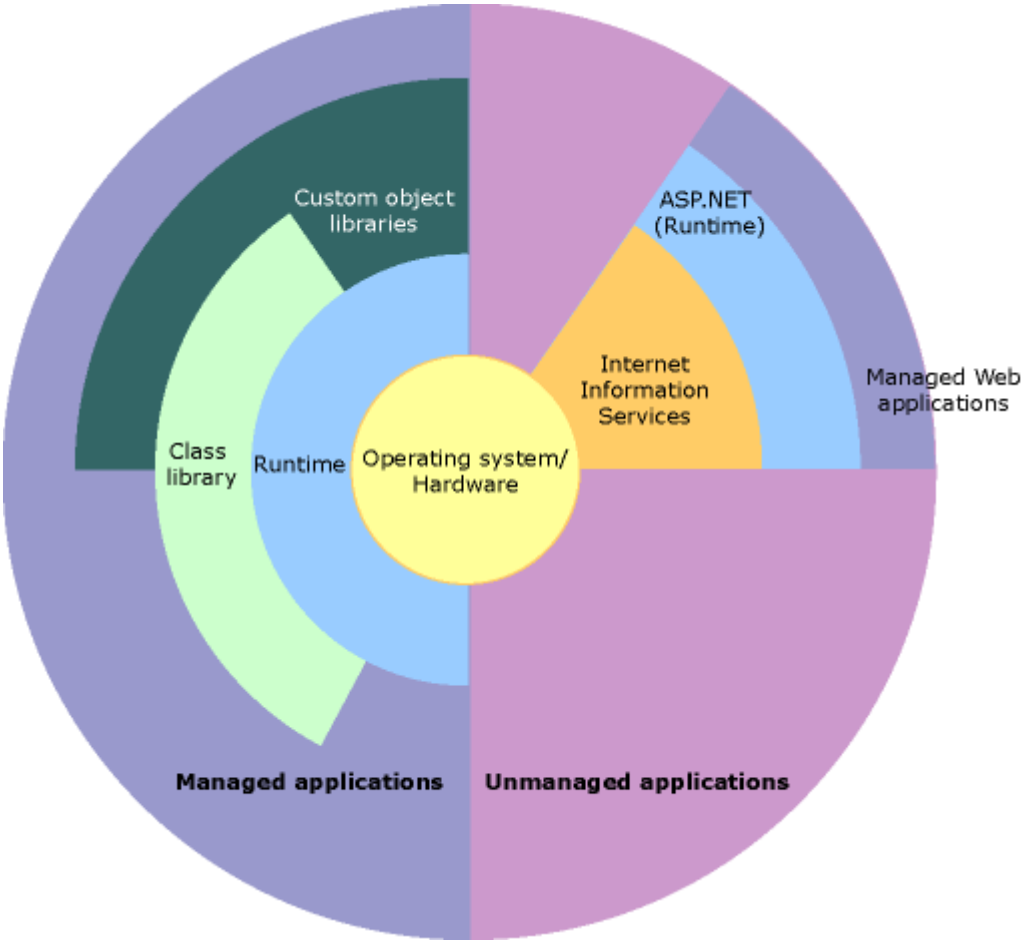

Figure 2. Relationship of the CLR and the class library in an applications and to the overall system[23]

The other important component of .NET Framework is .NET Framework Class Library, FCL, which includes a large number of reusable classes organized in a hierarchy of namespaces.[11] The FCL is an object oriented class which is hierarchically design and contains a subset Base Class Library. It contains the main set of namespaces common for all .NET languages. Whereas the FCL is superset of the Base Class Library and contains the entire class library for .NET framework. Some of the most frequently used namespaces which are included in my project are placed in the Table 1 presented below.

Table 1. Some of the most common namespaces included in project[5]

Namespace	Description
System	The core set of main classes and all data types: Math, Integer, Boolean, String, etc.
System::Collections	Defines commonly used collections like: List, Stack, Queue, etc.
System::Diagnostics	Includes event logging and provides interaction with system processes.
System::IO	Enables reading and writing from/to different streams (file, memory, network), provides a link to the file system, etc.
System::Exception	Provides a support to handle all kind of exception during execution of the application
System::Windows::Forms	Provides access to the native Microsoft Windows interface and contains classes for creating Windows-based applications (overridden in this project): Controls, Menus and Toolbars, Dialog Boxes, etc.

The principal design features of the .NET Framework:

- .NET language is neutral. All existing .NET compilers, compile source code to Microsoft Intermediate Language, MSIL. A source code can be developed using one of the languages provided by Microsoft like C#, C++/CLI or by third parties like Delphi. All .NET compatible languages can use the Base Class Library.
- .NET can be platform independently. It is possible to implement .NET Framework application to non-Windows platforms[5]
- .NET free developers from the memory management. There is no worry about memory leakiness because .NET is deleting allocated memory alone due to garbage collector.
- .NET provides consistent framework thanks to it developers can create their own applications.

In general this project is concentrating on Windows Application technology more precise on the Windows Forms namespace which is the graphical Application Programming Interface, API, included as a part of .NET Framework base library.

2.2.2. C# PROGRAMING LANGUAGE

The software was created by use of the programming language C# 4.0, multi-paradigm programming language (object-oriented, component-based). It is a principle Microsoft programming language developed within .NET initiative. C# is built on the syntax and semantics of well known C++, allowing C programmers to take advantage of .NET and the CLR.

Main C# goals used in its design were as follows:[6]

- C# is intended to be a simple, general-purpose, modern, object-oriented programming language.
- The language, and implementations, should provide support for software engineering principles and automatic garbage collection.
- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Source code portability is vital, as is programmer portability, especially for those programmers already familiar with C and C++.
- Support for internationalization.
- C# try to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
- C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C language.

C# programming language has gone through several versions. The version used in this project is 4.0. This version is compatible with .NET framework 4.0. The newest version is C# 5.0 which was launched in ends of 2012 and works only with the newest .NET framework 4.5. Independently of the C# version a main compiler is Microsoft Visual C# built in development environment such as Microsoft Visual Studio.

C# is a programming language that the most directly reflects the underlying MSIL. Some important features of C# that distinguish it from C and C++ are:[6]

- Like C++, and unlike Java, C# programmers must use the keyword ***virtual*** to allow methods to be overridden by subclasses.
- Local variables cannot shadow variables of the enclosing block, unlike C++ and C.

- C# ***namespace*** provides the same level of code isolation as a Java ***package*** or a C++ ***namespace.***
- Managed memory cannot be explicitly freed; it is automatically garbage collected. Garbage collection is freeing the programmers of responsibility for releasing memory that is no longer needed.
- C#, unlike Java, supports operator overloading.
- C# is more type safe than C++.
- In addition to the ***try...catch*** construct to handle exceptions, C# has a ***try..catch...finally*** construct to guarantee execution of the code in the ***finally*** block.
- C# language does not allow implicitly for ***global*** variables or functions. All methods and members must be declared within classes and only ***static*** members of public classes can substitute a ***global*** variables and functions.

C# has a unified type system that implies that all types, including primitives such as ***string***, are subclasses of the ***System.Object*** class. In C# a data type is divided in two categories: value type and reference type. Value type is a primitive data type like ***integer, float, char***, etc. On the other hand reference type has the notation of referential identity such as ***object.*** What is more in C# occurs two important terms when dealing with data types boxing and unboxing.[24] Boxing is the operation of converting a value type data into corresponding a reference type value like ***integer*** to ***object,*** it is always implicit. Inverse operation is unboxing where a value of a reference type is converted into a data of a value type, it require an explicit type cast. Important aspect of C# is generic data type which allows for example to define generic classes, methods or collections. Generics use type parameters, which make it possible to design classes, methods or collections that do not specify the type used until the element is instantiated. The main advantage of using generic type parameters to create classes, methods or collections is no need to additional cost of runtime casts or boxing operations.

2.2.3 MYSQL RDBMS

The MySQL is an open-source relational database management system, RDBMS, which belongs to Oracle company. The MySQL source code is available under the terms of the General Public License, MySQL is an open source project. MySQL is the world's most popular open source database software and has over 100 million copies downloaded and distributed.[25] MySQL is a part of LAMP (Linux, Apache, MySQL, PHP / Perl / Python), which is the fast-growing open source enterprise software package.

The actual version of MySQL which is used in this project 5.7. The MySQL from version 5.5 is full featured database system which is one of the best free database and can compete with commercial ones like Microsoft SQL. MySQL is characterized by following features:[26]

- consistent with ANSI SQL 99
- high availability and velocity
- cross-platform support
- stored procedures
- cursors
- triggers
- multiple storage engines
- Unicode support
- etc.

MySQL was chosen to this project in place of Microsoft SQL, which is high compatible with .NET Framework, because of two factors. First thing is that MySQL is an open source, non commercial software what means that has high scalability (works on almost every platform and architecture), has great support, is cheap in maintains and available for everyone without any additional limitations. The second factor and more important is a multiple storage engines support. MySQL support different kind of engines and each type is designed for different use as shown in Table 2 below.

Table 2. Most important storage engines in MySQL[27]

Engine name	Description
MyISAM	does not support transactions nor even foreign keys, but allows (as opposed to the other types) full-text search
InnoDB	is the most powerful engine which supports transactions, foreign keys, etc
MEMORY	the fastest engine which is storing data in memory, RAM. It has several limitations like does not store the data after shutdown, not support all types of data
FEDERATED	enables the creation of distributed databases
CSV	stores data in the standard of CSV files

Implemented software CubicSoft use two types of MySQL, the most functional MySQL InnoDB to store all database on hard disc (is used to manage all race data) and for the result calculation MySQL Cluster, the high speed and the high availability database. MySQL Cluster is a technology providing shared-nothing clustering for the MySQL database management system normally held in the memory RAM and optional in the hard disk. It is designed in a way to provide high availability and high throughput with low latency. MySQL Cluster is implemented through the Network Database NDB engine for MySQL.[28] The main assumptions of MySQL Cluster are: data replication (MySQL Cluster uses synchronous replication of data with a double confirmation of the changes in the database.), shared-nothing clustering architecture (does not have in its system a single points of failure) and hybrid storage (a data can be stored in the memory RAM and in the hard disk). The implementation of MySQL Cluster is enough simple and has three main parts: data node (store the data), management node (store the configuration) and API node (manage a requests from the client to the database). MySQL Cluster was found as the most appropriate technology and was chosen to accelerate a race result calculation. All of the factors described above are the reason of choosing MySQL as a main technology to store data in given project.

2.2.4 JSON DATA INTERCHANGE FORMAT

JavaScript Object Notation, JSON, is a lightweight computer data interchange format, text format, that facilitates structured data interchange between all programming languages. JSON is syntax of braces, brackets, colons, and commas similar to XML that is useful in many applications in data transfer. JSON was inspired by the object literals of JavaScript. JSON format is independent of any particular language. Many programming languages support this format of the data by additional packages or libraries.

In the project is used a JSON framework for .NET to interchange data between TimingSense systems' software. In typical cases, the data in JSON format is retrieved from the server as a text using *HttpWebRequest* object, and then converted to an specific JSON object. The text should be encoded with UTF-8, default JSON format.[29] JSON provides support for basic structure like object and for ordered lists of values. All programming languages will have some feature for representing such lists, which can be array, vector, or list. By accepting JSON's simple convention, complex data structures can be easily interchanged between incompatible programming languages. The basic structure an object is an unordered set of name/value pairs. An object begins with '***{***' and ends with '***}***'. Each name is followed by '***:***' and the name/value pairs are separated by '**,** '.

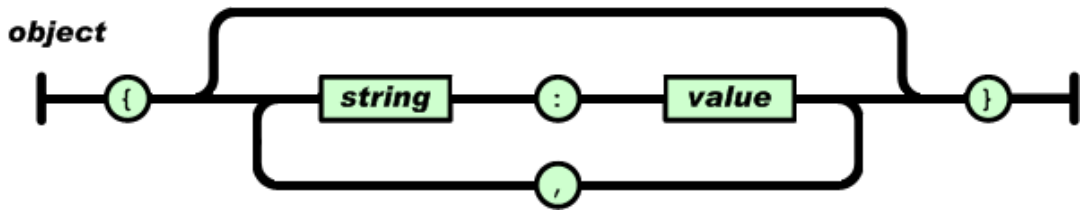

Figure 3. Graphical representation of the basic JSON structure, ***object.***[29]

As a limitation JSON does not support cyclic graphs, at least not directly. JSON is not indicated for applications requiring binary data.[7]

During software implementation there was doubts between XML and JSON format efficiency, but as a result of better data compression the second method JSON was chosen. What is more access to data in JSON format is more natural than access to the same data in XML format.

2.3. TIMING SOFTWARE

In this section are described specific technologies used to create the software for the time control purposes, CubicSoft, and there is a brief introduction to the world of races, is presented necessary terminology and methodology of a race timing. This part is crucial to understand the complexity of the timing software and the timing system at all.

2.3.1 CUBICSOFT'S COMPONENTS

CubicSoft, the timing software, was implemented as was mentioned earlier with technology .NET framework 4.0 and programming language C# 4.0. The main data storage technology is MySQL and to interchange data between the timing system is used JSON format. The whole layout of the CubicSoft was created due to DevExpress GUI widgets, a unique commercial part used in the project.

DEVEXPRESS CONTROLLER

Firstly the DevExpress' controllers will be described. In the project is incorporated a framework of DevExpress WinFroms, version 13.6, which is working with .NET Framework 4.0 and Microsoft Visual Studio 10.[32] DevExpress WinFroms is a set of GUI widgets, controls, for Windows Forms which are useful to create an application and contains a lot of additional functionality which was helpful during software implementation. DevExpress WinFroms contains more than 100 controls which are divided in four principal categories and then in subcategories:

- *Office-Inspired* consist of Data Grid, Spreadsheet, Scheduler, etc.
- *Reports & Analytics* consist of Banded Reporting, Charting, Gauges, etc
- *Windows 8 UI* contains a package of components for Windows 8 (touch screen)
- *Navigation & Layout* consist of Ribbon Menu, Docking, Navigation Bar, Layout Manager, Wizard Control, etc.

CubicSoft is based on components from *Navigation & Layout* category (used to create GUI) and *Office-Inspired* category (used to give additional functionality). The most common used component in a project is a *Data Grid* controller which contains a set of functional grids where the race data is presented and then modified by the user. There is also worth mentioning a component from *Reports & Analytics* category to generates reports.

On the market exists controllers similar to DevExpress. Before the project design started there was a debate in Cronochip which components is more appropriate. There was doubts between DevExpress and Telerik product.[33] Taking into account a price and a more appropriate design of controllers, chosen was DevExpress. What was more important in our case, was a design of the *Data Grid* controller which in case of DevExpress satisfy the project needs. DevExpress also posses elegant, customizable WinForms themes and app skins, straightforward localization and easy-to-use application templates.

MYSQL CONNECTOR/NET DRIVER

The next important component of this project is a ADO.NET Driver for MySQL (Connector/NET). In the development environment Visual Studio 10 only three types of database are available by default: Microsoft SQL, IBM and Oracle. To use additional database is needed a special driver. MySQL distribute Connector/NET driver which help to connect and manage a database from Microsoft Visual Studio.[34] There is a need to deliver together with CubicSoft Dynamic-Link Library, DLL, *MySql.Data*. Only in this way the correct management of MySQL database from .NET Framework will be available.

JSON FORMAT

The other principal part for the project is a JSON text format. CubicSoft make a use of *Newtonsof* JSON.NET, a popular high-performance JSON framework for .NET. The newest version available and used in project is 6. The main advantage of JSON are:[35]

- flexible JSON serializer for converting between .NET objects and JSON
- LINQ to JSON for manually reading and writing JSON
- Convert JSON to and from XML
- High performance, faster than .NET's built-in JSON serializers

JSON format is used to interchange data between TimingSense system software. First to communicate between *Connector* (one of the CubisSoft’s modules) and TimingSense equipment (the lector) to receive chips readings in JSON format by use of TCP/IP data transmission method. Then to interchange data between CubicSoft and online platform. CubicSoft can receive from online platform all inscribed athletes who will participate in given race and download the race configuration data. In the opposite direction CubicSoft can send a race configuration data with a final results calculation to online platform. In this part a lot of

data is processed. Then generated JSON text, which has more than 1 MB, is sent thorough network service implemented on the online platform side.

The main three components of CubicSoft was described, but there were smaller parts, which was used in the project implementation process. First there were consulted and found on some developer C# web pages and the reused in the project like C# NTP Client to synchronize Windows pc clock.[36][37]

2.3.2. TIMING METHODOLOGY

First of all to understand the timing software idea, "for what is it?" and "what have to do it?", is indispensable to get to know with terminology and methodology of a race timing. Timing is a method of determining the time required to perform work in our case time to accomplished sport event, race. The whole study associated with the time measurement can be divided into three steps, this idea is also implemented in CubicSoft:

- race preparation phase
- proper observation and measurement of time
- elaboration of results

As was mentioned earlier TimingSense is a complete system created to timing purposes of the high participation non-motorized sports events. TimingSense system is designed to timing cycling, triathlon, athlete races, etc.

RACE PREPARATION PHASE

To proper understanding of the first phase race preparation implemented in CubicSoft is necessary to know how look a typical race from the technical point of view. Let's look at the most popular athlete race which is a marathon. At the principle a ***race*** is a most general term which tell us about a race competition/s in a given day. Then a ***race*** consist of an ***event/s*** which is every single competition in a given ***race*** for example a ***race*** marathon can have two ***events***: the principle race marathon and the race on a distance of 10 km. Then every ***event*** has assigned athletes who will participate in a given event race. There is a possibility that in the race participates ***teams***, every team consists of a given number of athletes and has prepared separate classification. Every ***team*** belong to a given ***team type.*** Moreover every ***event*** consist of 5 basic elements:

- ***splits***, every event consist of several splits which number depends on the race type. ***Split*** is a place where a lector is placed to reads a chip tags of every athletes passing through that place for example in marathon are typically three splits: on the start, 21 km and meta.
- ***waves***, every athlete is assigned to his wave. Normally where is a lot of participants, a race start is divided into two, three or more parts and those parts create ***waves.***
- ***categories***, every athlete is assigned to his category. ***Categories*** can be divided into two parts: the most common "age based" and standard, non age based, category.
- ***attributes***, every athlete can have assigned several attributes like country, medical details, t-shirt size, etc.

The last step in the race preparation is an assignment of athletes to appropriate ***race number.*** During race a race number is what makes a participant unique. What is more every ***race number*** has assigned one or more ***chip tag/s.*** The chip tag is hardcoded in the chip which is carried during a race by a participant usually in the race number.

RACE TIMING PHASE

Then in the second phase which is a timing, during a race some chip is read in our case by a TimingSense lector when athlete pass through a ***split*** and this data is send to a CubicSoft module responsible for communication with lector. In this moment is used RFID technology which is described in the next section. Like that CubicSoft can calculate a classification during a race. Usually in a race day occur many incidents like some athlete disqualification, athlete change race number or start in different wave or event, etc. During the race a timekeeper has to be attentive and reflect every changes in CubicSoft. The process of a race timing in presented software is fully automatic.

RESULT ELABORATION PHASE

The last phase is an elaboration of the results. This part due to CubicSoft is very simple. CubicSoft deliver a two types of result presentation. First, very fast with just one click results generation and printing. Second, more advanced where user can adjust every aspects of a race data representation.

2.4. THE RFID TECHNOLOGY

The radio-frequency identification is a basic technology of TimingSense system. RFID is a wireless non-contact technique which uses a radio-frequency electromagnetic fields to transfer data for the purposes of automatically identifying and tracking tags attached to an objects. RFID's tags contain electronically stored information. The information that chips contain may be read, recorded, or rewritten. RFID is part of the family of Automatic Identification and Data Capture, AIDC, technologies. RFID technology is often a complement to a barcode method.[1]

Figure 4. RFID UHF 2 GEN chip tag.

The history of the RFID start at 1945 in Soviet Union. Léon Theremin invented a tool which retransmitted incident radio waves with audio information. From that day to now RFID technology is more and more accessible with every year because of lower cost of the production.[1] Typically RFID system consist of two main parts: tags and readers. RFID tags contain at least two elements: an integrated circuit for storing and processing information, modulating and demodulating a radio-frequency signal, collecting DC power from the incident reader signal, and an antenna for receiving and transmitting the signal. The tag information is stored in a non-volatile memory. RFID tags can be either passive, active or battery-assisted passive. Most chips used in the race are passive (single-use only) are cheaper and smaller because it has no battery. A battery-assisted passive (multi-use chip) are used in triathlons, duathlon, etc. These chips are located on an athlete's ankles (not in a race number) and are more resistant, more precise. What is more, passive chip tags to start operation they must be illuminated with a power level roughly three magnitudes stronger than for signal transmission.[2] The second element, RFID reader, contains two principle parts: transceiver antenna which transmits and receives radio signal and a reader to filter/decode signal. To sum up an RFID reader transmits an encoded radio signal to interrogate the tag, then the RFID tag

receives the message and responds with its identification and other information as showed at Figure 5 below.

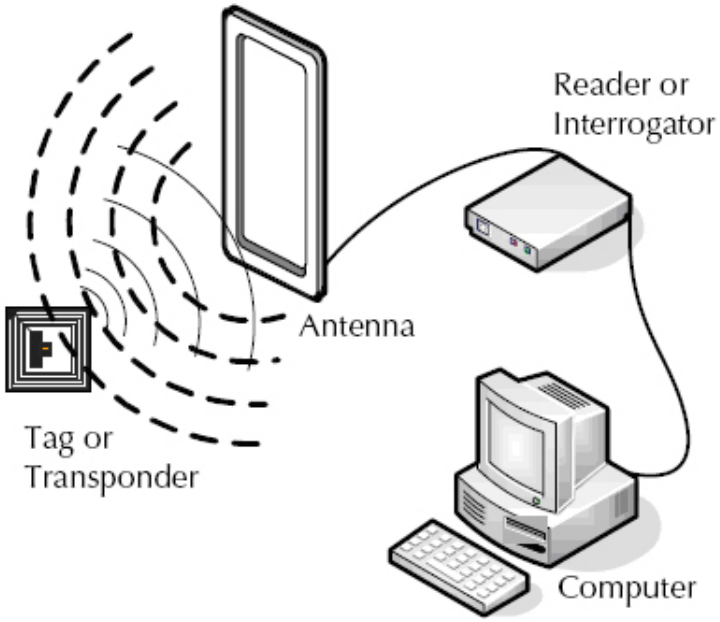

Figure 5. RFID system.

RFID systems can be classified by the type of the tag and reader:[2]

- Passive Reader Active Tag, PRAT, system has a passive reader which only receives radio signals from active tags (battery operated, transmit only).
- Active Reader Passive Tag, ARPT, system has an active reader, which transmits interrogator signals and also receives authentication replies from passive tags.
- Active Reader Active Tag, ARAT, system uses active tags awoken with an interrogator signal from the active reader (battery-assisted passive tag).

TimingSense system contains active reader and can work with two types of tags: battery-assisted passive and passive. TimingSense system work with ultra high frequency, UHF, (according to European standard) the second generation. RFID system frequency and signalling is another aspect which will be discussed. Tags operating on LF and HF bands need to be very close to the reader antenna. At UHF and higher frequencies there is no need to be very close to the reader antenna and the tag can backscatter a signal. In regards to the frequencies, the highest frequency is the fastest transfer and the biggest range, as showed in Table 3.

Table 3. RFID system frequency dependency.[8]

Band	Range	Data speed	Regulations	Tag cost
120–150 kHz (LF)	10 cm	Low	Unregulated	1$
13.56 MHz (HF)	10 cm - 1 m	Low to moderate	ISM band worldwide	0.5$
433 MHz (UHF)	1–100 m	Moderate	Short Range Devices	5$
865-868 MHz, Europe (UHF) 902-928 MHz, USA (UHF)	1–12 m	Moderate to high	ISM band	0.15$
2450-5800 MHz (microwave)	1–2 m	High	ISM band	25$
3.1–10 GHz (microwave)	to 200 m	High	Ultra wide band	5$

RFID technology is used in many area of our everyday life. The RFID tag can be affixed to an object and used to track and manage inventory, assets, etc. for example, it has an applications in areas like: access management, tracking of goods, animals and persons, contactless payment, timing races, etc. RFID for timing races began in 1990s. What is an advantage of RFID in a race timing is that it can provide a race start and end time for individuals in every single race whereas it is impossible to get accurate traditional stopwatch readings for every participant. TimingSense system make use of RFID during the race, the athletes wear tags that are read (in a given spot - ***split***) by antenna placed alongside the track inside the mats. UHF tags provide very accurate readings and needs specially designed antennas. Due to RFID technology rush error, lap count errors and accidents during the race are avoided since anyone can start and finish competition at any time without lose of his result.

2.5. EXISTING SOLUTIONS ON THE MARKET

Now is a time for a revision of all already existing solutions. On the market there are a few applications that can manage a real time measuring. All of the serious solutions which exist on the market are developed pretty well and are commercial products which costs a lot. There was found an open source project *fsTimer* but it is not capable to manage a races with more than 1000 athletes and do not posses logic and well designed user interface. What is more, a bigger part of the existing software on the market is dedicated for a many timing

systems not like the CubicSoft which is the only original existing desktop application working with the TimingSense system.

Not every timing system has its own desktop application to manage a time measuring for example MyLaps (ChampionChip Timing System) and ChronoTrack, companies which will be the main competition for TimingSense do not have such an applications. On the market, there are solutions adjust to a many timing systems like a RaceTec software.[38] It is a powerful tool created in Australia to manage a time measuring, but it uses a bit old technology. As a store engine it is using the Microsoft SQL and it is designed with ordinary windows controllers. Other application which is worth mentioning is a Clascycle from France. This application has a well designed GUI and very simple to manage interface. CubicSoft was designed as a combination of both applications taking the best from RaceTec, its great functionality, and Clascycle, its friendly looking and intuitive user interface. There are many web based timing tools, almost every timing system has its own online platform to manage a race timing. There is also a program like RaceResult 11, which is a web application, which can be configured to work with other timing systems.

TimingSense system in contrast to other timing companies will have its own dedicated desktop application CubicSoft but it also will permit to use other programs like RaceTec due to the additional CubicSoft module *Connector*. The *Connector* permits to send a chip tags readings to a database of any timing software. The limitation is a storage engine as *Connector* supports only the Microsoft SQL and the Oracle MySQL database technologies. Generally TimingSense system in contrast to its competition will have inaccessible for developers a communication protocol documentation so without use of the TimingSense software it will be impossible to communicate with the TimingSense lector.

Personally speaking, is barely possible to find any suitable desktop application on Windows platform with friendly looking interface and functionality which has CubicSoft. Generally, there are better and worse application to manage a time measuring but any of them is a part of a fully integrated timing solution for timekeepers like the TimingSense system with the CubicSoft software.

3. SOFTWARE DESIGN AND MODELING

Taking into account general requirements given by Cronochip company, software design was divided into three individual stages of work. Each stage of work has a specific level of difficulties and a duration time. Consequently, a first step was an user interface design, GUI, of the main application CubicSoft. In the next step was a database modelling. Then there was a modelling and a design of additional modules such as *Connector* which is cooperating with the main software CubicSoft.

3.1 GRAPHICAL USER INTERFACE DESIGN

The design of the user interface is a crucial part of the software engineering. For a developer, the implementation part is vital but for ordinary end-user the most important is the functionality and the appearance of the program, it means GUI. An user interface is an interaction between user and computer, more precise between application and operating system.

First step in the design of CubicSoft was an analysis of existing timing software's GUI and consultations with the best Spanish timekeepers. CubicSoft GUI was an essential concept of this work, a creation of the most efficient and the simplest in use user interface. One of the first software documents created was a wireframe of the CubicSoft application. A wireframe was designed due to Balsamiq web tool.[39] In this way first assumptions and main functionality was noted. During all process of software implementation there were introduced many changes but a main view of created wireframe was preserved.

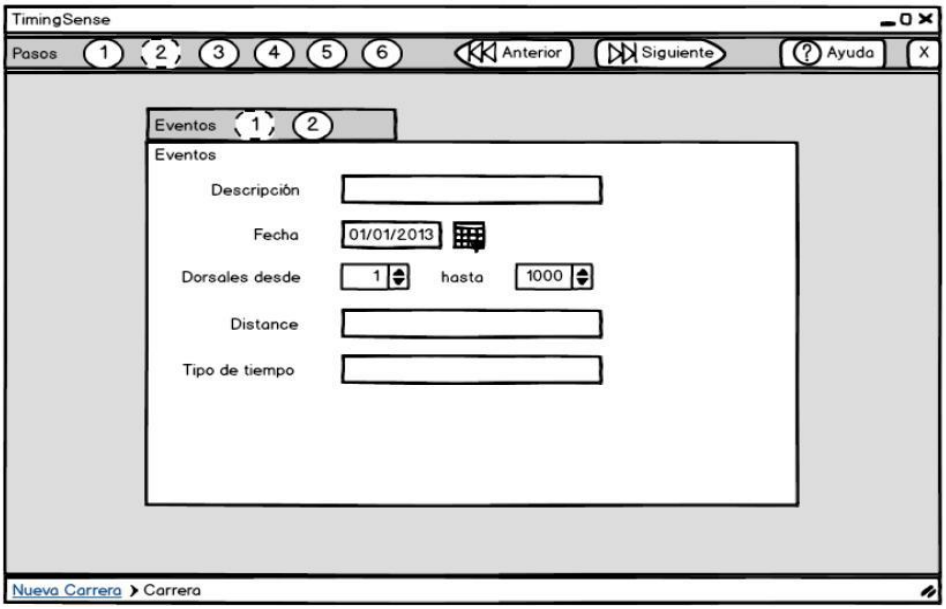

Figure 6. First draw of CubicSoft application – wireframe of wizard window.

The application main control was divided into one ribbon style top menu and four submenus. First is a race preparation phase where all race configuration data is stored. In this place was designed a special kind of wizard which help a user to create new race configuration with just only few steps (see Figure 6). The second part of the main menu is a result calculation, timing, phase. In this part user can manipulate the time measurement during the race and modify the race data if it is needed. The third section of the menu are results/reports where user at final stage of the race can prepare special kinds of reports with race classification. The last element in the menu is online platform section which is responsible for communication with online server which store vital race data. Apart from main menu there is a top bar menu which provide fast access to the most important functionality plus settings and tutorial. There is also a special kind of top menu which give a user an access to the additional modules of CubicSoft like "*Connector*", "*Lector*" or "*Fair*". In the bottom there is status bar containing information like actual status connection to database.

CubicSoft is an application based on Windows Forms. As was stated before Windows Forms are part of .NET Framework and provide access to the native Microsoft Windows interface. To design better looking and more user friendly interface was used additional commercial set of controllers DevExpress which overwrite standards Windows Forms. Except from the main bars' menus used to facilitate the control of application, there are other controls, which help to handle the application and contribute to GUI interface, all are kind of common Windows controllers like buttons, checkboxes, text boxes, labels as well as dialog boxes to deal with exceptions, file dialog or browser dialog to select files or directory respectively, panels and group panels, etc., and finally data grid controller, rich text editor controller and reporting controller, in general the application consist of all it. What is interesting about design and user interface of presented software is it that the interface can be adjusted to the user preferences. For the final user convenience themes can be changed to one of 11 available especially prepared layout designs, all grid data layout can be save and store to the future use, all layout main settings are stored in the XML files.

An user can control the software by use of mouse movement or keystrokes with computer keyboard. CubicSoft was designed to give access to some default functions by use of shortcuts like typical operation "copy & paste", "delete", etc. For user convenience also some of a grid data controllers has implemented "drag & drop" functionality. On the other hand, CubicSoft GUI was prepared with a view of a multi-language application. The software

GUI facilitate the translation process and in every moment even by final user it can be translated to other languages, the translation is stored in XML format file. Basically the application supports two languages: Spanish as default and English. It is vital feature of this software as in the future the TimingSense system will be sold abroad.

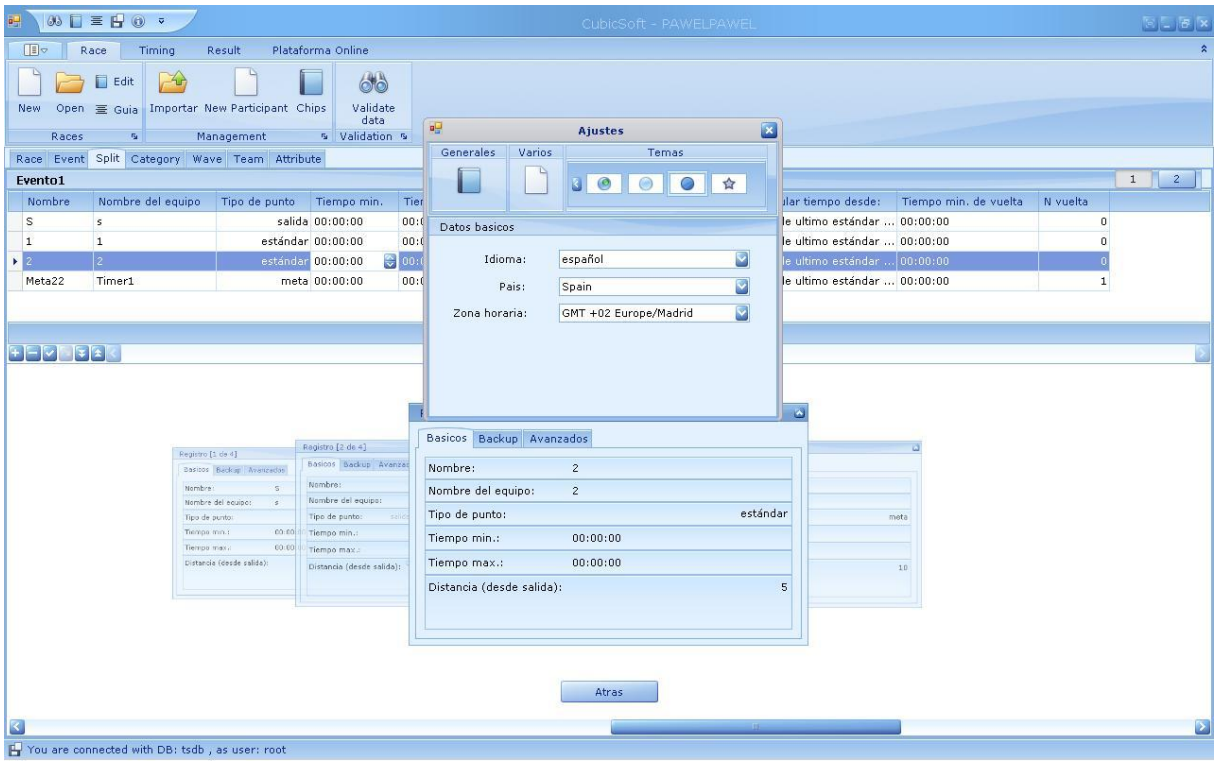

Figure 7. CubicSoft GUI, race configuration data edition and setting window.

3.2 DATABASE MODEL DESIGN

The key part of the CubicSoft program is a database and a fast management of it. Firstly, every aspect of the future database was discussed and compared to the competition program RaceTec, which has a huge database schema, which CubicSoft had to reduce and made more efficient. There was two main ideas during the process of design of a new database: reduce to minimum number of tables and connections between them, and reduce the size of memory occupied by every table so a data type for each column was chosen properly to do not waste space and memory.

The next step was a design of entity relationship database model for it was used MySQL Workbench IDE created by Oracle to manage the connection and design of MySQL database.

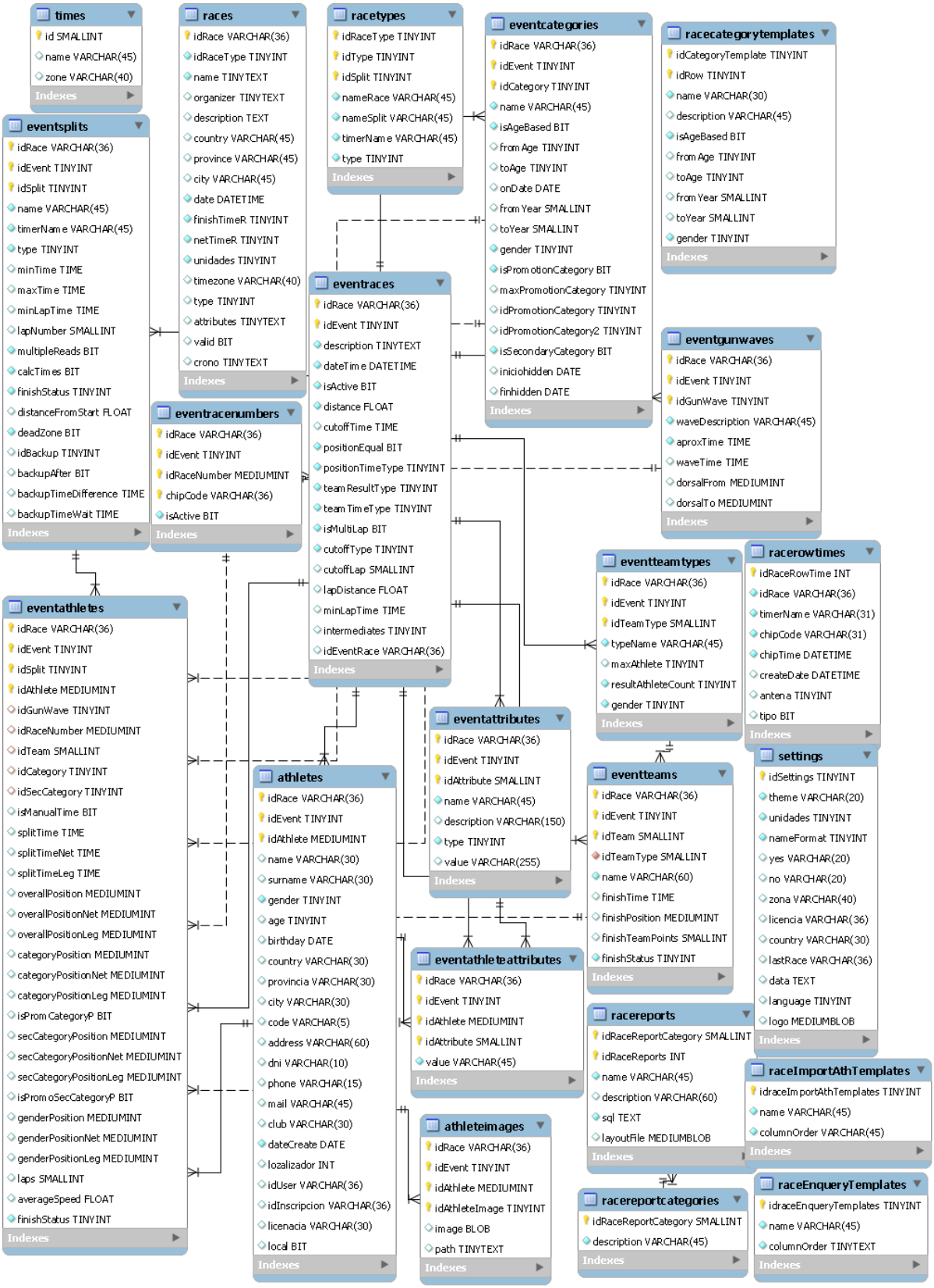

Figure 8. CubicSoft ER database model

THE DATABASE MAIN STRUCTURE

The presented database model consist of 22 tables. The main part, skeleton of the data base creates 13 multiple primary keys tables which are in relation to each other. Main table "***races***" contains all elementary data about created races, each race has his own unique primary key ***idRace***. Each race consist of events, as was discussed earlier, which are contained in "***eventraces***" table. Every event in a race has his own unique primary key ***idEvent.*** Then by ***idRace*** and ***idEvent*** are identifying other elements of a given race event, it is: "***eventcategories"*** (store all events' categories)***, "eventsplits"*** (store all events' splits - control points)***, "eventteamtypes"*** (store all events' teams type)***, "eventteams"*** (store all events' teams which belong to a team's types)***, "eventattributes"*** (store all events' attributes)***, "eventathleteattributes"*** (store all events' attributes with its values for every athletes)***, "eventracenumbers"*** (store all events' race numbers with chip tags)***, "athletes"*** (store all events' athletes)***, "eventathletes"*** (store all events' athlete for each split with race times and positions)**.** These tables store all important data to prepare race and manage the time measuring during competition. Last two tables ***"athletes"*** and ***"eventathletes"*** are multi records tables which means that it contains a lot of records for every race, for example a race of 10000 participant has 10000 records in "***athletes"*** table whereas ***"eventathletes"*** has more records ("***athletes"*** ***** events' splits) for 3 splits in one event it gives *30000* records.

THE CHIP TAGS READINGS TABLE

Apart from main tables there is another table which is the most heavily loaded during the race "***racerawtimes***". In this table is stored all data send by lector to CubicSoft, it contains columns: race name, split name, chip tag, chip time, antenna type and reading type. Taking into account that the "***racerawtimes***" table is heavily loaded, the proper chose of data type for each column is vital for example in a race of 10000 athletes, 3 points of control during the race and assuming that each athlete has 2 readings on one split and has double chip tag, in total it gives: *10000 * 3 * 2 * 2 = 120000 records* (one "***racerawtimes"*** record occupies **121 bytes**). This huge amount of data has to be inserted and read from database to further analysis, so every milliseconds of latency and byte of memory is important. The more detailed analysis of database memory/space usage is presented in the next chapter where is described special mode of MySQL database, Cluster, used to result calculation which store a data in memory in place of a hard disc.

REST OF THE DATABASE TABLES

Rest of the tables presented in database model store the template data like: time zones, countries, province, cities, category templates, split templates, race types, race import athlete templates, race enquiry templates, race reports and race reports category templates. The table "***settings***" store all program configuration data like actual language, actual theme, etc. The table "***athletetimages***" will be used in the future to store images for each athlete.

3.3 CUBICSOFT SOFTWARE MODEL

This section presents a model and a design process of CubicSoft application and it describes a main features of the program. The presented software first was modelled in Eclipse Ganymede using Ecore modelling tools and creating UML class diagram.[40] When dealing which such a big and complex project, it is impossible to start the implementation phase without creating main view, sketch up of the program. The first draw of the application was very useful in a design and then in setting up a main program functionality. The diagram model during whole process of a software creation has been modified and it has evolved. The final stage of the class model view of the CubicSoft software is presented below (it was created in Visual Studio 10).

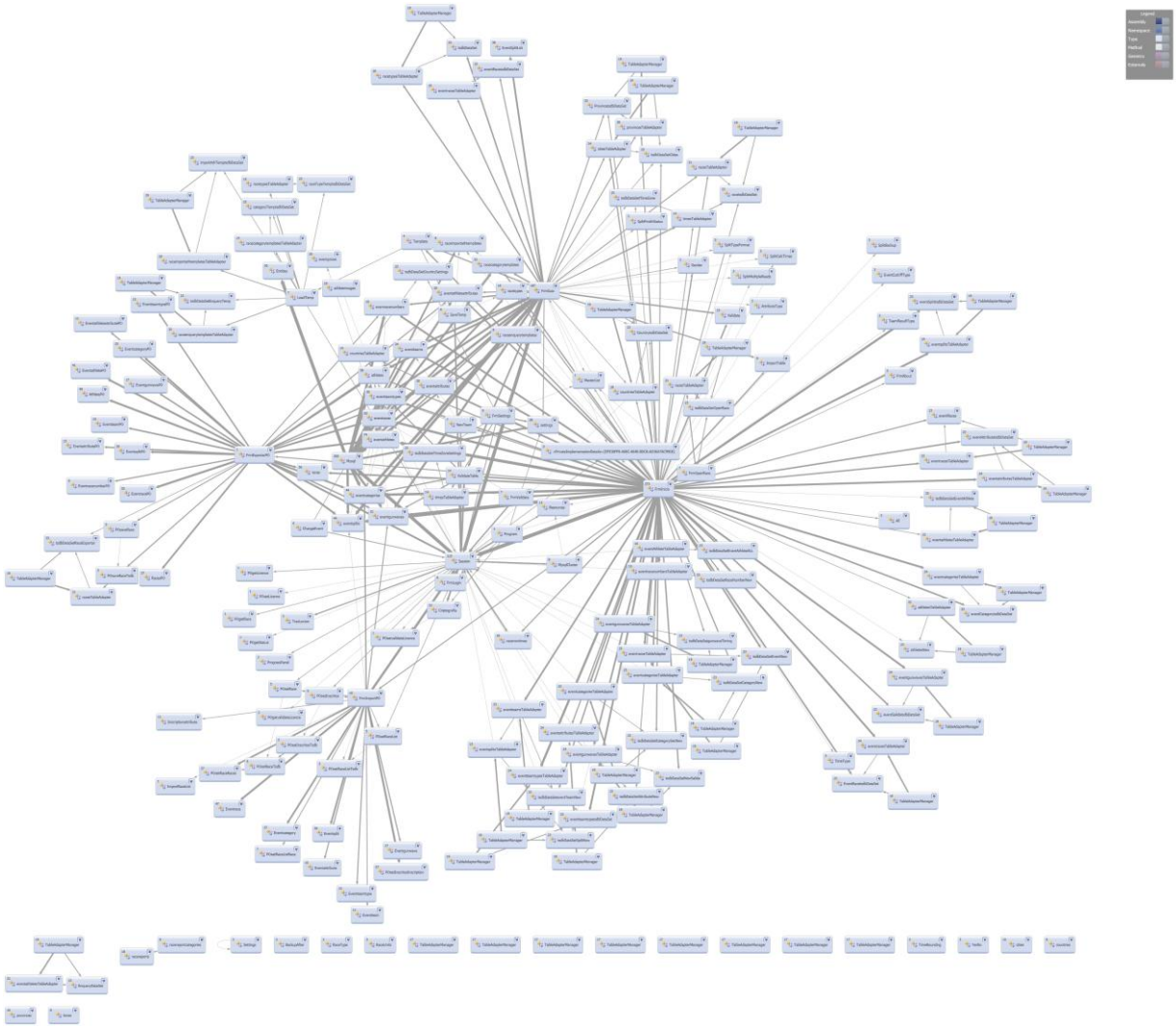

Figure 9. CubicSoft class model view

The main class visible at the diagram in the middle is "***FrmMain***" which is the main application window form that contains almost all important features of the program like race preparation, race timing, and reporting. Then at the top there is a class "***FrmGuia***" which is the wizard which user employ to prepare all data needed to race timing. On the left hand side there are two classes '***FrmExportPO***" and "***FrmImportPO***" which are responsible for communication with online platform to import and export, interexchange, data. Other crucial classes is "***Session***" which contain all data needed in current race session, "***MySQL***" is a class which defines all connection with MySQL database, "***Entities***" is a class generated by Entity Framework which maps whole entity relationship database to a classes. There is also many auto generated "*TableAdapter"* classes by LINQ technology used to fill grid data controller with data from MySQL database.

Main application window Form contains *ribbon menu* and *status bar* which provides fast and easy to manage access to given part of program. All of it is designed in a logical way that the user from the first view of the software know how to manage it. Below are presented and detailed described three basics elements of the application in order of the timing definition: race preparation, race timing and analysis and reporting.

3.3.1 RACE PREPARATION PHASE

The first step which user has to do is prepare all data needed before race timing will be started. For this purposes CubicSoft has designed two types of form: special kind of wizard form which facilitate a timekeepers to prepare correctly a race and setup, a form where user is not guided but can perform more advanced operation not permitted in wizard.

RACE CREATION - WIZARD

Wizard is divided into 10 steps, every step with error validation, at last step user can be sure that created race does not contain any serious errors. First step is an introduction of basic information about race like race name, race date, country, city, etc., new race can be created by coping old race (using race templates) or manually giving number of events and using split templates. There is used a mix of DevExpress components which facilitate the software management and provide nice look design. Another steps starting from events, splits, categories, waves, teams types and attributes configuration (all data is modified in potential DevExpress data grid controller) and ending on chips and athletes importation, and result calculation adjustment. All of the steps in wizard are design with high attention to timekeepers comfort and usability so with every step introduced data is validated and user has many functional buttons (save/load template, add, delete record, etc) and shortcuts to manipulate data like "***copy & paste***" or some additional functionality like "***drag & drop***" items into grids. What is important user in every moment can exit from wizard and come back to it but only in the same session. After 10-th step the user finish wizard and come back to main window which contains setup of created race.

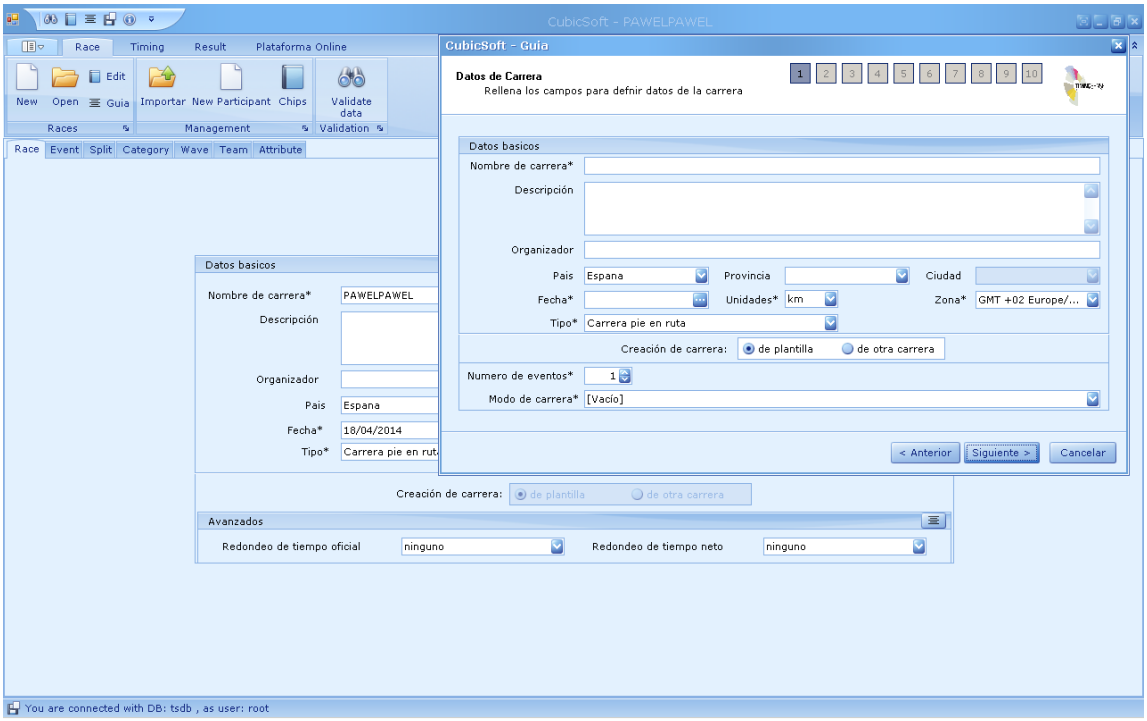

Figure 10. CubicSoft - race preparation view including wizard window

RACE CREATION - SETUP

Setup consist of ordered steps from wizard displayed in tabbed group controller. User can pass through one tab to another one fluently, the process of validation is also present but only in form of pop-up notification in the left bottom corner if some error has been encountered. Most of the data is displayed in a grid data controller and in edition mode special carousel display mode is activated. Moreover DevExpress has special kind of controller "*repository item*" which helps to present data in a grid controller in more user friendly way, for example using icons or some graphics.

Another elements of race setup is an importation of athletes and chips. User can import data from file, available formats .xls, .csv, .txt, other races or online platform. Online platform has separate menu section, the communication between CubicSoft and online platform is bidirectional but from assumption race participant are enrolling by web page and then all data is downloaded by CubicSoft. All imported data athletes and chips are presented in separate data grid controller for each event. Additionally user can add participants manually using cleverly design form and verify correctness of introduced data by using verification windows form. Every encountered error is listed and by one click user can get to the place with erroneous data. The process of validating data is a more complex task which is

divided into three main parts: basic race configuration (where splits, categories, attributes, etc. are checked), athletes (where all athlete information accordance is checked) and race number (where all race numbers and chip tags are checked). The last item of race preparation section is race management where user can select from the filtered and sorted grid earlier created race and open it.

3.3.2 RACE TIMING PHASE

During the race day an user makes a use of a timing part of the application. This part was basically simple in the design. A main feature here are *enquiries* design as a grid controller where user can manipulate all athletes data and times, it also has a simple module to generate fast report from the grid view. In this area timekeeper can manage all race data necessary during competition. The discussed grid delivers a pallet of useful tools like sorting, filtering (by columns and creation own complex filters), multiple column grouping, advanced search based on "*like*" method and column selector to adjust the grid view to the user needs. The grid posses own context menu to apply any required operation during a race like edit athlete, change race data or export grid to the one of the available formats.

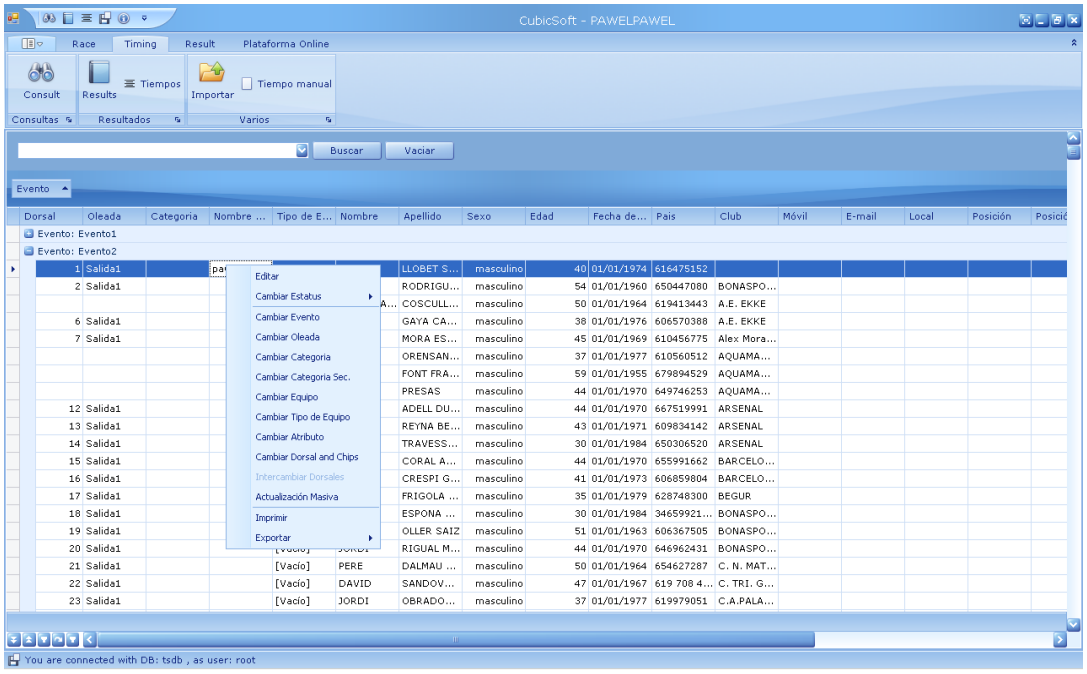

Figure 11. CubicSoft - enquiries view

Other element is a *result calculation* module. Here user can start the result calculation by a single button click (gets times reading, analyse data and calculate positions) and adjust the advanced options like: gun time calculation, net time calculation, team position calculation, etc. Additionally there is a possibility to initialize a new window form visualising a live results which is still in developed. Next element of race timing is a *start time* where timekeeper in a moment of race starting has to assign a race start time to each wave – start group. In a design of race timing section there is a place for manual adding athlete race time and editing/importing race raw times. Especially, importing race raw times directly from file is sometimes a lifeline when communication between CubicSoft and TimingSense lector is not working so the only possibility to get race raw times from TimingSense lector is importing back up files from lector directly in the CubicSoft due to specially designed form where all race raw times are presented in the data grid controller.

3.3.3 REPORTING PHASE

The last element is a report preparation and generation. User has two modes to prepare reports: one is simpler and faster and second is for more expert users. The first one is for not advanced users with friendly GUI where due to DevExpress controller user can "***drag & drop***" chosen columns to the prepared before layout and can print it or export to the one of the available format (.xls, .html, .pdf, .jpg, .txt, .csv). The second, for expert users, inserts specially prepared sql enquiry to chosen layout. Thanks to this option timekeepers can create every required report. This part was designed in a way to give an user full control to report preparation and generation process with the most friendly possible user interface.

The layout of the report is created in a Microsoft Word like document which give a user an access to many formatting tools to create perfect report layout including objects from external sources like images. Every generated report can be saved with a specific report number in a template table. What is important for a timekeepers there is a possibility to exchange prepared earlier reports with other timekeepers from the same company due to online platform where user can upload or download all available reports.

3.4 CUBICSOFT ADDITIONAL MODULES

In this section are described all additional CubicSoft modules created to improve the functionality and give some extra features to the program. The most important module is a "*Connector"* which is a middleware between TimingSense lector which stores all readings and CubicSoft which analyse all received data. "*Connector"* is a principle module without it the whole TimingSense system cannot exists. Other module is a "*Lector programmer*" which permits to program chip tags before the race. The third module is a "*Fair*" which is used during race fair before a race where athletes come to collect its race numbers. This module helps to assigned automatically to athletes a race numbers which will identify every participant during the competition.

3.4.1 THE CONNECTOR MODULE

Connector is a small and complex program which deals with connection with TimingSense equipments or online platform (if equipments are connected directly to online platform) to gets readings and sends it to a CubicSoft database. *Connector* module was as a first part of the project terminated and tested in a real race.

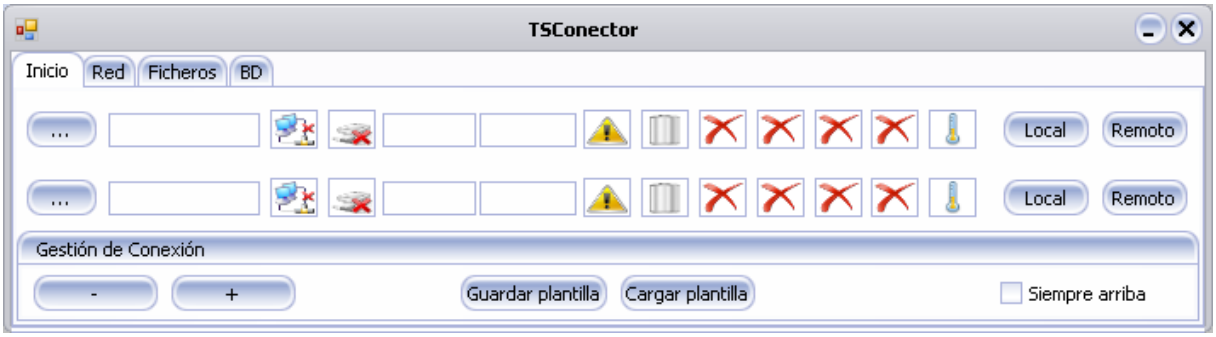

Figure 12. CubicSoft Connector module – main view

THE CONNECTOR MAIN STRUCTURE

Connector was first discussed with a timekeepers and then was created a general wireframe. Finally *Connector* was modelled like CubicSoft by Eclipse Ganymede using Ecore modelling tools. *Connector* was designed in similar methodology as CubicSoft. The main part of *Connector* are DevExpress controllers (like tabbed group form). The principle window form consist of four tabs, general *Connector* configuration:

- home – where all connectors are listed in graphical form, icons are indicting status of connection with TimingSense equipment. There are also special kind of DevExpress buttons to activate/deactivate connection. The whole connection configuration can be saved or loaded to/from template stored in .xml file format encrypted by ***RIJNDAEL 256***, standard algorithm of .NET Framework.
- network – in this tab user can edit network configuration of his own workstation or general configuration like IP address to communicate with TimingSense equipment
- files – this part serve as back-up system, when *Connector* lose connection with database always all readings send by TimingSense lector are stored in file in one of available format .csv or .txt. User can easily activate or deactivate the back-up mode
- database – this is crucial part where user has to introduce all database connection data and choose the race for which the readings will be send by lector. User can connect to CubicSoft database or one specified by himself, available are two types of database MySQL or Microsoft SQL.

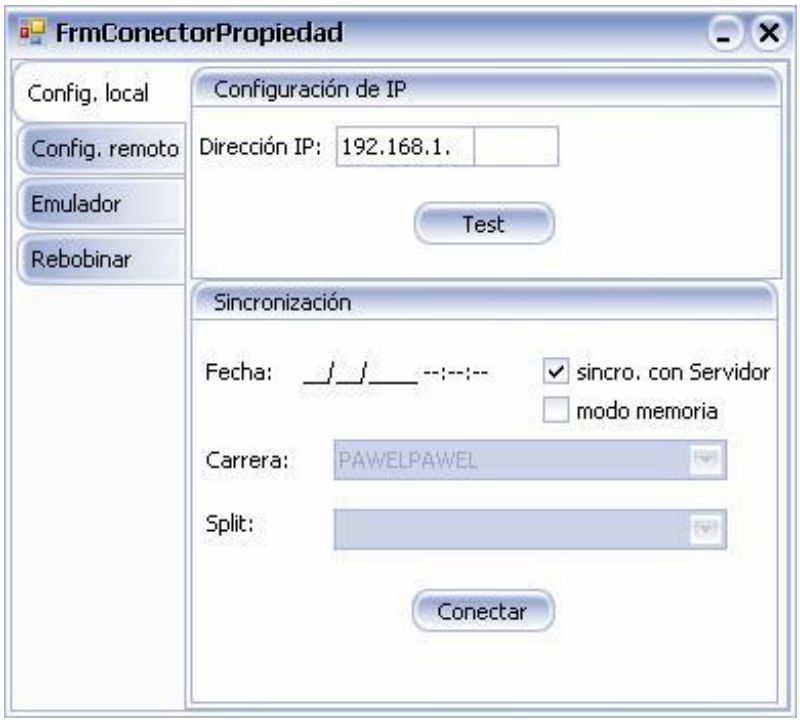

Figure 13. CubicSoft Connector module – property window

THE CONNECTOR COMMUNICATION METHODS

The last step is to connect all TimingSense equipment with a connectors listed at the main window (every connector has to be configured, every connector is working on separate thread). User can connect with TimingSense equipment by means of two modes: first is a local connexion by inserting proper IP address and synchronizing PC and TimingSense lector with the same time. User PC clock is synchronizing always with Network Time Protocol, NTP, pool and then this time is send by TimingSense protocol, described more precise in implementation chapter, to TimingSense lector. NTP is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. NTP pool where user PC is connecting is a big virtual cluster of timeservers providing reliable NTP services, PC always is connecting with the closer server available depending on its location.[41] The second mode is a remote connection. TimingSense lector is connected in cloud normally with online platform and Connector to grab readings connects remotely to online platform to get data. This type of connection is realized by web service lunched on the online platform side. There is an option to simulate reading and send data manually to database and the last element is a "*rewind*". "*Rewind*" tab is a vital element of *Connector* where user can get in every moment (rewind process is working on separate thread) any readings available on TimingSense equipment side. *Connector* communicate with TimingSense lector and receive asked readings. Thanks to DevExpress's controllers the *Connector's* forms look pretty and are intuitive in use for final users.

3.4.2 REST OF MODULES

THE LECTOR

Lector programmer is a module which permits to program chip tags which are used in a race. *Lector programmer* is compatible with HBR-D406-E USB Reader and supports other devices from this family. The hardware supplier delivers also WL-RFID105 demo software with a documentation. *Lector programmer* was designed by use of DevExpress controllers and then implemented by use of delivered WL-RFID105 demo software with ZK_RFID105CSharp dynamic link library designed to facilitate EPCC1-G2 and 18000-6B protocol UHF tag application software development.

Figure 14. HBR-D406-E USB Reader

Lector programmer first instruct the user to connect with USB reader due to ZK_RFID105CSharp dynamic link library by COM port. Then user can perform two operation: write chip tag or read chip tags. To write chip tags user first has to import a file .csv with a list of race number and chip tags into a grid data controller then the process of writing chip tag can be started by just simple button click. User put the chip near the USB reader and *Lector programmer* automatically assign a new chip tag to given chip. To read the chip tag user just put the chip near USB reader and *Lector programmer* save the given chip tag in the grid data controller. When user terminate reading/writing chip the result grid can be exported to .xls excel file.

THE FAIR

The next module is a *"Race Fair*" which is used during a race fair where athletes come to collect its race numbers. This module helps to assigned automatically to athletes a race numbers during fair which will identify every participant during the competition. Usually timing companies use its own software to assign race numbers to athletes during a fairs and then the exports result to a file which then is imported to manage time measuring software. Timekeepers just assign an athlete to a race numbers which is stored directly in CubicSoft database so there is no additionally work needed.

Thanks to all modules TimingSense system has fully equipment system with all software integrated in one place. TimingSense system with CubicSoft and all its modules was designed and created with idea of giving a fully functional software system with no need to use additional tools.

4. SOFTWARE IMPLEMENTATION STAGE

This chapter illustrates the implementation process of earlier designed stages. The implementation process presents the problems, which was needed to face up and then overcame. This chapter omits an ordinary process of code writing which not give any conclusions or problems. At the end of this chapter is emphasized the versatility of the implementation and a direction of further development of the application.

At the beginning is worth mentioning that all project from the start point to nowadays is stored in cloud like *Dropbox* and SVN servers. *Dropbox* is used more to give an access to all timekeepers dealing with and testing TimingSense system. In this way the communication between CubicSoft programmer and Cronochip workers was established. The most important part is SVN which is a software versioning and revision control system. Developers use Subversion to maintain current and historical versions of files such as source code, documentation, etc. CubicSoft project is kept on *Assembla* SVN server.[40]

Other important aspect worth mentioning before the full implementation process will be presented is a use of *Redbooth* (formerly *Teambox*) web-based collaboration tool. Due to this tool all process of implementation TimingSense system was described in detailed tasks with deadlines assigned to a particular persons dealing with TimingSense. *Redbooth* web page tool caused that the process of TimingSense creation and CubicSoft implementation was in order and all modules was done on time. *Redbooth* arrange work according to Gantt diagram.

4.1 CUBICSOFT DATABASE POPULATION

Database population concept corresponds to filling a database with a race configuration data. The implementation process presented here consist of a coding all main functionality of CubicSoft race preparation section. Most of this process is an insertion of a few records to the database by means of Entity Framework which seems to have pretty well efficiency for this task. Entity Framework (LINQ to Entity) has two main advantages is easy to use for programmer and gives satisfactory results of insertion of a few records to a database for a final user. Entity Framework has advantage over LINQ to SQL, it allows for a broad definition of object domain models and is a bit faster and works with non Microsoft database like MySQL.[44]

The problem occurred when CubicSoft start to dealing with a multi record insertion like importation from file athletes with all corresponding data/attributes or importation of race number with corresponding chip tags. The same problem occurred when CubicSoft was

dealing with importation data from online platform. In this case an appropriate tests was done to calculate the speed of insertion multiple records to MySQL database be means of using *System.Diagostics* .NET namespace.

THE PROBLEM OF INSERTING A HUGE AMOUNT OF DATA TO THE DATABASE

The test was conducted on "***eventracenumbers***" table which consists of 5 fields in total ***79 bytes*** per one record. Typical test performed an insertion of 10000 race number with double chip tag so in total it gives 20000 records (20000 * 79 bytes = 1580 KB of data). Entity Framework cannot perform this operation with satisfactory results. In the default configuration of Entity Framework the discussed insertion take more than 7 minutes. Entity Framework dealing with more advanced configuration and disposing *ObjectContext* gives the best result almost 20 seconds but it is also not satisfactory. The solution was found after brain storm. Microsoft SQL has a special type of fast insertion of many records BULK INSERT, unfortunately MySQL do not has implemented such a method but has very powerful family of INSERT statements like INSERT IGNORE, INSERT ... ON DUPICATE KEY UPDATE, etc. Finally a huge INSERT IGNORE query was generated by processing whole imported file and inserting it to MySQL database by use of native connection. The result time was impressing, approximately 2 seconds, and 10 times faster than Entity Framework in the best configuration. In a CubicSoft software Entity Framework is used to insert a small number of records less than 100 at the same time, in other cases is used INSERT IGNORE query with native connection to MySQL database.

Table 4. The test result of insertion 20000 records to "***eventracenumber***" database table.

Description	Speed in Milliseconds
Entity Framework - insert all records at once without dispose *ObjectContext*	almost 7 minutes
Entity Framework - insert all records at once and dispose *ObjectContext*	21048.4575 21152.943
Entity Framework – insert every 1000 records and dispose *ObjectContext*	21736.762 21275.0055
Native Connection – insert all records in one INSERT INTO query	2057.4855 1998.855

RACE NUMBER AND CHIP TAGS IMPORTING PROCESS

Now the all process of importing data from the file to query generation will be presented, it is one of more complex task in CubicSoft software. First a list of race numbers is imported from one of the three types of file format: .xls, .csv or .txt with tabulation. An user have a special button to select file or can just ***"drag & drop"*** file into the grid data controller (race number also can be copied from another event from the same race to the new event). The process of race numbers importing is much simpler than athletes importing. After importing file user has to assign column in grid controller to appropriate column in a file, race number to a race number and chip tags to a chip tags. Then the process of importing is started, all document is processed and query is generated with every read of file line. The last process is an injection of generated query to the database.

ATHLETES IMPORTING PROCESS

The process of athletes importing is more complex but also fast. First importing data from file (name, surname, birthday, etc. and other attributes and information like: team name and team type, wave, category, etc.) has to be assigned to adequate CubicSoft's columns (athlete name to a athlete name column in a file, etc). This process is a half automatic because CubicSoft is reading all columns header from the file and if it find a one with the same name as grid column, assigned it automatically. Athletes can be imported with race number or without it. When athletes are without race number CubicSoft can auto-assign race number or leave athletes without race number. The next step is an assignment of unique key to one of the imported column by default race number is a unique key column, every imported athletes have own unique key. If user imports athletes without a race number a special numeric column has to be assigned as a unique key. Then if an user is importing another file and new athletes have an unique key which already exists in a database, user can ignore or overwrite them. Then all document is processed and query is generated and inserted to the database. For every importing athletes is used a following methodology:

- if an athlete will be imported with a race number CubicSoft checks if the race number is a numeric filed or create one automatically.
- the name and surname attributes are formatted to a given by a user style: upper case, lower case, etc.

- the gender attribute (male, female, mix) is assigned to the athlete (if specified gender name does not exists the athlete gender information will be corrected).
- the birthday attribute, if is given, is checked and converted to a MySQL date format (mm-dd-yyyy) and age is calculated (or reverse the age is given and the birthday is calculated). Then the athlete is assigned to an appropriate category (if there exist "age based category" and birthday is in the given category range).
- if the athlete is not assigned to any "age based category" or in the file is specified the category/secondary category name, the athlete is assigned to it (if category/secondary category name does not exists in given event the athlete category information will be corrected).
- if a given event has a team type created and the athlete in the file has specified team type and team name this data is also imported (if a team name for a given team type does not exists, is created, and if team type does not exists in given event the athlete team information will be corrected).
- the wave is assigned to the athlete, if in the file is specified the wave name the athlete is assigned to it (if wave name does not exists in given event the athlete wave information will be corrected). The other option to assignee the athlete to a wave if the name was not given in the file is an assignment by a race number. If a waves have introduced a range for race number (from race number to race number), the wave can be assigned according to the athlete race number and the wave range.
- the additional information like city, country, id, etc. if given in a file, are assign to the athlete.
- if in the given event exists extra attributes and the imported file specified some, first the attribute value is verified (only *Boolean*) with attribute type and if it is correct, is assigned to the athlete (if attribute is not valid the athlete attribute information will be corrected)

The last process is an injection of all processed data to the MySQL database and eventual correction by the user of all wrong imported data (gender, wave, category/secondary category, teams type, attributes *Boolean* type).

The presented process of importing athletes seems to be very complex but due to great logic of processing file and fast MySQL INSERT statement, user even not notice when the process starts and ends. The full test of the importation of 10000 athletes with full information

and 10000 race number with single chip tag and one additional attribute in a race of one event with 2 splits gives in total 10000 records inserted to "***athletes***" table (10000 **** 514 bytes*** per record = 5104 KB), 10000 records inserted to "***eventracenumber"*** table (10000 **** 79 bytes*** per record = 790 KB), 10000 records inserted to "***eventathleteattribute***" table (10000 **** 88 bytes*** per record = 880 KB) and 10000 athletes * 2 splits = 20000 records inserted to "***eventathlete***" table (20000 * 80 bytes per record = 1600 KB) in time of less than 20 seconds.

4.2 COMMUNICATION WITH TIMINGSENSE LECTOR

An important part in TimingSense system during a race competition is CubicSoft module *Connector*. As was described earlier *Connector* is responsible for a communication between TimingSense lector and CubicSoft software and insertion of a received chip tags readings to database. *Connector* can receive readings in two ways. First by use of remote connection, TimingSense lector is sending reading to online platform and *Connector* connects by use of web service working on the side of online platform and downloads readings. In the moment of writing this, given kind of connection is in a phase of development, not working yet. Second by use of local connection where TimingSense lector communicate directly with *Connector* by standard TCP/IP socket using own protocol presented later. *Connector* is implemented to works not only with CubicSoft but with other programs to manage time measuring like RaceTec. *Connector* permits the connection to Microsoft SQL and MySQL database. This solution is prepared to not obligate TimingSense's clients to use CubicSoft if they prefer their own software. *Connector* is a crucial tool and every data *Connector* received is stored in back-up file in the one of the available formats: .csv or .txt.

THE CONNECTOR SYNCHRONIZATION PROCESS

Connector can manage infinite number of connection with TimingSense equipment, the only limitation here are the parameters of user workstation. Every local connection has to be configured by introduction of proper ip address. Then before connection is established the timekeeper pc is synchronizing clock with NTP server, http://www.pool.ntp.org. *Connector* needs the administration rights to be executed, the reason of it is a necessity of modifying windows registry and setting direction pool.ntp.org as a default pc NTP server. The synchronization with NTP server is realized by NTP Client class using standard System.Net .NET Framework namespace.

THE CONNECTER COMMUNICATION PROTOCOL

The next step is an establishment of connection with TimingSense lector by means of specially prepared protocol with use JSON text format technology. Due to *Newtonsoft.Json* class an objects send by communication protocol are serialized and deserialized. TimingSense lector is active when equipment is turn on. Lector open TCP/IP socket and takes the server role, remaining in listen state to connection requests. There are three sockets: data socket (to interchange readings), control socket (to inform about TimingSense equipment state) and rewind socket (to receive specific readings). At the beginning *Connector* set up connection on a control socket by use of a four way *handshake*. *Connector* with TimingSense lector interchange four packets: "***hola***", "***syncrequested***", "***sync***" (packet with the actual lector time) and "***carrera***" (packet with a race configuration data). For example the last packet contains a name of the actual race and the point of control configured on the TimingSense lector and it has a given form:

{"tipo":"CARRERA","carrera":"xxxxx","split":"yyyyy"}\n

After this quick handshake the control socket is opened and *Connector* is listening for any information from lector about TimingSense equipment state. *Connector* also established connection on data socket with lector and is listening for new readings. A packet containing a chip reading in JSON format is presented below *w*here all labels are explain in Table 5:

{"t":1,"h":"20061230T14:05:20","c":39,"a":1,"x":"04","s":"split","r":"race","u":"user"}\n

Table 5. Explanation of all labels used in JSON chip reading packet

Parameter	Description
t	reading type (integer)
h	hour (date time)
c	milliseconds (integer)
a	antenna (integer)
x	chip (string)
s	split (string)
r	race (string)
u	timekeeper (string)

Every packet interchange between *Connector* and lector always contain a single information, not in a list form. The last rewind socket can be used when two other sockets are opened. *Connector* sends JSON packet "***rebobinado***" which contains a SQL query to be executed on the lector side. In the response *Connector* receives a number of returned records with error text message and rewound data. More technical information about interchange information between *Connector* and TimingSense lector can be found in Appendix 1.

4.3 RESULT CALCULATION STAGE

The result calculation section is dealing with a management of race time measuring during the competition. The most important elements are: enquiries where user can manage and modify all race athlete data and result calculation module which performs every second overview of "***racerawtimes***" database table to looks for new chip tags reading and conducts the result calculation. Two speed up the result calculation process a special kind of database was used, MySQL Cluster (to put all data in memory RAM). These two elements are indispensible in a race timing, it is a CubicSoft brain, and were pretty complex in implementation stage.

RACE DATA MANAGEMENT – ENQUIRIES

First will be discussed enquires part and methodology used to manage it. In a potent DevExpress grid data controller are displayed athlete data with all necessary information. The data loaded in the grid is a combination of JOIN (OUTER LEFT and OUTER RIGHT) queries of 6 tables: "***eventathletes***", ***athletes***", :***eventsplits***", "***eventgunwaves***", "***eventcategories***" and "***eventteams***". Whole enquiry in Visual Studio Query Builder is presented in Figure 15 below.

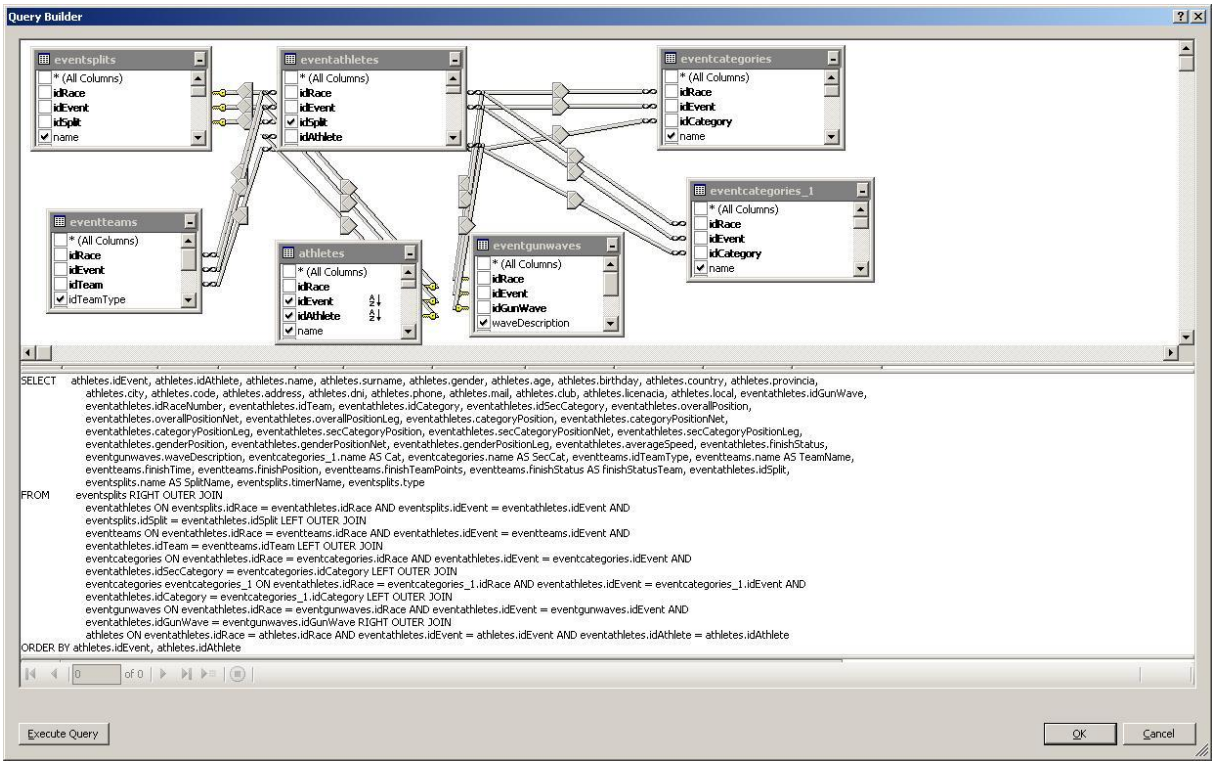

Figure 15. Microsoft Visual Studio Query Builder – view on the enquiries model

User can modify and manipulate all data showed in the grid. User can make use of helpful tools which grid delivers like grouping column headers, filtering by column or used filter query builder to build some expressions, multiple column sorting, advanced search using "*like*" method. What is more user can adjust the grid layout by use of column selector or possibility to load/save layout templates. The implementation methodology process of modifying athlete data by grid context menu is described now. The context menu is composed of 12 elements:

- edit participant, is a form which allow to modify all athlete information (name, surname, birthday, etc.), attributes (all extra attributes if there exists) and principle race configuration data (athlete wave, category, race number, team, etc.), the layout is similar to the *New Participant* form available in race preparation section (one grid row can only be selected).
- change event, is a more complex operation which involve coping or adding new item to the database tables. The change event operation is as follow:
    a) change wave, if there is the same wave in a new event choose this wave from this new event if not, choose a new one wave from the new event or copy the old one wave to the new event.

b) change category and second category, if there is the same category in a new event choose this category from this new event if not choose a new one category from the new event or copy the old one category to the new event.
c) change team type, if there is the same team type in a new event choose this team type from this new event if not choose a new one team type from the new event or copy the old one team type to the new event.
d) change team, if there is the same team in a new event choose this team from this new event if not choose a new one team from the new event or copy the old one team to the new event.
e) copy attributes, user can decide if all attributes from the previous event are copied to a new event if these attributes do not exists in the new event.

- change wave, change a wave to a new one from the same event.
- change category and second category, change a category to a new one from the same event.
- change team, change a team to a new one from the same team type from the same event.
- change team type, change a team type to a new one from the same event.
- change attribute, permit editing all attribute for the given athletes.
- change race number and tag, change a race number to a one not assign yet or change chip tag assign to a given race number (one grid row can only be selected).
- swap race numbers, change a race number with chip tags between two athletes (two grid's row have to be selected).
- bulk update, select from the grid controller all inserted by user athletes' race numbers.
- change finish status, the finish status for given athletes is changed to the selected one.
- print gird/report, the grid can be adjusted before the print, user can use column selector, filter or column grouping to create a view which want to print. Other facility is a use of templates which user can save and load to have a grid adjusted to a printing. First report preview is prepared with automatically added name race, date, logo and additional information if needed. What is more the report preview is prepared to fit the page to do this CubicSoft is using many tools like: grid scaling, choosing appropriate page orientation, etc. One important thing to notice is when user is using column grouping to print results then the presented logic is applied. Users are grouping

columns to create desire report view. When many columns are selected the CubicSoft is looking for expanded groups (using parent and child methodology) and for the result printing choose a first expanded group founded.

- grid/result exporting, user can export grid to the one of the given formats: .xls, .csv, .txt with tabulator, .pdf, .html. All generated documents are prepared similar like printing one with appropriate page fitting, etc.

For fast work and no need to refresh a grid manually almost all modifying operations generate queries (in multiple changes like edit participant or change event) or use Entity Framework (for single row modifications) to update database and grid data in the same time.

RACE RESULT CALCULATION MODULE

Now will be discussed result calculation module which is composed of advanced calculation's options which have to been adjusted by user before calculation process start. The result calculation module is run on separate thread and can work in two configurations: in standard mode with MySQL InnoDB database or in memory mode with MySQL Cluster. In the second case at the beginning all data needed is replicated from standard database to memory database. First idea was to replicate all race data to memory which was time consuming. Fortunately, due to great InnoDB database design to memory are copied only three, really two, tables: "eventathletes" and "eventteams". The third table "racerawtimes" which stores all chip tags reading inserted by CubicSoft module *Connector* which during insertion process verify if memory mode in CubicSoft is activated, if so, it inserts data two both InnoDB and Cluster database.

For the process of inserting data to the Cluster database very precise calculation of memory use have been done to avoid "run out of memory exception". The Table 6 presents the space occupied by each table, in total it gives **1,99 KB** per one record in each of 12 InnoDB tables. In memory Cluster database each variable occupies more space so in abbreviation Cluster occupies two times more space than InnoDB database, so it gives **3,98 KB** per one record. Taking into account the newest assumption that only three tables are stored in a memory mode the maximum size which can be consumed by Cluster during race is the following: "***eventteams***" (102 bytes * 10.000 teams = 1020 KB per race), "***eventathletes***" (80 bytes * 10 splits * 30.000 athletes = 24000 KB per race), "***racerawtimes***" (121 bytes * 10

splits * 30.000 athletes * 2 chip tags * 3 reading of each chip tag = 217800 KB per race) in total **242,82 MB (MySQL InnoDB) * 2 = 485,64 MB (MySQL Cluster)**. From this calculation a given assumption is done: Cluster node has to have minimum size of **485,64 MB** + additional space for configuration data **< 50 MB**. A node of size **524 MB** is sufficient to process the heaviest race result calculation. The presented calculation and tables size are exact on a day of writing this, takes into account that in the further development of CubicSoft the given sizes will be changing.

Table 6. MySQL Cluster database memory usage

Name of table	Size in KB
racerawtimes	121 bytes (multiple records)
eventraces	325 bytes
eventsplits	159 bytes
eventcategories	101 bytes
eventgunwaves	97 bytes
eventteamtypes	89 bytes
eventteams	102 bytes (multiple records)
eventattributes	339 bytes
eventathleteattributes	88 bytes (multiple records)
athletes	514 bytes/for memory 114* (multiple records)
eventracenumbers	79 bytes (multiple records)
eventathletes	80 bytes (multiple records)

Then the most crucial data used in result calculation is loaded from database and stored in memory in form of objects like *lists* for example all male and female athletes id are stored in a separate *lists*, all splits, waves, categories and race numbers are stored also in the *lists*. Now the result calculation can be performed in the fastest possible way with all data stored in memory.

1. the result calculation module every second is looking for a new chip tags readings in database for given race and split name. Every record consist of unique key, race ID, split name, chip tag (every athlete participating in race has a chip with coded unique tag, usually chip is glued to a race number), chip time (time recorded by TimingSense lector when athlete pass through control point - split), create date, antenna (indicates

which antenna from control point reads given chip tag), type (indicates type of the antenna read 0 – low power, 1 – high power). Every reading of chip is send by the lector to a CubicSoft due to the additional module *Connector*.

2. every record from the database is processed, and chip tag from the selected record is matched to the race number form a race configuration data.
3. split name form the selected record is matched to the split ID from a race configuration data.
4. chip time is inserted in an appropriate row (race ID, race number, split ID, athlete ID) for appropriate athlete in the database table "***eventathletes***". Additionally the table can store and calculate up to three type of time for every split: gun time, net time and leg time.
5. the last part in a calculation process is a setting position for every athlete. Due to great logical design of the database table "***eventathletes***" which store the race result to obtain a classification is only needed a pair of sorting technique:
    a. Overall position, sorting event column, split column and the meta split time column (gun, net or leg time).
    b. Category position, sorting event column, split column, category column and meta split time column (gun, net or leg time).
    c. Secondary category position, sorting event column, split column, secondary category column and meta split time column (gun, net or leg time).
    d. Gender position, sorting event column, split column, male athlete/female athlete column and meta split time column (gun, net or leg time).
    e. Attribute position, depend on a data type of the selected attribute (int, string, select, bool) and value of the given attribute (int value, string value, select value, bool – true, false). Selecting attribute column and choosing athlete column corresponding to the selected attribute column and then sorting event column, split column and the meta split time column (gun, net or leg time)

Moreover when is a team race the additional calculation is performed to obtain a team classification:

6. Teams position, the calculation of teams position depends on the user choice of the calculation method (the most common are: summing team total time or summing team total position), program can be extended by any additional calculation method if it is

needed. Selecting team column and athlete column corresponding to the same team column where a sufficient number of athletes of a team have finished race and have their finish time and positions. Then the team points and team time is saved and sorted by event column and one of the points/time column to obtain a team positions.

There are additional aspect considered during the result calculation (depends on an advanced options chosen by the user) like backup splits, laps or promotion category, and always athlete/team finish status is taken into account. There is an option of gun/net time rounding up or down. Another important aspect is a distance unit which is used to calculate average speed. In the event configuration data user choose individual and team time type to be used to calculate positions, additional there is a possibility to configure multi lap event. In split configuration data user choose maximum and minimum time for given control point, multiple readings (use first or last chip tag reading) and split time calculation interval, there is an option to configure backup split. The presented result calculation process in one second can process more than 300 athletes what during the race competition is more than sufficient. What is important to mention for timekeepers using CubicSoft every changes made in race configuration for example modification in race category or for example addition of new athletes demands the recalculations of results.

Additional element of result calculation module is a live result display which can be adjust by user and use to show classification on a big screen, but this part is under construction yet.

4.4 REPORTS GENERATION STAGE

At the end of the race a results are presented to a participants. The process of report generation must be fast and easy to manage by a timekeeper who not always is a computer scientist. Due to use of well defined logic the timekeeper can create every kind of classification with every single data included in the race configuration. CubicSoft has two forms of preparing and generating reports.

REPORTS GENERATION FROM GRID CONTROLLER

First method is a very fast and demands only a grid data display modification then user can generate and print report with a default layout created by application. CubicSoft takes header names of grouped columns or filtered columns and put all together in the header on the left hand side of the generated document. In the middle is placed a race name and actual date. On the right hand side is inserted logotype of the given timing company inserted by user in the application setting window. Moreover generated reports adjust grid display to A4 paper size using scaling and best fitting methods.

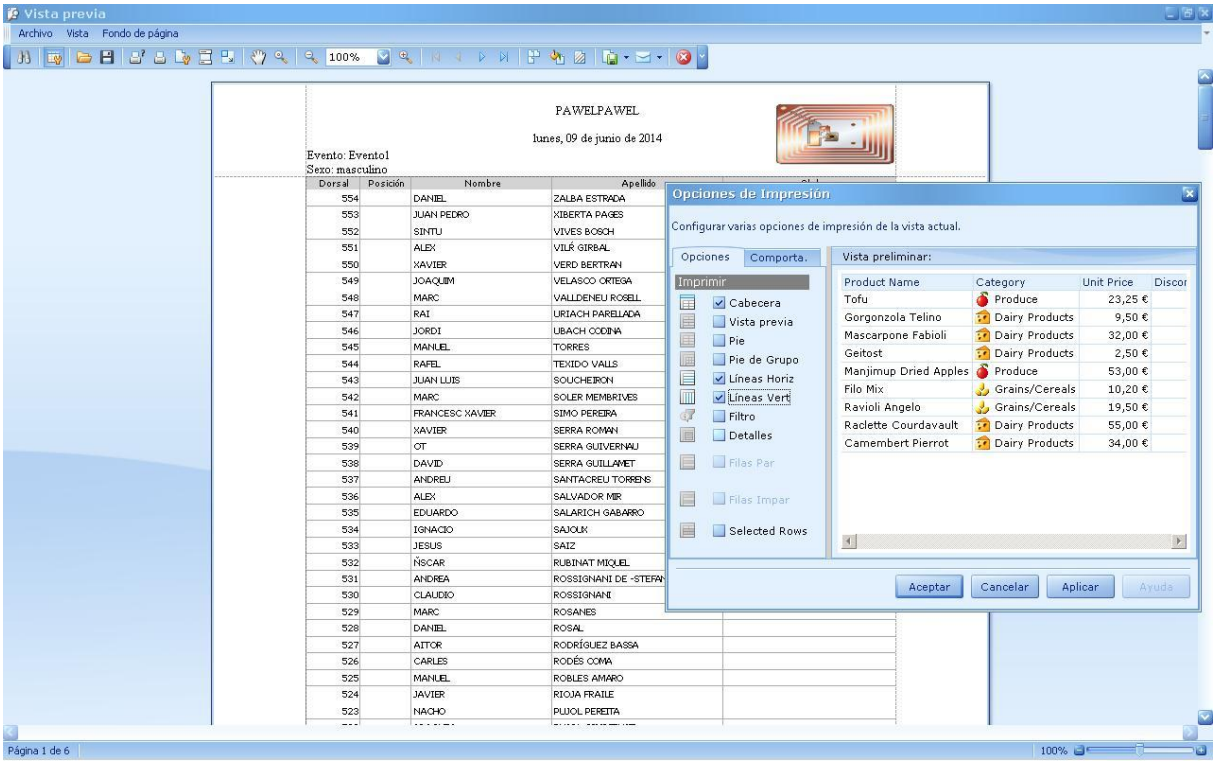

Figure 16. Enquiry report view.

REPORT GENERATION USING GUI

Second method is a generation of reports in a reporting section. Here user can use more complex but with intuitive and user friendly interface tool. Report preparation here is divided into two modes: expert and standard. In an expert mode an advanced user can used SQL query to get result data. In a standard mode user has an interface where can choose a data (columns) which want to include in a report. Then expert mode like a standard mode can create report layout due to DevExpress controller where a selected data will be placed.

DevExpress controller gives an user an interface similar to Microsoft Word where the user can prepare adequate to needs reports. What is more report configuration with layout can be saved or loaded to further use by means of online platform. Thanks to this solution whole timing company has own base of reports stored in one place. This section is in a constant development.

4.5 COMMUNICATION WITH TIMINGSENSE ONLINE PLATFORM

Online platform is a crucial element of TimingSense system. The platform is developed by third company placed in Valencia. During all implementation process there was a close collaboration between CubicSoft and online platform developers. Till now there exists 6 web services to data download (***wsGetLicense, wsValidateLicense, wsGetRacesList, wsGetRace, wsGetInscriptions***) and data upload (***wsSaveRace***). There is still a lack of two services to exchange reports and to receive chip tags reading send by lector directly to online platform instead of CubicSoft *Connector*. The connection to a web services is established by use of a standard .NET Framework System.Net namespace (*HttpWebRequest* class) over TCP/IP. In a communication process both response and request are encrypted by RIJNDAEL 256 algorithm to preserve confidentiality and data security. The messages consist of JSON interchange text format (data is serialized/deserialized by means of *Newtonsoft.Json* class).

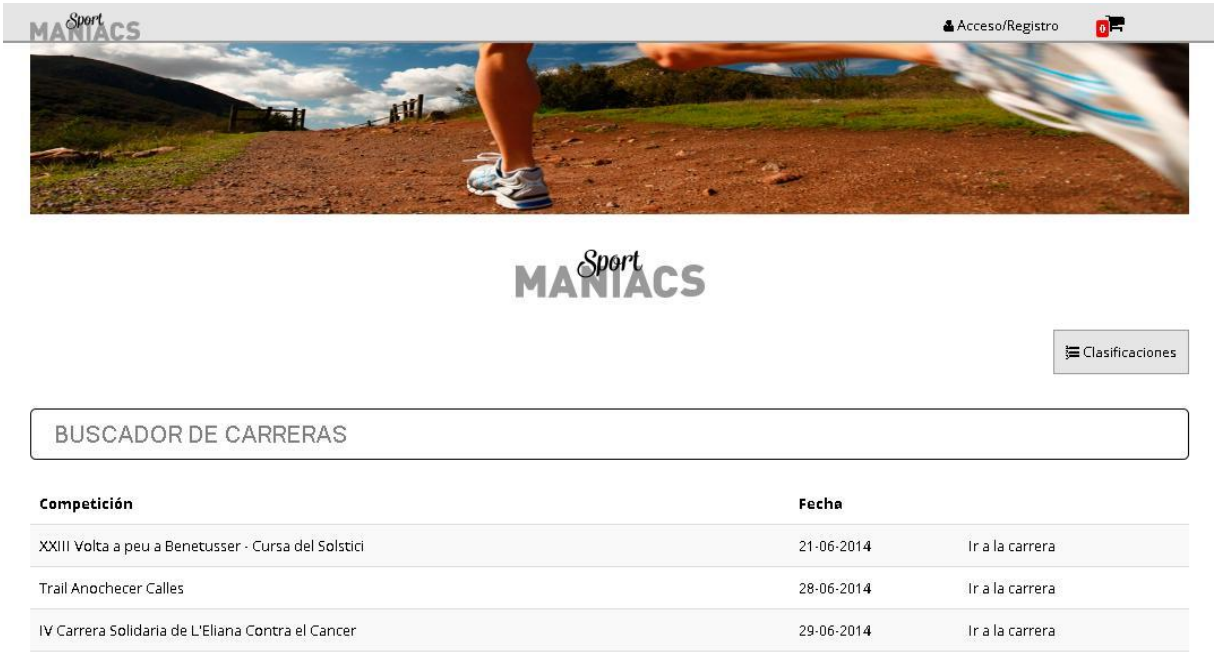

Figure 17. Online Platform web page.

LICENSING WEB SERVICES

The existing web services can be divided into two groups: web services dealing with licensing and web services responsible for a race configuration data interchange. First time when user executes CubicSoft there is a login window where user has to introduce its email and password generated in online platform. Through ***wsGetLicense*** web service CubicSoft interchange data with online platform, CubicSoft receives information about unique license number, validate date of the license, company id and unique user id (timer id).

Then there is generated an unique *id* which is used as a race id (every new race created in CubicSoft increments given race id by 1). The construction of discussed *id* is the following: race id has given format 000001111122222 where 00000 corresponds to a company id where given timer works, 11111 corresponds to a timer id, 22222 corresponds to the id part of the race id.

The following *id* format is preserved by both sides CubicSoft and Online Platform, and is crucial in functionality of TimingSense system. An user of TimingSense system can create new race in online platform as well as in CubicSoft software the race id has to be always unique therefore there is a race numeration logic applied. Online platform starts numbering race id from 00001 whereas CubicSoft starts from 50001. Secondly the user license received from online platform is processed by CubicSoft and with specially prepared xml file encrypted with RIJNDAEL 256 cipher is saved in database. The encrypted xml file consist of given labels:

- <Empresa></Empresa> - stores a company id
- <Timer></Timer> - stores a timer id
- <FechaIni></FechaIni> - stores the first date of initializing license key
- <FechaFin></FechaFin> - stores the finish license date
- <FechaActual></FechaActual> - stores the last date of synchronizing with platform
- <Windows></Windows> - stores the specially calculated system distribution key

Due to given construction of license file the software cannot be corrupted so easily. First the CubicSoft need a connection to Internet to perform well time measuring during race competition so license key with date fields can be checked and updated due to ***wsValidateLicense*** web service. Storing information about actual pc system version protect CubicSoft from coping version of program from one computer to the other one.

RACE DATA INTERCHANGE WEB SERVICES

The next portion of web services are used to interchange race configuration data. In most cases there is performed an online inscription to a race. Then before a race will start timekeeper has to download all inscribed athletes to CubicSoft program. For it first CubicSoft uses ***wsGetRacesList*** web service to get a list of existing races created in online platform, the races are grouped in two parts: future races and old races. The next step is to download race configuration data by means of ***wsGetRace*** web service and then download all athletes inscribed to the given race due to ***wsGetInscriptions*** web service, the presented order of data downloads is obligatory. In just two simple click user can transfer all race data from online platform inside CubicSoft software. The process of downloading inscribed athletes can last a bit as a large data packet is being transmitted by the network. Finally all received information is deserialized from JSON object and processed to generate SQL INSERT query and update database by use of native connection. Every race downloaded again from online platform first is removed and then inserted once more.

Figure 18. CubicSoft uses web service to download race configuration data from platform.

The last web service ***wsSaveRace*** is useful when a race is finished and user wants to uploads result to online platform. CubicSoft first gather all race data from database, then serialized it and sends to online platform. The generated data has a large size so the upload process will last a bit more. In the future when online platform will be more powerful the calculation process will be performed also online.

4.7. FUTURE DEVELOPMENT

CubicSoft with its modules is creating a part of TimingSense system which is supposed to be a commercial product addressed to a customer from all over the world. CubicSoft is in a constant development and surely as a commercial product will be adjusting to the customers' needs.

At the present moment two elements are in a developing phase: result display and report layout builder. Result display will serve to presents live result in user friendly way with nice designed interactive interface. User will decided which information want to display and in which order, all of this can be shown on an extern display like other computer screen or big TV screen during race competition. Result display module will work only with executed result calculation module. The second element, report layout builder, is used by user to prepare own layout by means of DevExpress controller with a similar interface to Microsoft Office Word. Report layout builder is helping an user to easily create demanded layout with just only a few clicks. It always gives a great support to less advanced users.

There are some ideas to future development, additional functionality for CubicSoft. One of the ideas is adding a special method which will verify, during importing athletes process, if corresponding athlete name corresponds to athlete gender. The action method is the following: checking if an athlete with the same name was imported earlier to the database, if so checking if the gender corresponds to the gender of newly imported athlete, if not user will be inform about it and will choose which gender is a correct gender for the given athlete. This problem is a very common as during competition organizer gives wrong sex or forget about delivering this data. Then the given athlete does not have a gender position or is assigned to wrong category, then to correct it all result recalculation is required.

Likewise there are two pending web services to be developed. Exactly communication between CubicSoft and online platform to interchange reports data. Timekeepers from a given timing company can shared all their reports with other colleagues. Sharing reports it means storing layout design and SQL query which construct proper classification. User can in every time upload or download reports from templates stored in database. Another aspect is adding new web service to CubicSoft module, *Connector*, to communicate with online platform and download chip tags reading. As TimingSense lector can send readings directly to online platform, the *Connector* will need to connect with platform to receive data. This type of communication will be used commonly if the online platform will be enough powerful to run result calculation module on the server side.

CubicSoft is the program that will be developed surely in the future as the TimingSense system will became more common and start gaining the market. The needs of customers will be increasing so more work will be required.

5 TESTING

This is the last stage of the software development before it can be widely applied to normal use. In this phase, a general tests to verify the main functionality of CubicSoft software on various user's behaviours were performed. Likewise a tests to check the performance of the race result calculation capability during a real race were conducted. The testing part is a time consuming part where very precise work is needed. There is a lot of work for programmers and testers.

5.1. SOFTWARE TESTING

The software testing was performed to verify and validate the CubicSoft. The most important part was to validate if the software reacts correctly on the user's behaviours, if there are any unexpected exceptions. CubicSoft software will be used in many race's competitions so any unhandled exceptions are unacceptable, especially operations including data manipulation can have catastrophic results, for example modifying an athlete race configuration data or coping information from one event to other.

First tests were performed during implementation phase. CubicSoft was frequently checked if a new added structure works as expected and does not influence incorrectly on the others parts by means of traditional method of breakpoints and watches, both tools are available in Microsoft Visual Studio. This was a white-box testing carried out by software developer. Techniques used in white-box testing including: API testing, code coverage testing and static testing (code reviews and inspections). Tests were performed at all levels: unit testing, integrated testing, system testing and now application is in the phase of acceptance testing (where timekeepers are testing CubicSoft as a fully functional product). CubicSoft was exactly checked if manage to react correctly on every type of behaviours and now application is testing in a real race competition environment. CubicSoft now is in Alpha testing where independent TimingSense test team is dealing with it. What is more to handle an unaccepted behaviours, a list of exceptions for correct interaction of the software with the user was implemented, a given list is presented in the next chapter in a user documentation.

The last part of testing was an installation and compatibility testing. CubicSoft has been installed and executed successfully on three most commonly used operating systems: Windows 8, Windows 7 Home Premium SP1 and Windows Vista Home Premium SP2 each of them with minimum .NET Framework 4.0 and MySQL database with MySQL Connector/NET installed.

5.2. EXAMINATION OF TIMING CAPABILITIES

There were performed many result calculation to examine the efficiency and timing capability of CubicSoft with many parameters' configurations. The general rule is the bigger race is (with many athletes and attributes) more complex and time consuming the result calculation process is.

CUBICSOFT DOMESTIC TESTS

The tests of result calculation capability was done with use of chip tags reading simulator, created for test purposes, which was inserting chip tags in the database table "***racerawtimes***". CubicSoft was receiving every entrance of new reading and perform all process of result calculation with insertion of correct classification data to database table "***eventathletes***". In a domestic condition the result calculation module was able to process between 300 (MySQL InnoDB) and 400 (MySQL Cluster) athletes which gives more than satisfactory results. During a race the more loaded point of control is a "start" where during short period of time a huge amount of athletes is crossing the line for example start of 10000 athletes in a one wave can last approximately 4 minutes what gives 42 athletes per second and every athlete has many chip tags readings. However capability of processing more than 300 athletes per second is enough. The heaviest aspect of processing calculation is a reading of new chip tags time from database and then inserting the calculated result back to database. Here is visible a difference between database storage engine where better performance has MySQL Cluster over MySQL InnoDB. The less important here is an aspect of performing sorting to obtain athletes' positions. The Cluster memory mode result calculation can process more athletes per second then InnoDB. Although bigger difference between given database storage engines can be seen on a race result calculation at once for example when is need to recalculate all results. In this case CubicSoft has to gets all chip tags reading for given race, it can be more than 100.000 records, then conducts positions calculation and finally inserts result back to database. Here the MySQL Cluster seems to have two times better efficiency than MySQL InnoDB thanks to storing all data in memory.

CUBICSOFT REAL RACE TESTS

CubicSoft is now in a phase of real race environment testing. In a domestic condition is impossible to deal with all kind of situation which can occur during a race competition. The result of this testing are still developed, but generally CubicSoft seems to work as expected with high efficiency.

6. SOFTWARE DEPLOYMENT

This chapter is presenting the final release of the CubicSoft with all details about the installation process, the usage requirements and the user documentation. The release version was prepared for installation purposes where all source code was inspected to eliminates all unnecessary classes, methods, variables and comments. Additionally there was used a specially prepared compilation method "*release*" available in Microsoft Visual Studio which makes use of compiler optimizations to obtain final version of the software.

6.1. SOFTWARE INSTALLATION AND MAINTENANCE

This section presents the last aspect of the project which is an installation process phase and a general requirements description. Before the installation process will be described the CubicSoft main requirements will be presented. After a creation of the final release version 1.0 of CubicSoft it was possible to set the final requirements of user workstation. It is obvious that CubicSoft was written in C# programming language and is using .NET Framework 4.0 which is indispensable to be installed on user workstation. Apart from it the necessary is MySQL database with MySQL Connector/NET to be installed on an user pc. The CubicSoft is compatible with 32 bits architecture and works on 64 bits architecture as well. The operating system compatible with CubicSoft is Microsoft Windows XP SP3 up to the newest Windows 8. The user pc must have 2 GB of memory RAM as a minimum to correct runs and executes result calculation module.

INNO SETUP INSTALLATION PROGRAM

The final released version of CubicSoft 1.0 was prepared to the distribution by use of the free installer for Windows platform Inno Setup.[45] Inno Setup is a powerful script based on an installation builder. To build any installer knowledge of a main Inno Setup's script commands is necessary. CubicSoft installer version was prepared with the aid of the Inno Setup. Figure 17 presents a snippet of the installation script which generates the CubicSoft installer.

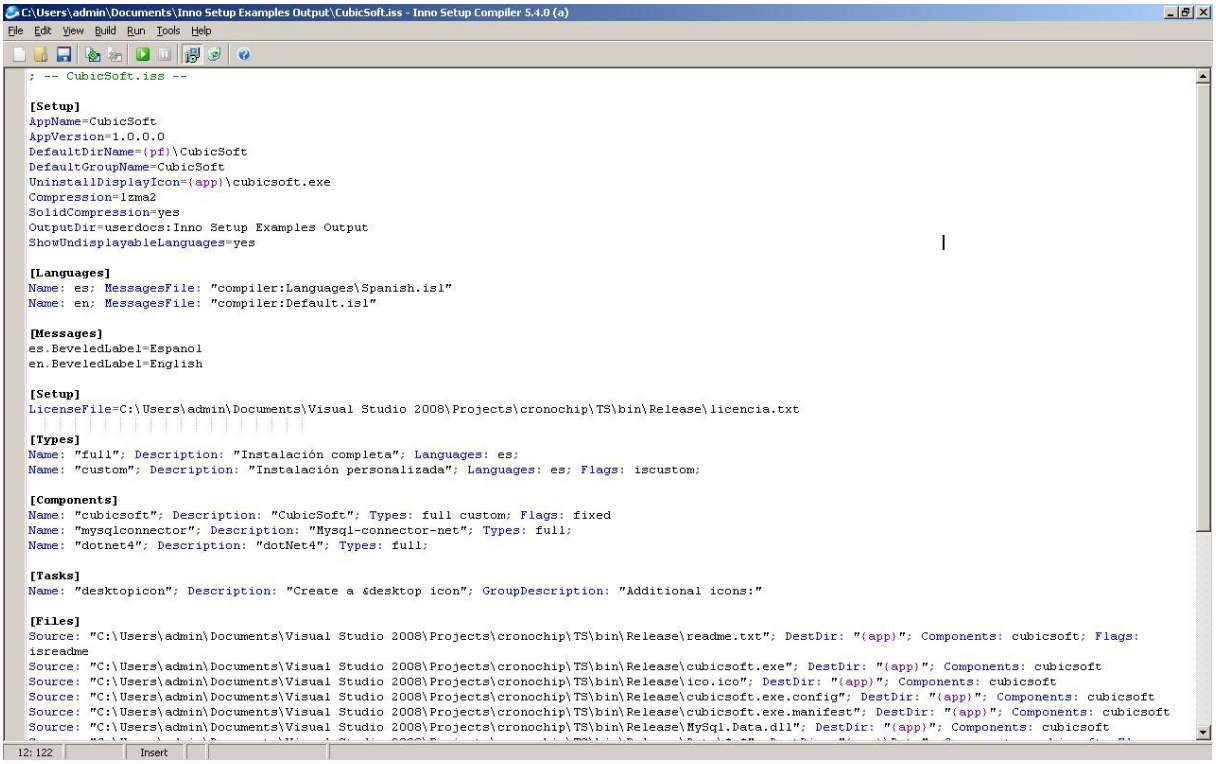

Figure 19. A part of the CubicSoft script installer builder.

The script consist of tags. Every tag has specified values and options that can be used. In the first part are defined major properties of the CubicSoft like name, current version. Then are prepared installer options. The tag [Languages] allows to choose multi-language interface. As CubicSoft is also a multi-language application there is a need of this type of the installer. The tag [Tasks] provides additional options like desktop shortcut. The main function in the script has the tag [Files]. Here are defined all necessary files for CubicSoft which will be copied and without them the application will not work properly. The installer is including all CubicSoft configuration files (.exe, .xml, etc.), all dll's (DevExpres, MySQL, etc.) and other files (readme, license, etc.). There was an idea to bundle with the installer, .NET Framework 4.0 installation file, but after short examination this idea was denied as all new Windows system has it installed as default. The same idea to bundle the installer of MySQL server was raised, but unfortunately because of license term and complexity of the above it cannot be done. The CubicSoft installer when finished its work indicates a user that he has to install the MySQL server with Connector/NET and provides a MySQL download site URL. The whole process of CubicSoft installation take a few minutes. There is one recommendation to install CubicSoft outside the *Program Files* directory because when user do not have the administration privileges Windows operating system makes CubicSoft unable to modify the

configuration layout file stored in this directory. Moreover the installer package provides the uninstall program, as well. This feature makes easy to uninstall earlier installed program with all its files from the user computer.

CUBICSOFT UPDATE PROCESS

The last aspect not mentioned here is a CubicSoft auto-update. Till now the updating process is not developed. The idea is to use a web service running on online platform side with a very simple methodology:

- check the new version by comparing numbers from last version update on server side with current Assembly version on user pc.
- if the version is higher on server side, the latest installer is downloaded
- run the downloaded installer and close the application (installer will take care of the rest)

The Visual Studio has a built-in installer builder and auto-update tool which name is ClickOnce. However, CubicSoft use very personalize installer with many advanced functions like multi language, bundled installers, etc. which ClickOnce not support. Therefore alternative way was used.

6.2. USER DOCUMENTATION

The user documentation provides full information about how to use the application in form of brief tutorial. The tutorial guides an user through each step of configuration and usage of the CubicSoft. Moreover user documentation provides thorough troubleshooting assistance. There is also included a list of typical exceptions handled by CubicSoft software.

6.2.1. TUTORIAL

The tutorial is divided into two parts: installation and first configuration part, and usage part. It supports an user from the beginning, setup, through the usage of the program. Each step is described in the manner that even not advanced user can go through the tutorial and understand it clearly. The tutorial is presenting a real race case - a creation and a management of a race data.

INSTALLATION AND CONFIGURATION

After downloading the installer from online platform and executing it the first window of the installer will occur. The user must select one of the given language: Español or English. Then will appear the installer welcome window in the selected language.

Figure 20. CubicSoft installer window.

The installation process consists of a few steps which user has to follows. Then user will go form CubicSoft installer directly to MySQL server download page where after downloading the MySQL installer and executing it the user has to follow the setup (remembering to install Connector/Net and setting default user and password for root user of the database). After installation is finished user can execute installed application, CubicSoft.

During the first run of CubicSoft there will occurred login window where user has to introduce e-mail and password received during registration on online platform. Then the application will open, it can last a few seconds because during the first execution MySQL Cluster and InnoDB database will be created and will be configured. Then CubicSoft configuration window will occurs where user can adjusts the application interface to its needs for example timekeeper can select the interface language, default is Spanish, can indicate the

actual country and time zone or can select appropriate theme interface, etc. Then finally the user can start to use the CubicSoft and create a first real race.

Figure 21. CubicSoft settings window.

CUBICSOFT USAGE

This part presents the principle usage of the CubicSoft, creation of the real race, without description of the user interface which is detailed featured in design chapter. User can get know with principle information, how to create and then manage a race data.

Figure 22. CubicSoft principal ribbon menu.

Apart from a main ribbon menu, CubicSoft posses additional pop-up menu with access to all CubicSoft extra modules (*Connector*, *Lector*, *Fair*) and quick access bar menu. Quick bar menu consist of shortcuts to the most commonly used items, it is: enquiries, result calculation module, reports, general application settings and *about* program information.

RACE DATA PREPARATION

The main window of CubicSoft consists of a ribbon menu bar where are placed all available options of the application. First section is a race preparation where user creates and then configure a race. By clicking button *New* user starts to create a new race in a wizard form which consist of 10 steps which a timekeeper has to follow. First user has to introduce basic information about created race (fields with asterisk '*' are mandatory) and decides if created race will copy all basic configuration data from earlier created race or puts the number of event and splits for new race. The user configure every event by introducing data. Considering that the user is dealing with a marathon race which consist of two events: 10 km and 42 km race. Then a timekeeper go to the next steps where configures for every event:

- number of splits (the obligatory is one split of type "meta"), in a marathon race case will be 3 splits: 1 – start, 2 – control point on a distance of 21 km, 3 – meta.
- all categories (are two types of categories: primary category and secondary category which can be *age based* categories or *standard* categories sorted by gender type, every athlete can pretend to only one category), in a marathon race case will be created a categories consisting of an age ranges. User can copy once created categories from one event to another as the two events will use the same categories classification.
- number of waves (the mandatory is one wave which provide a race start time), in a marathon race case the user will define one wave for every event.
- all team types (if the competition is a team race, user has to define here all types of teams participating in a race), in a marathon race case the user will omit this step as a marathon is a individual race.
- all attributes (every additional information about athlete which is not included in a standard athlete data), in a marathon race case the user will omit this step as any additional information about an athletes will not be imported.

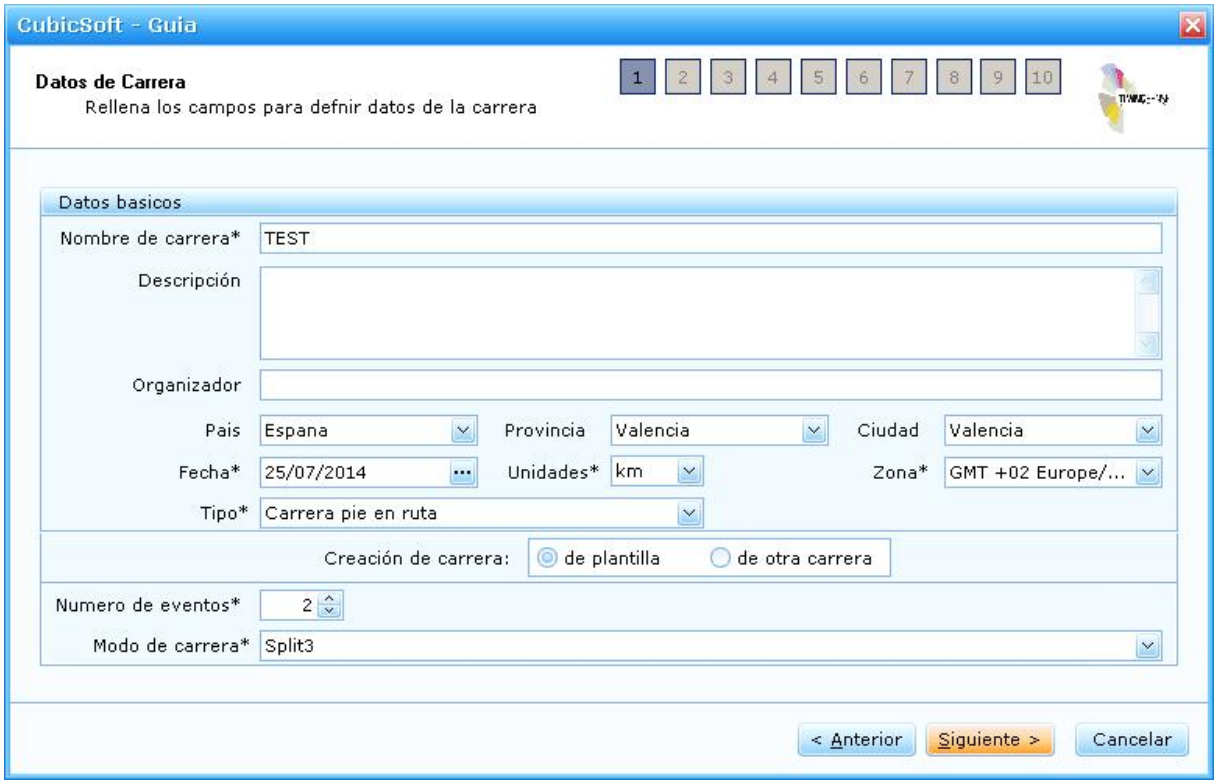

Figure 23. CubicSoft wizard window, a new race preparation.

The next step is an athlete and chip importation where user selects a file to import an athlete information and a race number with a chip tags. Chip importation also allow a coping chips from earlier created races. The file with athletes data is usually prepared by a race organizer and a few days after the race a timekeeper receives the given information. The race number file is prepared by a timekeeper by use of additional CubicSoft module the *Lector programmer* which is programming chips and generating a given file. The last step is just adjusting an advanced race calculation options like a race time type for result calculation (gun time, net time, leg time in a marathon race case). What is more the every presented step posses its own functionality and options like coping data from other event, loading/saving templates, etc.

RACE DATA CONFIGURATION

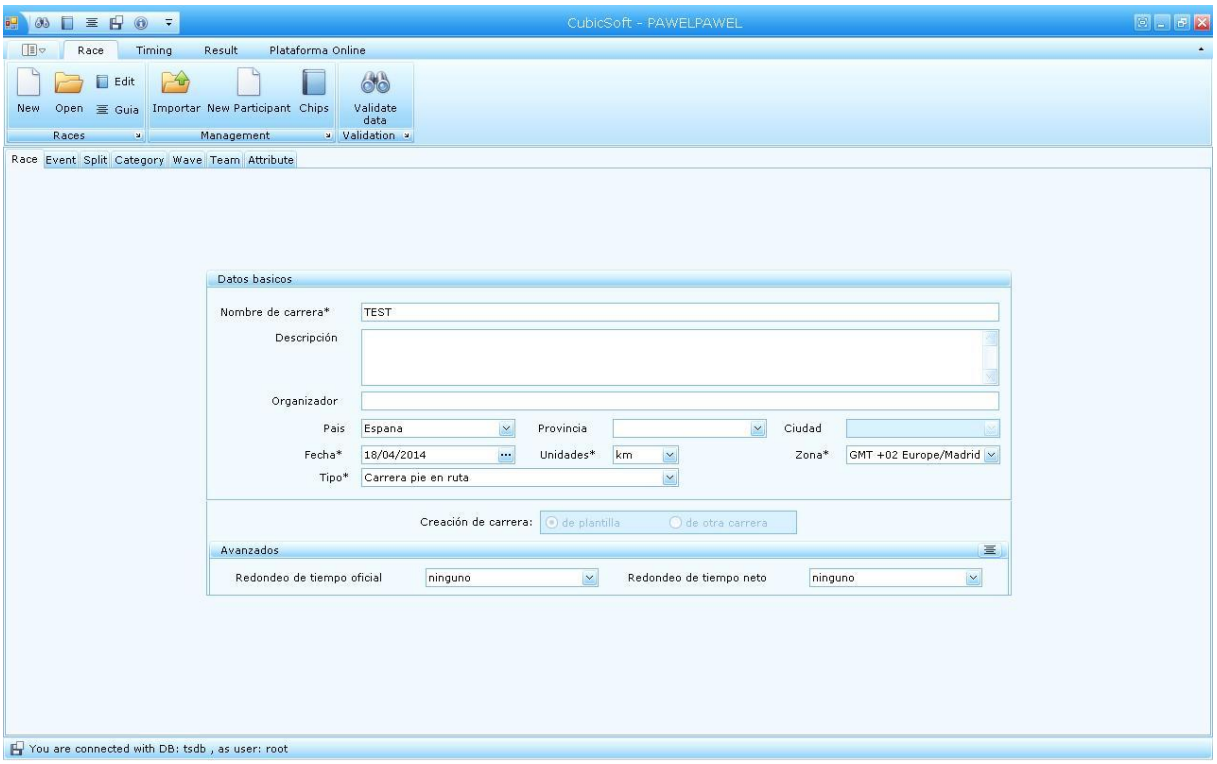

Figure 24. CubicSoft main window, a race data configuration.

After the wizard window is closed and a race data is basically prepared the user can manipulate all race data from CubicSoft main window view (an user can come back to the wizard window only if the given race was created in the same session). All steps presented in the wizard are grouped now in tabs: edit (race, events, splits, categories, waves, teams type, attributes), import athlete/chip. User in every moment can edit or add a new race information like change a second split distance on demand of the marathon race organizer and put it at 22 km not as earlier on 21 km. There are additional element like add *new athlete* where user can append a new athletes manually, in case of marathon race the organizer send as mail in the race day with two new athletes so a timekeeper has to introduced this data to the program.

The last step recommended for a timekeeper is a race data validation. The timekeeper will perform a race validation to check a race configuration correctness in the marathon race day by use of validation window. If the validation window shows some errors, user with a single click can go to an exact place in the program where the given error is occurring like the Figure 25 below is presenting there are some athletes without race number assignation.

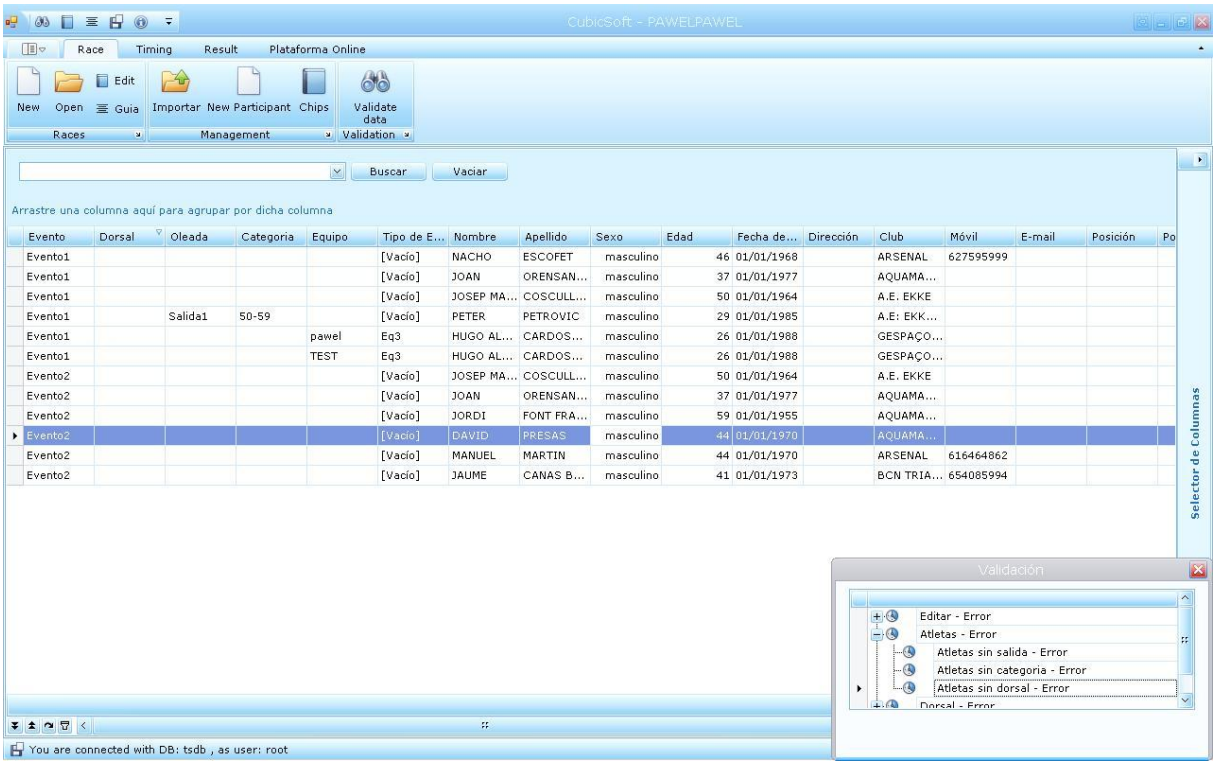

Figure 25. CubicSoft validation window, enquiries view showing all athletes without race number assignation.

MANAGEMENT OF RACE TIME MEASURING

The next section a result calculation is a timing part which the user is handled during a competition, marathon race. It consists of enquiries where user can modify all race athlete data created earlier like: edit athlete information, change athlete race status, change event (coping old configuration and adjusting it to a new event), change wave, change category, change secondary category, change team, change team type, change attribute (edit every attribute value), change race number and corresponding chip tags, swap race number only with selected by user chip tags. In this part the timekeeper spends most time of the race dealing with all type of incidence which the marathon organizer before and during a race delivers to the timekeeper for example the provided information contains the change of event for 3 athletes who decided to participate in a 10 km race instead of a marathon race and the change of the category for 1 athlete.

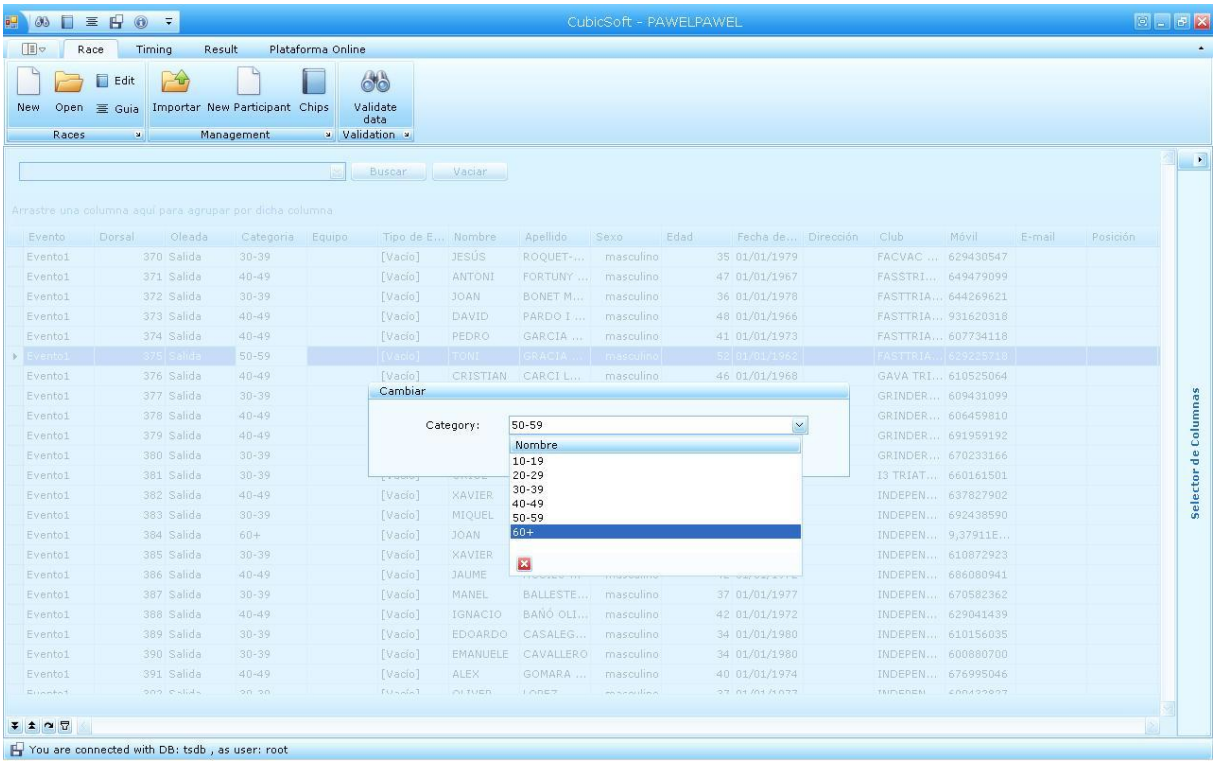

Figure 26. CubicSoft enquiries view, change the race category for a selected athlete.

The next step before the race will start is necessary to execute the *Connector* module and configure it with TimingSense lectors. In case of the marathon race there are 3 splits with three distinct TimingSense equipment which will send a chip tags readings. After the all connections are established the user can go to the next step.

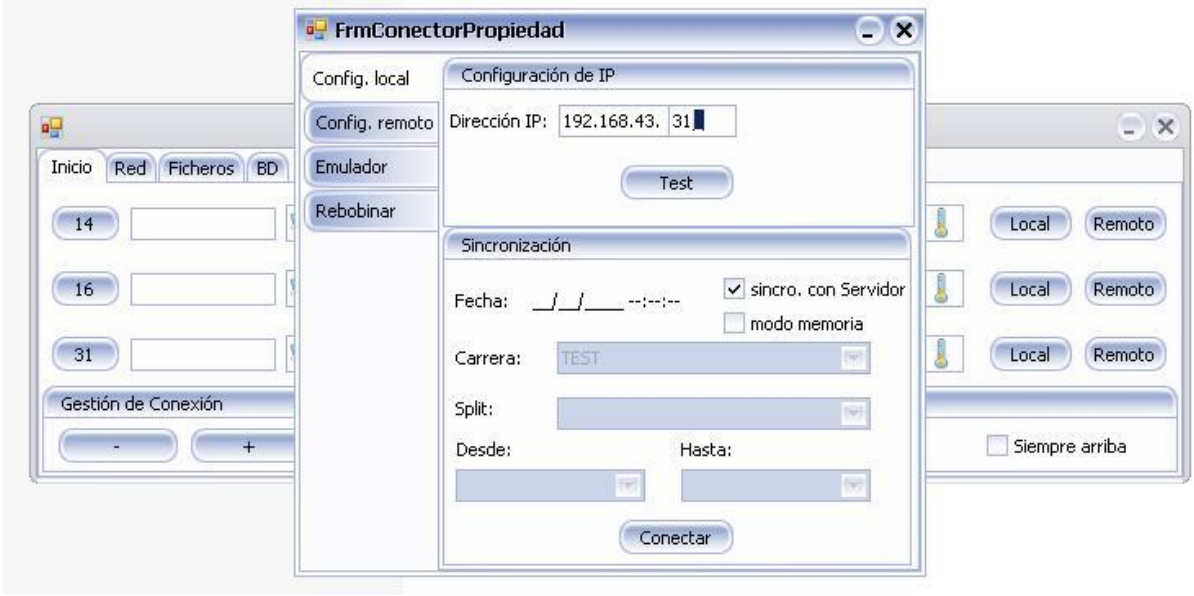

Figure 27. CubicSoft *Connector* module, connections with TimingSense lectors.

The next element is a *results calculation* to perform result calculation process by just one click with additional calculation options which can be adjusted by user. The timekeeper should execute a result module before a race starts to check if an every chip tags readings is process correctly (the result calculation process is performed every second). The user choose if wants to use a memory mode to calculate results which speeds up the calculation process (the memory mode result calculation is not recommended only if the user is recalculating all results after the race competition).

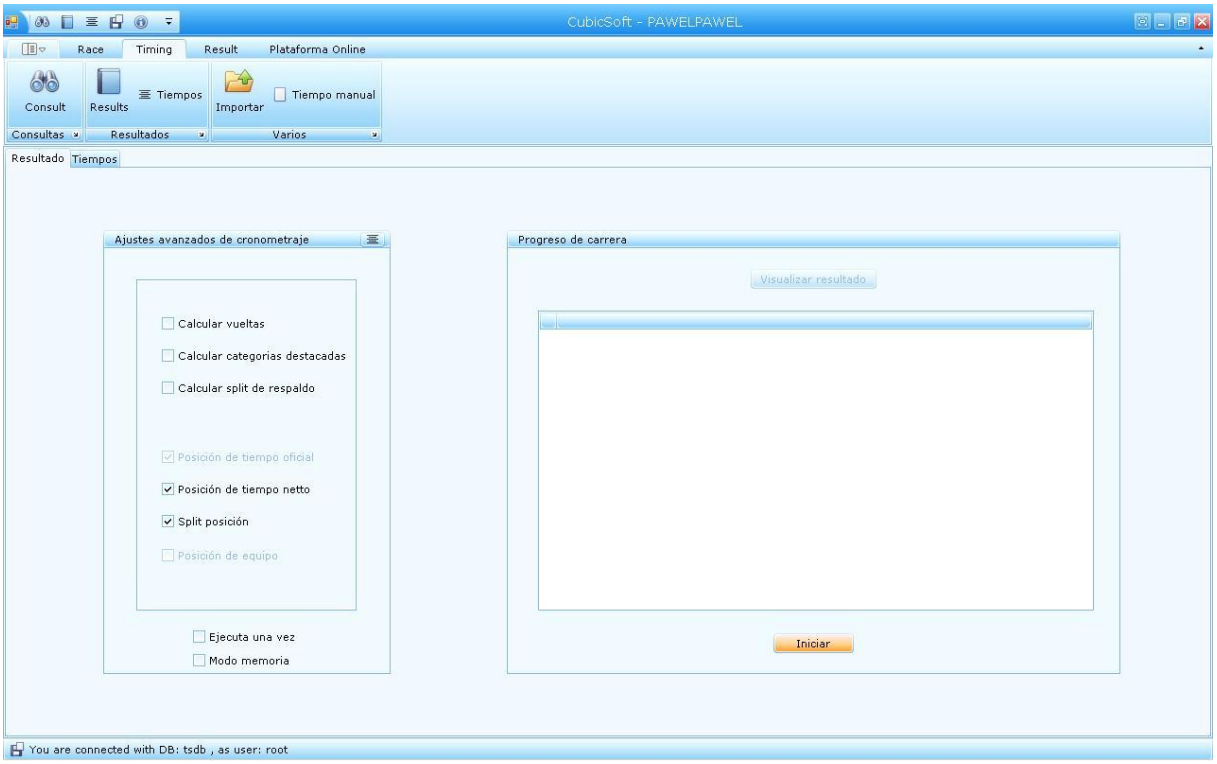

Figure 28. CubicSoft result calculation module.

Then the timekeeper move to the *wave times* part where user places a start time for every wave in every event. The timekeeper can update time manually, with PC clock or with online platform. This part is crucial as the race start time is being saved. In the presented marathon race are two events: 10 km and marathon with one wave each so the timekeeper presses update button during the race start.

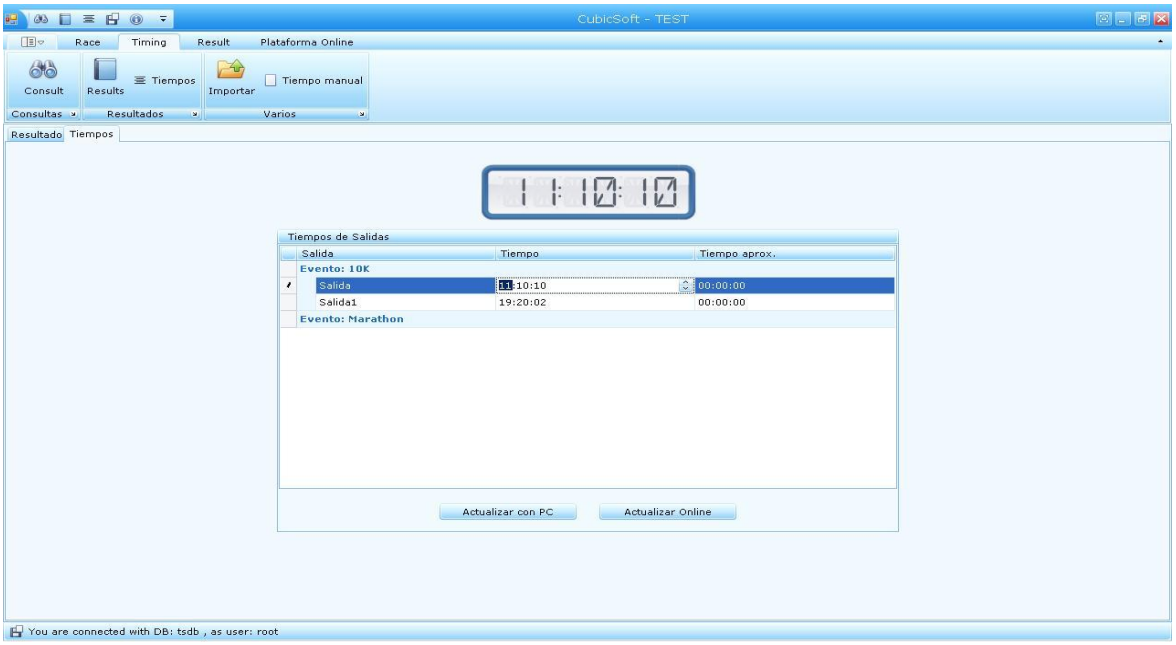

Figure 29. CubicSoft event wave time view.

REPORTS GENERATION

During the race and after the race, the timekeeper has to prepare reports for a speakers and for the race organizers. The user can make it with very easy way just adjusting the enquiries grid layout and then generates/prints final report by choosing appropriate option from the contextual menu.

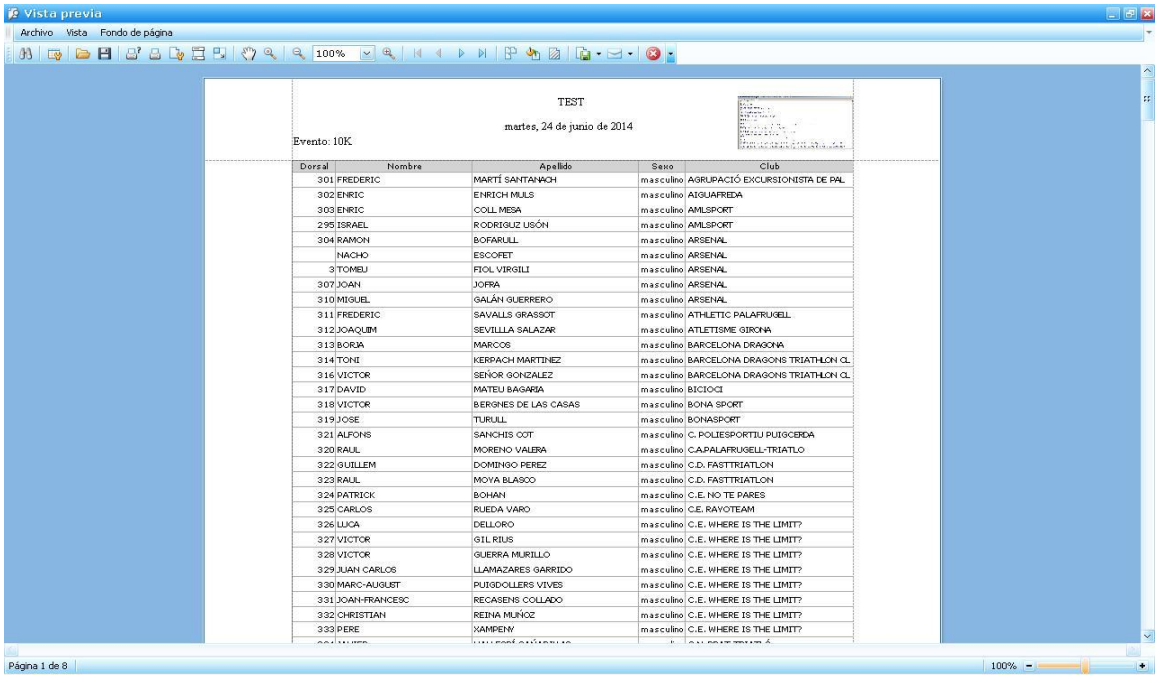

Figure 30. CubicSoft grid report view.

ONLINE PLATFORM RESULT PUBLICATION

The last section which the user can use is the online platform where the timekeeper can upload the race result data. The timekeeper with a just single click can export all result with a race configuration data to the online platform. Then the race participants can check the race result in the internet.

6.2.2. LIST OF COMMON EXCEPTIONS

The list of common exceptions presents the most frequent exception which may occur during usage and execution of the CubicSoft program. The list of exceptions is an important part of the user guide as it provides thorough troubleshooting assistance. Given list of the exceptions of CubicSoft software is described in the Table 7 below.

Table 7. List of the basic exceptions handled by CubicSoft

Type of message	Description
DB connection failure	A connection to MySQL database failed
License error	Verify correctness of provided data
Invalid license	The license is not valid, contact with TimingSense
Download error	Error occurred during downloading data
Upload error	Error occurred during uploading data
File incompatible	Check the content of the file
Restricted area	The given area is inaccessible
Operation not allowed	The operation cannot be performed in given grid customization
Distinct events	This change is not permitted for multiple events
Empty column	Race number column cannot be empty
Incompatible column	Birthday column has wrong format
Lack of race	Corresponding race has to be downloaded first
Validate race	The race validation is needed

7. CONCLUSIONS

The goal of this project was to design and implement a specialized software for the time control purposes of the high participation non-motorized sports events (athletic races, cycling, triathlon, etc.) where a timing process is done by the RFID technology. The project till now is developing successful, and the developed application, CubicSoft, already has been testing in real race environment. Implemented application was coded in C# programming language and .NET Framework 4.0 technology. CubicSoft fully manage all aspects and features of the race, starting from race preparation, result calculation and ending on reports generation. Additional feature of the program, essential for TimingSense system, is communication with online platform, to import participants and race configuration data and to publish the results so that the whole process becomes easier for the timekeeper, and all due to web services. Moreover the software can communicate with a timing equipment to receive a chip tags readings, monitor the status of equipment or recover old chips readings. Described software, CubicSoft, is a motherboard for the newly formed sport timing system, TimingSense, and it manages all necessary race timing functions and communication between timing equipment and online platform. To sum up the practical part, as a result of the cooperation with Cronochip company a technologically advanced application with alternatively simple and efficient interface was implemented and now it is preparing to be used by timing companies all over the world.

The theoretical part shows all stages of the software development: modelling, design, implementation, testing and deployment phase. Moreover, this dissertation explains all technical aspects and a theoretical issues necessary to understand the construction of the TimingSense system and the basis of the race timing.

8. REFERENCES

8.1 PRINTED SOURCES

[1] Cornel Turcu, “Current Trends and Challenges in RFID”, InTech, Croatia, 2011

[2] Daniel Dobkin, “The RF in RFID: Passive UHF RFID in Practice”, Newnes, Oxford, 2008

[3] Jeff Ferguson, Brian Patterson, Jason Beres, Pierre Boutquin, and Meeta Gupta, “C# Bible”, Wiley Publishing, Indiana, 2002

[4] Wei-Meng Lee, “C# 2008 Programmer's Reference”, Wiley Publishing, Indiana, 2008

[5] Weis Stephen, “RFID (Radio Frequency Identification): Principles and Applications”, MIT CSAIL, 2007

[6] ECMA-334 Standard, “C# Language Specification”, Ecma International, 2006

[7] ECMA-404 Standard, “The JSON Data Interchange Format”, Ecma International, 2013

[8] ECMA-335 Standard, “Common Language Infrastructure (CLI)”, Ecma International, 2010

8.2 INTERNET SOURCES

[20] http://msdn.microsoft.com/en-us/library/8z6watww.aspx, available at April 2014

[21] http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx, available at April 2014

[22] http://www.oracle.com/us/products/mysql/overview/index.html, available at April 2014

[23] http://msdn.microsoft.com/en-us/library/zw4w595w.aspx, available at April 2014

[24] http://msdn.microsoft.com/en-us/library/kx37x362.aspx, available at April 2014

[25] http://www.mysql.com/about, available at April 2014

[26] http://www.oracle.com/us/sun/index.htm, available at April 2014

[27] http://dev.mysql.com/doc/refman/5.7/en/storage-engines.html, available at April 2014

[28] http://www.mysql.com/products/cluster/, available at April 2014

[29] http://www.json.org/json-pl.html, available at April 2014

[30] http://www.microsoft.com/visualstudio/products/default.mspx, available at April 2014

[31] http://www.mysql.com/products/workbench, available at April 2014

[32] https://www.devexpress.com/Products/NET/Controls/WinForms, available at April 2014

[33] http://www.telerik.com/products/winforms.aspx, available at April 2014

[34] http://www.mysql.com/products/connector, available at April 2014

[35] http://www.nuget.org/packages/newtonsoft.json, available at April 2014

[36] http://www.codeproject.com, available at April 2014

[37] http://stackoverflow.com/, available at April 2014

[38] http://www.racetectiming.com, available at April 2014

[39] http://balsamiq.com/, available at May 2014

[40] https://www.eclipse.org/ganymede, available at May 2014

[41] http://www.pool.ntp.org/, available at May 2014

[42] https://www.assembla.com/, available at May 2014

[43] http://www.redbooth.com/, available at May 2014

[44] http://msdn.microsoft.com/data/ef.aspx, available at May 2014

[45] http://www.jrsoftware.org/, available at June 2014

APPENDIX 1 – COMMUNICATION BETWEEN CONNECTOR AND LECTOR

DATA SOCKET

Description of the interchange protocol of the information about chip readings by data socket. Socket TCP/IP by default work on port 10. For the negotiated connection, the lector takes the server role, remaining in listening state to connection requests.

The basic unit of information transferred by the socket data from lector to the *Connector*, is a follow JSON package with the following labels:

t	reading type (integer)
h	hour (date time)
c	milliseconds (integer)
t	reading type (integer)
h	hour (date time)
c	milliseconds (integer)
a	antenna (integer)
x	chip (string)
s	split (string)
r	race (string)
u	timekeeper (string)

Example

Suppose the following reading: *aa00470011223344090006123014052027xx.*

A message delivered by the lector to the *Connector*, is following:

{"t":1,"h":"20061230T14:05:20","c":39,"a":1,"x":"044","s":”split1”,"r":”race1”,"u":”user1”}

The message corresponds to an entity of JSON serialized through the *Newtonsoft.Json* package. The aim is a brevity of the package and reduced number of padding characters. Always after a message, end of line character appears, ASCII 10 character (\n).

REWIND SOCKET

Description of the interchange protocol of the information about chip readings by rewind socket. Socket TCP/IP by default work on port 9998. For the negotiated connection, the lector takes the server role, remaining in listening state to connection requests.

Rewind request

The prompt rewind message (pc → lector), has the following format:

{"tipo":”REBOBINADO”,"sql":"xxx"}

Knowing ‘xxx’ is a SQL query to be executed on the lector database and respecting:

- a data to return, must match all fields in the table 'Lecturas' (select * from Lecturas)
- must be a SELECT query

Always after a message, end of line character appears, ASCII 10 character (\n).

Rewind response

The response to the request rewind (lector → pc), has the following format:

{"tipo":”REBOBINADO”,"rows":"yyy","msg":"xxx"}

Knowing the following:

- ‘yyy’ the number of rows that has resulted from the SQL query (the number of readings that will proceed to be sent by rewind socket)
- ‘xxx’ the message which has returned a database manager system (for the malformed query). If the query has been executed correctly, the message will contain 'OK' value.

If the query has generated any results, it is sent by the rewind socket. No further communication by the pc is necessary. Each of the data row sent, has the same format as the data interchanged by data socket. Always after a message, end of line character appears, ASCII 10 character (\n).

CONTROLL SOCKET

Description of the interchange protocol of the information by control socket. Socket TCP/IP by default work on port 9999. For the negotiated connection, the lector takes the server role, remaining in listening state to connection requests.

Messages from *Connector* to lector

Handshake message send to lector has the following format:

{"tipo":”HOLA”}\n

This message has to be responded by lector with *sync* request packet.

Information about new race data send to lector has following format:

{"tipo":”CARRERA”,"carrera":"xxxxx”,"split":"yyyyy”}\n

Knowing the following:

- xxxxx → new race name
- yyyyy → new split name

The *Connector* issues this message when user wants to change the configuration of a race of the lector. This message must be answered by lector with race configuration information packet.

Synchronization information for lector has the following format:

{"tipo":”SYNC”,"datetime":"xxxxx”}\n

Knowing that ‘xxxxx’ is a System.DateTime’ object with a date of the Connector pc.

This message is a response to the lector synchronization request. This message must be answered by lector with synchronization confirmation.

Messages from lector to *Connector*

CPU temperature alarm send to Connector has the following format:

{"tipo":”ALARMA”,"temperatura":"cpu|XX”}\n

Knowing that ‘XX’ is a CPU temperature.

Lector broadcasts this message every time when checks a temperature of the CPU, and it is greater than that established in the following firmware parameter:

<add key="cpu.temperatura.maxima" value="65"/>

No response awaiting.

Lector temperature alarm send to Connector has the following format:

{"tipo":”ALARMA”,"temperatura":"l|XX”}\n

Knowing that ‘XX’ is a Lector temperature.

Lector broadcasts this message every time when checks a temperature of the Lector, and it is greater than that established in the following firmware parameter:

<add key="cpu.temperatura.maxima" value="65"/>

No response awaiting.

Antenna configuration alarm send to Connector has the following format:

{"tipo":”ALARMA”,"antena":"XX”}\n

Knowing that ‘XX’ is a decimal value that indicates the status of the 4 antennas connected to a lector and has following meaning:

- XX = 00 (dec) = 0000 (bin) → no antenna connected
- XX = 01 (dec) = 0001 (bin) → antenna 1 is connected
- XX = 02 (dec) = 0010 (bin) → antenna 2 is connected
- XX = 04 (dec) = 0100 (bin) → antenna 3 is connected
- XX = 08 (dec) = 1000 (bin) → antenna 4 is connected
- XX = 03 (dec) = 0011 (bin) → antenna 1 and 2 are connected
- XX = 05 (dec) = 0101 (bin) → antenna 1 and 3 are connected
- XX = 09 (dec) = 1001 (bin) → antenna 1 and 4 are connected
- XX = 06 (dec) = 0110 (bin) → antenna 2 and 3 are connected
- XX = 10 (dec) = 1010 (bin) → antenna 2 and 4 are connected
- XX = 12 (dec) = 1100 (bin) → antenna 3 and 4 are connected
- XX = 14 (dec) = 1110 (bin) → antenna 2, 3 and 4 are connected
- XX = 13 (dec) = 1101 (bin) → antenna 1, 3 and 4 are connected
- XX = 07 (dec) = 0111 (bin) → antenna 1, 2 and 3 are connected
- XX = 11 (dec) = 1011 (bin) → antenna 1, 2 and 4 are connected
- XX = 15 (dec) = 1111 (bin) → antenna 1, 2, 3 and 4 are connected

Lector broadcasts this message whenever the Lector will communicate a change in the antenna settings.

No response awaiting.

Battery alarm send to Connector has the following format:

{"tipo":”ALARMA”,"bateria":"XXX”}\n

Knowing that XX is a decimal value that indicates, as a percentage, the state of charge battery and has following meaning:

- XXX = 100 → 100% available battery charge
- XXX = 45 → 45% available battery charge
- ...

It remains to define when lector deliver this message. It is not yet developed.

No response awaiting.

Electrical connexion alarm send to Connector has the following format:

{"tipo":”ALARMA”,"conexion":"X”}\n

Knowing that X is the binary value that indicates the state of the connection to the electricity network and has following meaning:

- X = 0 → not connected to the power supply
- X = 1 → connected to the power supply

It remains to define when lector deliver this message. It is not yet developed.

No response awaiting.

Information about new race data send to Connector has following format:

{"tipo":”CARRERA”,"carrera":"xxxxx”,"split":"yyyyy”}\n

Knowing the following:

- xxxxx → new race name
- yyyyy → new split name

Lector issues this message as a response once a communication is established with *Connector.*

No response awaiting.

Synchronization request send to Connector has the following format:

{"tipo":”SYNCREQUESTED”}\n

Lector broadcast this message in a given situations:

- at the beginning of the communication with *Connector*
- when from the display of TimingSense lector, time synchronization is requested again

Lector expects response with the message synchronization information from *Connector*.

Synchronization acknowledgment send to Connector has the following format:

{"tipo":”SYNC”,"offset":”xxxx”,"synctime":”yyyy”}\n

Knowing the following:

- xxxxx → ‘TimeSpan’ object with the offset that has needed the lector clock to synchronize with *Connector*. This is the offset of the lector relative to *Connector*.
- yyyyy → lector time in the moment of delivery of this message

This message is a reply to message synchronization information send from *Connector*.

APPENDIX 2 - COMMUNICATION BETWEEN CUBICSOFT AND PLATFORM

Communication between CubicSoft and online platform is base on web services. The discussed services can be divided into two groups: download data and upload data. In a communication process both response and request are encrypted by RIJNDAEL 256 algorithm, whole message of the post request is encrypted within the variable content. The message consist of JSON text format.

DATA DOWNLOAD

wsGetLicense

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array('user': usuario,'password': password)

Knowing the following:

- user – is an email
- password – is a user password

Response:

The system will return a JSON encrypted with the following structure:

array('error': error,'license': licencia,'finish': fecha,'idEmpresa': empresa,'idTimer': timer)

Knowing the following:

- error - can has following values: 0 - request completed successfully, 1 – incorrect user data, 2 – other error
- license – license number associated with the given user in varchar(32) format if request completed successfully
- finish - date of license validity in given yyyymmdd format
- idEmpresa – unique timing company identification in given 00000 format
- idTimer – unique timer identification in given 00000 format

wsValidateLicense

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array('license': licencia)

Knowing the following:

- license – is a application license

Response:

The system will return a JSON encrypted with the following structure:

array('error': error,'date': hoy,'finish': fecha)

Knowing the following:

- error - can has following values: 0 - request completed successfully, 1 – incorrect user data, 2 – other error
- hoy - today date on the server side in given yyyymmdd format
- finish - date of license validity in given yyyymmdd format

wsGetRacesList

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array('license': licencia)

Knowing the following:

- license – is a application license

Response:

The system will return a JSON encrypted with the following structure:

array('error': error,'tsdb': tsdb)

Knowing the following:

- error - can has following values: 0 - request completed successfully, 1 – incorrect user data, 2 – other error
- tsdb – JSON information *{"tsdb":{"races":[{"idRace":,"name":,"date":}]}}*

wsGetRace

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array('license': licencia,'race': carrera)

Knowing the following:

- license – is a application license
- race - is a race id of the given format 000001111122222 where 00000 corresponds to idEmpresa provided in *wsGetLicense service*, 11111 corresponds to idTimer provided in *wsGetLicense service,* 22222 corresponds to the id part of the race

Response:

The system will return a JSON encrypted with the following structure:

array('error': error,'tsdb': tsdb)

Knowing the following:

- error - can has following values: 0 - request completed successfully, 1 – incorrect user data, 2 – other error
- tsdb – JSON information *{"tsdb": {"races": {"idRace": "","idRaceType": "","name": "","organizer": "","description": "","country": "","province": "","city": "","date": "","finishTimeR": "","netTimeR": "'}"eventraces": [{"idRace": "","idEvent": "","description": "","dateTime": "","distance": "","cutoffTime": "","positionEqual": "","positionTimeType": "","teamResultType": "","teamTimeType": "","isMultiLap": "","cutoffType": "","cutoffLap": "","lapDistance": "","minLapTime": "","intermediates": "","eventsplits": [{"idRace": "","idEvent": "","idSplit": "","name": "","timerName": "","type": "","minTime": "","maxTime": "","minLapTime": "","lapNumber": "","multipleReads": "","calcTimes": "","finishStatus": "","distanceFromStart": "","deadZone": "","idBackup": "","backupAfter": "","backupTimeDifference": "","backupTimeWait": ""}],"eventcategories": [{"idRace": "","idEvent": "","idCategory": "","name": "","isAgeBased": "","fromAge": "","toAge": "","onDate": "","fromYear": "","toYear": "","gender": "","isPromotionCategory": "","maxPromotionCategory": "","idPromotionCategory": "","idPromotionCategory2": "","isSecondaryCategory": ""}],"eventattributes": [{"idRace": "","idEvent": "","idAttribute": "","name": "","description": "","type": "'"}],"eventgunwaves": [{"idRace": "","idEvent": "","idGunWave": "","waveDescription": "","aproxTime": "","waveTime": "","dorsalFrom": "","dorsalTo": ""}],"eventteamtypes": [{"idRace": "","idEvent": "","idTeamType": "","typeName": "","maxAthlete": "","resultAthleteCount": "","gender":*

"”}],"eventteams": [{"idRace": "","idEvent": "","idTeam": "","idTeamType": "","name": ""}]}]}}

wsGetInscriptions

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array(‘license’: licencia,‘race’: carrera)

Knowing the following:

- license – is a application license
- race - is a race id of the given format 000001111122222 where 00000 corresponds to idEmpresa provided in *wsGetLicense service,* 11111 corresponds to idTimer provided in *wsGetLicense service,* 22222 corresponds to the id part of the race

Response:

The system will return a JSON encrypted with the following structure:

array(‘error’: error,‘tsdb’: tsdb)

Knowing the following:

- error - can has following values: 0 - request completed successfully, 1 – incorrect user data, 2 – other error
- tsdb – JSON information *{"tsdb":{"inscription":[{"idRace": "","idEvent": "","name": "","surname": "","gender": "","age": "","birthday": "","country": "","provincia": "","city": "","code": "","address": "","dni": "","phone": "","mail": "","club": "",“dateCreate": "","lozalizador": "","idUser": "","idInscripcion": ","inscriptionattributes": [{"idRace": "","idEvent": "","idAthlete": "","idAttribute": "","value": ""}]}]}}*

DATA UPLOAD

wsSaveRace

Request:

In this service we receive by means of post request the following information encrypted in the variable content of the petition post:

array(‘license’: licencia,‘tsdb’: tsdb)

Knowing the following:

- license – is a application license
- tsdb - JSON information *{"tsdb": {"races": {"idRace": "","idRaceType": "","name": "","organizer": "","description": "","country": "","province": "","city": "","date": "","finishTimeR": "","netTimeR": "","unidades": "","timeZone": ""},"eventraces": [{"idRace": "","idEvent": "","description": "","dateTime": "","distance": "","cutoffTime": "","positionEqual": "","positionTimeType": "","teamResultType": "","teamTimeType": "","isMultiLap": "","cutoffType": "","cutoffLap": "","lapDistance": "","minLapTime": "","intermediates": "","idEventRace": "","eventsplits": [{"idRace": "","idEvent": "","idSplit": "","name": "","timerName": "","type": "","minTime": "","maxTime": "","minLapTime": "","lapNumber": "","multipleReads": "","calcTimes": "","finishStatus": "","distanceFromStart": "","deadZone": "","idBackup": "","backupAfter": "","backupTimeDifference": "","backupTimeWait": ""}],"eventcategories": [{"idRace": "","idEvent": "","idCategory": "","name": "","isAgeBased": "","fromAge": "","toAge": "","onDate": "","fromYear": "","toYear": "","gender": "","isPromotionCategory": "","maxPromotionCategory": "","idPromotionCategory": "","idPromotionCategory2": "","isSecondaryCategory": ""}],"eventattributes": [{"idRace": "","idEvent": "","idAttribute": "","name": "","description": "","type": "","value": ""}],"eventgunwaves": [{"idRace": "","idEvent": "","idGunWave": "","waveDescription": "","aproxTime": "","waveTime": "","dorsalFrom": "","dorsalTo": ""}],"eventteamtypes": [{"idRace": "","idEvent": "","idTeamType": "","typeName": "","maxAthlete": "","resultAthleteCount": "","gender": ""}],"eventteams": [{"idRace": "","idEvent": "","idTeam": "","idTeamType": "","name": "","finishTime": "","finishPosition": "","finishTeamPoints": "","finishStatus": ""}],"eventracenumbers":[{"idRace": "","idEvent": "","idRaceNumber": "","chipCode": ""}],"athletes": [{"idRace": "","idEvent": "","idAthlete": "","name": "","surname": "","gender": "","age": "","birthday": "","country": "","provincia": "","city": "","code": "","address": "","dni": "","phone": "","mail": "","club": "","dateCreate": "","lozalizador": "","idUser": "","idInscripcion": "","licencia": "","local": """idTeamType": "","teamName": "","dorsal": """eventathleteattributes": [{"idRace": "","idEvent": "","idAthlete": "","idAttribute": "","value": ""}]}],"eventathletes": [{"idRace": "","idEvent":*

"","idSplit": "","idAthlete": "","idRaceNumber": "","idGunWave": "","idTeam": "","idCategory": "","idSecCategory": "","isManualTime": "","splitTime": "","splitTimeNet": "","splitTimeLeg": "","overallPosition": "","overallPositionNet": "","overallPositionLeg": "","categoryPosition": "","categoryPositionNet": "","categoryPositionLeg": "","isPromCategoryP": "","secCategoryPosition": "","secCategoryPositionNet": "","secCategoryPositionLeg": "","isPromoSecCategoryP": "","genderPosition": "","genderPositionNet": "","genderPositionLeg": "","laps": "","averageSpeed": "","finishStatus": ""}]}]}}

Response:

The system will return a JSON encrypted with the following structure:

array('error': error)

Knowing the following:
error - can has following values: 0 – created request completed successfully, 1 – updated request completed successfully, 2 – given license does not exist, 3 – other error