Escola de Engenharia

Universidade do Minho

Hugo José Pereira Dias

Extending an Ontology-Based Personalized Dietary Recommendation for Weightlifting with Biomechanical Knowledge

Master Dissertation

Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Supervisors

Professor Paulo Cardoso

Professor Adriano Tavares

Acknowledgments

I’m grateful for the opportunity to create something useful and that makes possible to improve the others work.

I want to express my sincerely thanks to everyone who contributed to the development and conclusion of this dissertation work.

It was a great experience working with people outside of my knowledge area who gave me the background knowledge to be able to start my work specially Por Turnmark for her co-operation and Professor Filipe Conceição for his advices.

Also big thanks to Professor Paulo Cardoso for his guidance and Professor Adriano Tavares, who had the major influence on my dissertation work, a special thank you for all support and advices.

Abstract

On olympic weightlifting, just like on any high level competition or sport athletes are very competitive. World records are being beaten over and over and the winners are decided by details, that until recently were unnoted by the athletes themselves and their managers.

The focus of this dissertation is the development of a system to help olympic weightlifting athlete's to improve their performance and prevent injuries from the biomechanical analysis during the athlete's training sessions.

This dissertation presents a biomechanical ontology using a knowledge-based framework. Over this ontology will be applied a rule-based knowledge specific from olympic weightlifting that relates athlete's body positions and angles that are collected using an external system during the athlete's training sessions. From the inferred rules the system can provide information to the athletes and their managers if the athlete is doing the exercise correctly and at same time give some help to prevent injuries.

The system includes a database that saves the athlete's information and all data that is collected during the training sessions. It can be useful to verify the athlete progress comparing the actual and previous data.

This work is focused on olympic weightlifting, however it can be used for any kind of sport just by doing some adaptations.

This dissertation is a complement of another work focused on nutrition and menus recommendation for weightlifting.

Keywords: weightlifting, ontology, biomechanics, sports.

Index

Figure Index

Acronyms

3D – three dimensions

API – Application Programming Interface

OWL – Ontology Web Language

SQL – Structured Query Language

SQWRL – Semantic Query-enhanced Web Rule Language

SWRL – Semantic Web Rule Language

XLS - Extensible Stylesheet Language

XLST - Extensible Stylesheet Language Transformations

XML – Extended Markup Language

1. Introduction

Nowadays, competition in sports is stronger than ever and a small detail can make the difference between win or lose.

Technology is present on every kind of sport, not only to complement or help the judges or referees on their decisions, it can also detect details that can not be perceptible by the human and consequently decision can be taken quickly and accurately; but also because it can help athletes to improve their performance, prevent injuries and give to the managing teams feedback about the athlete's profile and performance in order to get the best training plan.

This dissertation presents the design and implementation of a system that analyses biomechanical data from weightlifting athletes, and based on that provides some information about the exercise and suggestions about changes that the athlete can make in order to improve his performance in competition.

1.1. Motivation

The creation of a system that analyses athlete's biomechanical behavior not only is benefic for them to improve their performance, but also for the prevention of serious injuries, that in some cases can endanger the professional career of an athlete.

Olympic weightlifting is also a sport that is growing globally, with more supporters and athletes on countries where there is no tradition in the practice of this sport, which makes it a sport with a great growth potential to the global scale.

So, a system like the one proposed in this thesis can have a high demand in this growing sport. It can also be adapted to work in other sports.

1.2. Contextualization

There is an increasing competitiveness on sports athletes, and weightlifting is no exception. Each phase of the exercise is well studied and prepared by the athletes and their managing teams. Data is collected during the training sessions and analyzed in detail to understand what the athlete is doing and where he has to improve in way to do an execution as perfect as possible.

Another subject that torments high level athletes are the injuries. An injury can take a promising athlete to stop competing for several months or even to abandon his career. The most common body part where the athletes suffer injuries is the back due to the high loads that athlete lifts combined with wrong posture.

The technologic systems that are used nowadays to collect data offer a precision that is impossible to human to detect, which is highly value for data analysis. However the decisions taken from the analysis of the data are responsibility of the managers because each athlete is different from another one, so each analysis has to be interpreted on a different way.

1.3. Objectives

The objective of this dissertation is the design and development of a system that analyzes the athlete biomechanics during the training sessions and provides some recommendations to the athlete to improve his posture and consequently improve his performance and prevent the possibility of the appearance of severe injuries.

Initially we will study the biomechanical domain of weightlifting athletes, followed by the design and implementation of an ontology that represents the biomechanical domain.

The ontology will receive some queries represented on a rule-based model. This model relates ontology properties applying one or more requirements. The result of each query depends if the requirements were fully satisfied or not.

A database will be created to save the important information about each athlete as well as every training session data to be possible to visualize after the athlete's behavior. It will also be implemented access to his historic data and consequently check his progress.

To implement the communication between the ontology, the database and the user it will be created an application with graphic environment based on JAVA programming language. The communication between the queries and the ontology the will made using the OWL API. The queries will be written on SQWRL query language.

1.4. Structure of the Thesis

The second chapter refers the state of art. Here we delve into the biomechanical knowledge of weightlifting athletes. We will present some study cases and other work already made on this area. It will also make an approach to all programming languages that will be used to create the system and in particular the ontology.

The third chapter describes the system analysis. We will do a study of the system, separating it on subsystem and check its requirements and constraints.

On fourth chapter we will study the system design, studying deeper how each subsystem works and how it interacts with another subsystem. It will be shown all system functionalities step-by-step using block diagrams.

The fifth chapter is the implementation chapter. We will show the implementation of all subsystems and the interactions between them.

The results obtained from the system implementation will be shown on sixth chapter. There will be used data from several athletes to analyze and compare the results.

Finally on seventh chapter we will discuss the obtained results. Here will be also present the conclusion of this dissertation work, future work and possible improvements and modifications to do to work with another sports.

2. State of Art

2.1. Olympic Weightlifting

Olympic-style weightlifting also called olympic weightlifting or just weightlifting is a sport in which athlete attempts to lift a maximum-weight of a bar loaded.

There are two competitions in weightlifting, snatch and clean and jerk.

Snatch consists of lifting the barbell from the floor to an overhead position in one single movement. clean and jerk, otherwise consist of lifting the barbell from the floor to an overhead position in two movements. The movements consist of lifting the barbell from the floor to the shoulders and then from the shoulders to overhead.

2.1.1. Bodyweight Categories

Weightlifters compete in bodyweight categories which are different for men and women.

Men's weight Category:	**Women weight Category:**
Under 56 kg;	Under 48 kg;
Under 62 kg;	Under 53 kg;
Under 69 kg;	Under 58 kg;
Under 77 kg;	Under 63 kg;
Under 85 kg;	Under 69 kg;
Under 94 kg;	Under 75 kg;
Under 105 kg;	Over 75 kg;
Over 105 kg;	

2.2. Biomechanics on Weightlifting

Unlike other strength sports, which test limit strength, weightlifting tests explosive strength, the lifts are executed faster and with more mobility and a greater range of motion during the execution of the exercise.

2.2.1. The Snatch

As was mentioned before, the snatch involves lifting the weight from the floor, catching it overhead in a squatting position, and then driving it upward to a standing position [1].

Snatch includes six phases. On *start position* both knees and ankles shall be aligned, hip's position must be higher than knees, shoulders are aligned with the bar or slightly ahead and the back is curved.

The *first pull* is initiated when the lifter extends their knees raising the barbell from the floor to a position below or at same height than knees.

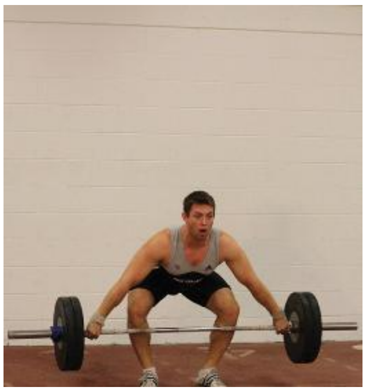


Figure 1 - Snatch 1st Pull

Transition also referred as *double-knee bend* happens when the barbell moves up over the knees and the lifter moves to a near vertical position.

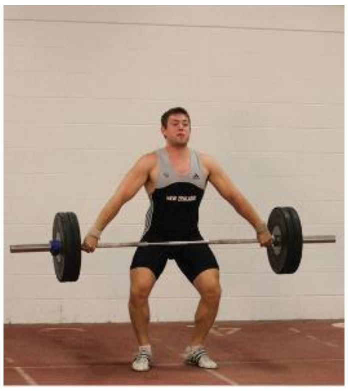


Figure 2 - Snatch Transition

During the *second pull* barbell reaches its maximum acceleration. Simultaneously athlete shrugs his shoulders and extends the hip, knees and ankles.

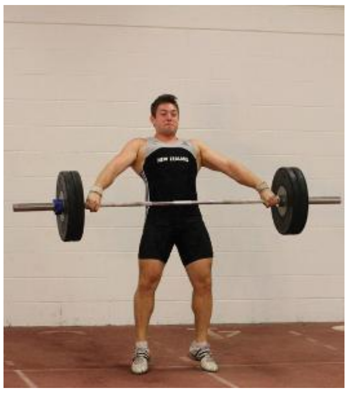


Figure 3 - Snatch 2nd Pull

On *turnover*, as barbell raises on a vertical plane, athlete begins to his body underneath the barbell [2].

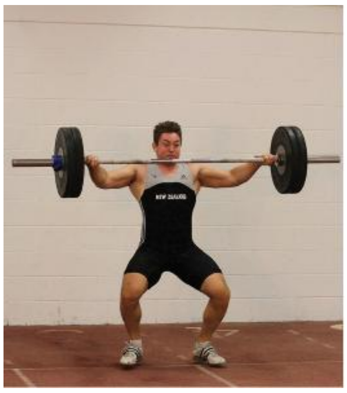


Figure 4 - Snatch Turnover

On *catch* position, athlete holds the barbell in a straight-arm overhead position while flexing the knees and hip, reaching a full squat position. After, athlete moves from a squat to a standing position, maintaining the barbell overhead.

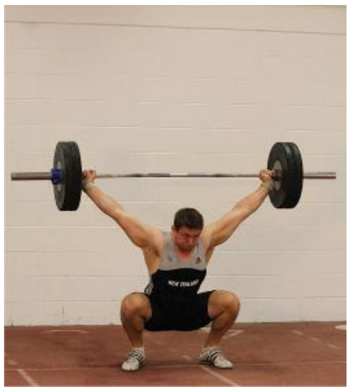


Figure 5 - Snatch Catch

2.2.2. The Clean and Jerk

Clean and jerk is a two-part exercise and its divided in two parts, the clean and the jerk.

The *clean* requires to athlete to lift the barbell from the platform to the shoulders height in one single movement. It is subdivided in six phases.

The biomechanical principles of the first four phases (*start position*, *first pull*, *transition* and *second pull*) are the same principles as those of the *snatch*.

Lifter initiates the *turnover* phase when barbell rises to 55% - 65% of lifters height.

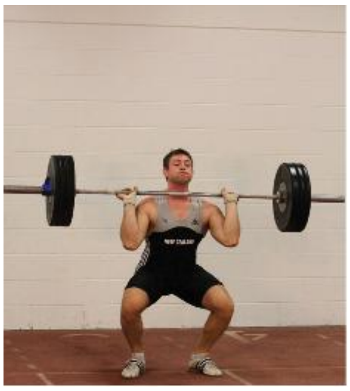


Figure 6 - Clean Turnover

On *catch*, athlete receives the barbell on his shoulders and descends into a squat position. Lifter starts preparing the *jerk*.

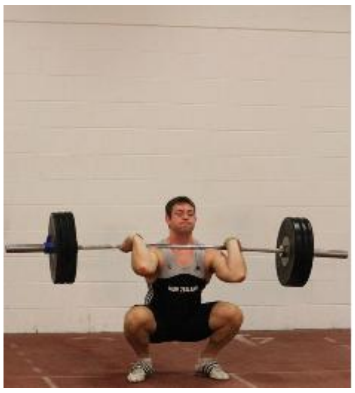


Figure 7 - Clean Catch

The *jerk* has also six phases, *start*, *dip*, *drive*, *unsupported split*, *supported split*, *recovery*.

On *start* phase lifter and barbell become motionless.

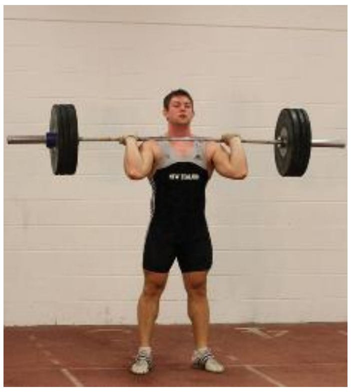


Figure 8 - Jerk Start Position

Dip starts when lifter starts flexing his knees and hip with barbell over the shoulders.

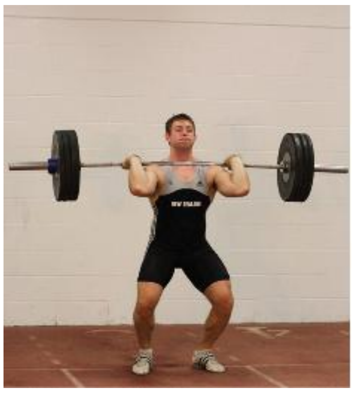


Figure 9 - Jerk Dip

At *drive*, athlete is required to accelerate the barbell in vertical plane.

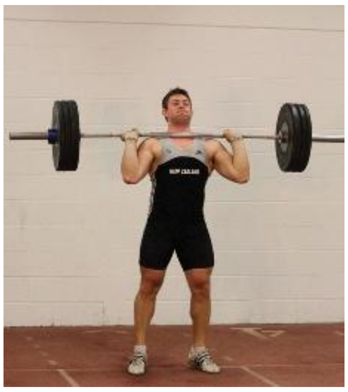


Figure 10 - Jerk Drive

At *unsupported split* moves vertically off the shoulders and the lifter's feet leave the ground.

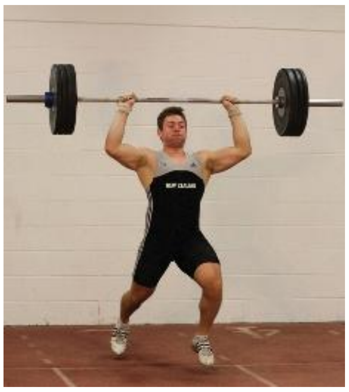


Figure 11 - Jerk Unsupported Split

Lifter is in the *supported split* when his feet get again in contact with the floor and barbell is overhead with the arms fully extended.

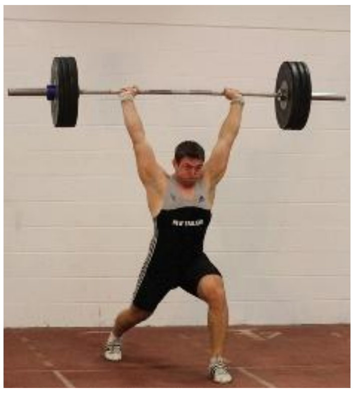


Figure 12 - Jerk Supported Split

The *recovery* happens when athlete gets motionless and his feet are parallel one to another.

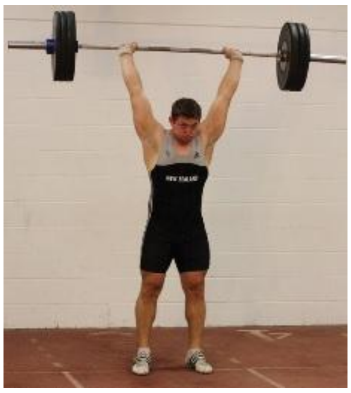


Figure 13 - Jerk Recovery

2.3. Injuries on Weightlifting

On weightlifting, like in other sports, injuries are a serious problem for athletes. The most common injuries are relates the high loads and a bad execution of the exercise.

These are the most common injuries and syndromes on weightlifting.

- **The Tired Neck Syndrome:** Some athletes perform an inordinate amount exercises to exclude the execution of other complementary movements. The resulting imbalance includes tight pectoralis minor and external rotary shoulder muscles.

- **The Thoracolumbar Syndrome:** This syndrome is caused by compressive and large flexion bending moments in the early lifters training life.

- **Sacroilac Joint Dysfunction:** Sacroiliac joints are situated between the spine and the lower extremities. This type of dysfunction is caused by a wrong pelvis movement during the lift of high loads.

- **Extremity Injuries:** Upper and lower extremities are at risk of injury during weightlifting activities. Upper extremities injuries usually occur when lifter is not using a correct technique or results from overtrained muscles not prepared to lift heavy loads.

- **Cardiovascular consequences:** Weightlifter's blackout typically occurs after a squat movement as a result of blood pooling in the lower extremities. This can be avoided using proper breathing techniques [1].

2.4. Relevant Systems for Data Collecting and Analysis

As known so far, there is no automated systems for data analysis dedicated exclusively to the sport of weightlifting.

Recently were designed some systems for weight lifting as body building which has not the same purpose of weight lifting as olympic sport. Most of existing devices are for personal use and are wearable devices or smartphone applications which analysis is based on data received from accelerometers and heart beating sensors.

For professional use, the most common systems used are system for data collection which collect data during the exercise and in the end traces graphics with some parameters along time. Systems with most accuracy are composed by a set of points that are placed in the athlete's body, each point refers to a body part, and a set of cameras which traces the points along the exercise.

In all cases the data analysis must be done by humans.

The system that will be used for data collecting for this dissertation work is based on camera's system mentioned above.

2.5. Nutrition Work

This dissertation work is part of a major work composed also by a based personalized dietary recommendation work. The nutrition system is also a knowledge-based work. Its goal is to give to athletes nutritional menus recommendations based on their training plans, nutritional needs and personal preferences.

During the work was developed ontology with rule-based knowledge to provide specific menus for different times of the day and different train phases for athlete's diary nutritional needs and athlete's preferences for better sport performance.

The main components of this system are the food and nutrition ontology, the athlete’s profiles and the nutritional rules for sports athletes.

2.6. OWL Ontologies

Ontologies are used to capture knowledge about some domain interest.

From the philosophy, ontology studies the categories and their relations and deals with questions about what entities exists, how they are grouped, their hierarchy and the categories which subdivide them. In other words, ontologies describe a domain, its concepts and the relationships between those concepts.

OWL (Web Semantic Language) is a computational logic-based language. The goal of OWL Language is to exploit knowledge using computer programs. OWL documents are also known as OWL ontologies.

OWL is used for authoring ontologies. It will be used this ontology language because OWL allows the import of other ontologies, adding information from the imported ontology to the current ontology which means that using OWL it is possible to join biomechanical information to nutritional ontology.

2.6.1. OWL Sub-languages

OWL ontologies can be categorized into three sub-languages.

OWL-Lite: OWL-Lite is the simplest OWL sub-language. It is used where a simple class hierarchy and simple constraints are needed.

OWL-DL: OWL-DL is more complex than OWL-Lite and is based on description logics [3]. It is used for creating ontologies with large expressivity and at same time conclusive and computable.

The ontology created for this dissertation work was programmed using OWL-DL.

OWL-FULL: OWL-FULL is the most expressive OWL sub-language. It is used when high expressiveness is required, however there is no guarantee that can be computable.

2.6.2. Components of OWL Ontologies

OWL ontologies have three main components, classes, properties and individuals. In way to ontology to have the best performance these components names should be atomic.

2.6.2.1. Classes:

Classes are collections that contain objects, or abstract groups. Classes describe concepts of a specific domain. They can contain subclasses describing even more a specific concept.

2.6.2.2. Properties

Properties are related to individuals or classes. They are divided in two.

Data Properties: Data properties are used to assign values to individuals or classes.

Object Properties: Object properties are binary relations on individual or classes. This type of properties can have an inverse.

2.6.3. Individuals

Individuals are instances of classes and represent object in domain that is being studied.

2.7. SWRL

SWRL means Semantic Web Rule Language and is used to express rules as logic. It can be used over OWL-DL and OWL-Lite.

This rule language will be used create rules that contain data from training sessions and constraints to comply. Rules will be inferred to ontology that validates the rule if its data complies with its constraints.

SWRL rules are divided in two parts, the antecedent, also called body, and the consequent or head, and it has the form of:

[*antecedent*] -> [*consequent*]

Rule bellow asserts that if a person (p) who has a brother(x) and a child (y), then the individual *X* is uncle of individual *Y.*

$$\underbrace{\mathit{Person}(?p)\ \text{^}\ \mathit{hasBrother}(?p, ?x)\ \text{^}\ \mathit{hasChild}(?p, ?y)}_{\text{antecedent}}\ \text{->}\ \underbrace{\mathit{hasUncle}(?y, ?x)}_{\text{consequent}}$$

2.8. SQWRL

SQWRL (Semantic Query-enhanced Web Rule Language; pronounced *squirriel*) is built on the SWRL language. SQWRL takes a standard SWRL rule antecedent and effectively treats it as a pattern specification for a query [4].

This language allows querying OWL ontologies and it has the same operators than SWRL allowing the creation of rules inside queries. There are also SWRL editors that generate SQWRL code.

On this dissertation work, SQWRL will be used to query ontology with rules about data collected from athletes training sessions.

2.9. Protégé

Protégé is a free open-source framework focused for development of knowledge-based systems using ontologies.

The use of Protégé for the development of our ontology is due to the fact that Protégé allows the ontology's development in OWL language and because it is worldwide used there is a lot of support and documentation.

Protégé also provides API's for Java applications to communicate to ontologies.

2.10. Java Programming Language

Java is an object oriented programming language developed in the early of 90s. Java applications are usually compiled to bytecode. Unlike low level programming languages, Java doesn't depend of computer architecture because it runs over a virtual machine.

System will be programmed using Java due to *Protégé* API's were written for this language and as it is used globally, there is a large documentation that can be used.

2.11. MySQL

MySQL is an-open source relational database management platform. It is the second most widely used system for database management.

MySQL development project has made its source code available under terms of the GNU General Public License.

Database will be created using this system because it is open-source and there is a previous knowledge on working with that.

For this dissertation work database will be used to store data from athlete's personal profiles and from training sessions.

3. Project Analysis

3.1. System Overview

The goal of the system is to provide recommendations to athletes on how to improve their posture during the exercise based on data that is collected during the training sessions and some biomechanical rules.

Figure 14 represents the system's block diagram and gives an overview on how system works. Users can insert the athlete's personal and biomechanical data collected during training sessions on the application using its graphical interface. Then, the application will fill some rules already given to the application with the biomechanical data. Based on that rules, the application, then will query the ontology through its API (Protégé-OWL) that works as an inference agent.

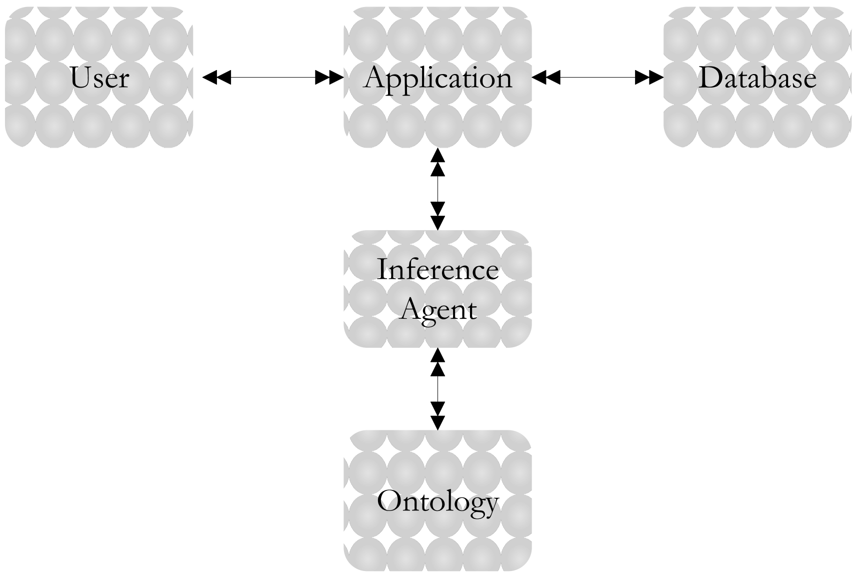


Figure 14 – System's Block Diagram

3.2. System Constraints

The system has a very little number of constraints:

- Protégé framework to create the ontology.
- The ontology must be compatible with the nutrition recommendation ontology.

3.3. System Requirements

The system requirements are divided in two, functional requirements and non-functional requirements.

Functional requirements:

- The user must be able to insert athlete's personal information;
- The user must be able to insert biomechanical information collected during the training sessions;
- User must be able to ask to system for recommendations.

Non-functional requirements:

- From the inserted biomechanical data, the system must apply rules based and query the ontology;
- From the inferred rule, the system must give a report with some recommendations to the user;
- The system must respond as fast as possible;
- The user interface must be user friendly.

3.4. Reading biomechanical information

Since the goal of this system is not collect biomechanical data, we will collect that data using an external system. There are a large number of systems available on market to collect data and each one uses a different type of technology to the reading of the athlete's posture.

The user needs to insert on our system over than 200 parameters that were collected from the external system and that can take a lot of time, can be tiring, and the user can enter wrongly some parameter and consequently the results are not what should be expected. So the resolution of this problem can be the use of something that saves all the parameters directly from the collecting data system to ours. The majority of the systems that are used to collect data can export the data to a file.

We choose to read the biomechanical information using *xls* files. The reason for that choice is that every collecting data system can export the biomechanical information to this file extension. Besides that, the *xls* are used worldwide and there are programs to read this file extension accessible to everyone and for free.

3.5. Use Cases Diagram

Figure 15 represents the use cases diagram for the personal information window.

The biomechanical information is read directly from a file, so there is no need of a user case diagram for that.

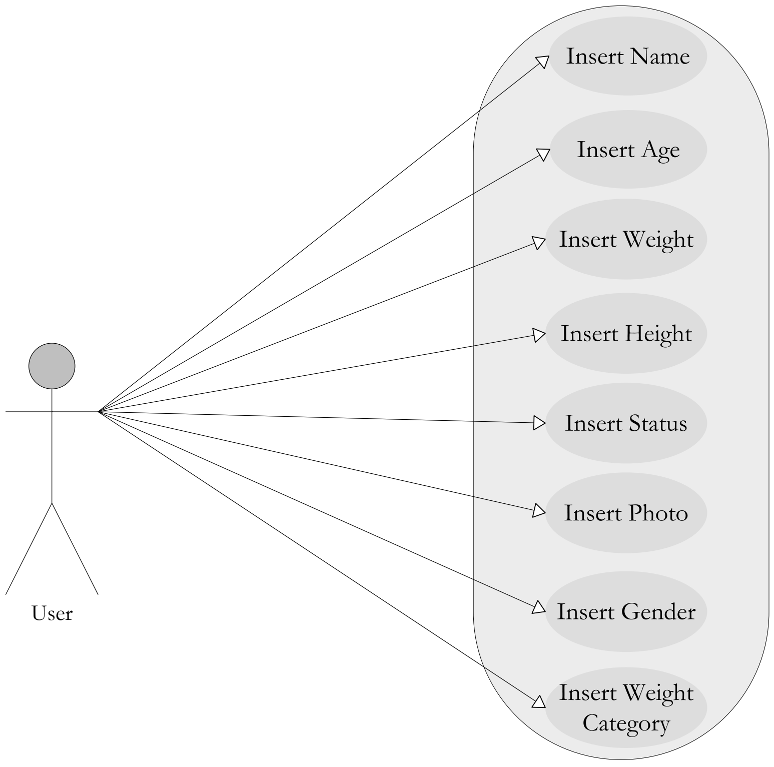


Figure 15 – Use Case Diagram for Entering Personal Data

3.6. Sequence Diagram

On Figure 16 we can see the system sequence diagram when the user runs the application to evaluate the biomechanical data.

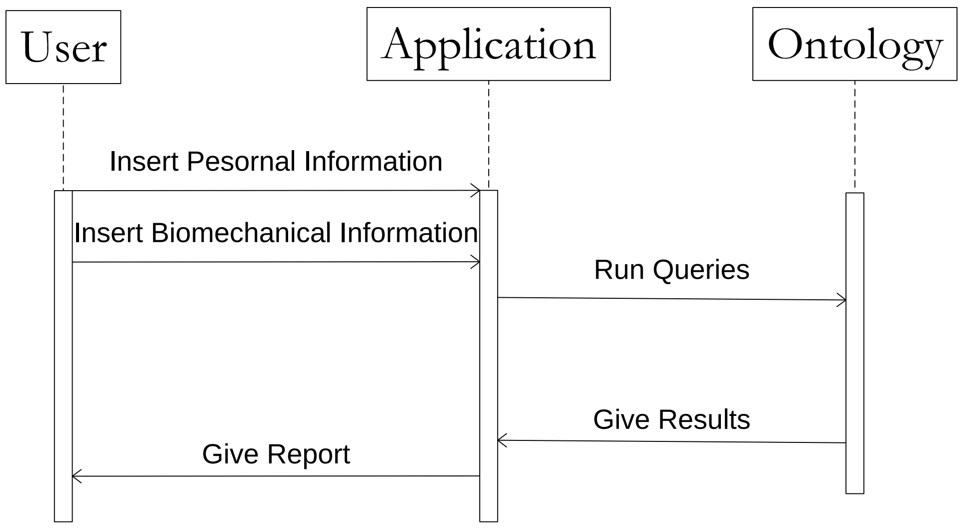


Figure 16 – Sequence Diagram for Biomechanical Data Evaluation

Figure 17 represents the sequence diagram for saving data on database.

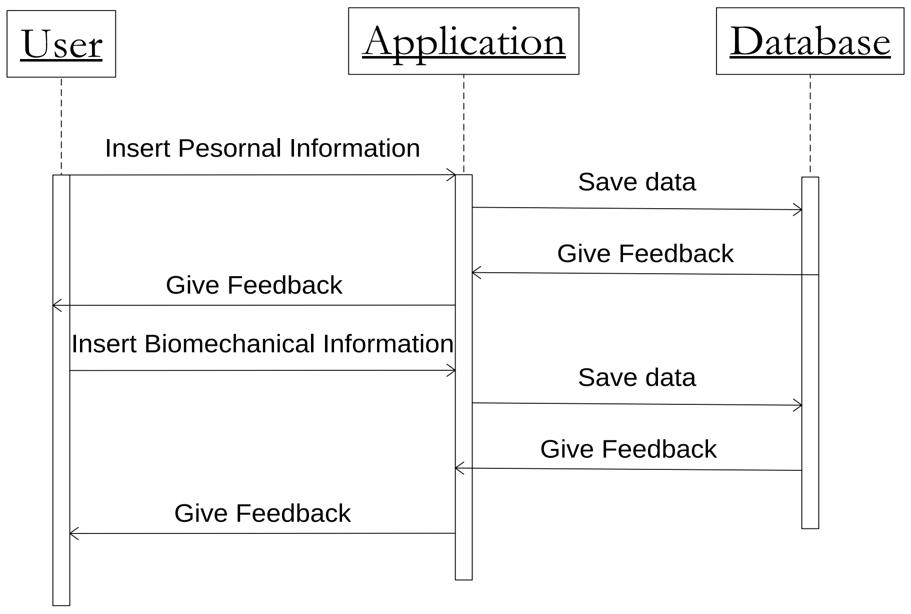


Figure 17 – Sequence Diagram for Saving Information

3.7. Ontology Rules

Rules will be written in SWRL language and will be inferred to ontology. They refer to movements and body postures that athlete needs to have to comply in order to execute the exercise correctly.

These rules contain data from athlete's training session and constraints. In order to verify a correct execution they need to be satisfied.

Example:

<u>Rule:</u> On *first pull*, barbell position has to be at a lower position than knees.

<u>SWRL Translation:</u> *Athlete(?a) ^ Barbell(?b) ^ barbellPosition1stPull(?b,?x) ^ kneesPosition1stPull(?a,?y) ^swrlb:greaterThan(?y,?x) -> RuleOk(?a,true)*

4. System Design

4.1. Ontology Design

4.1.1. Ontology Overview

Ontology design starts with design of the main concepts. Figure 18 shows the main concepts and the relationships between them.

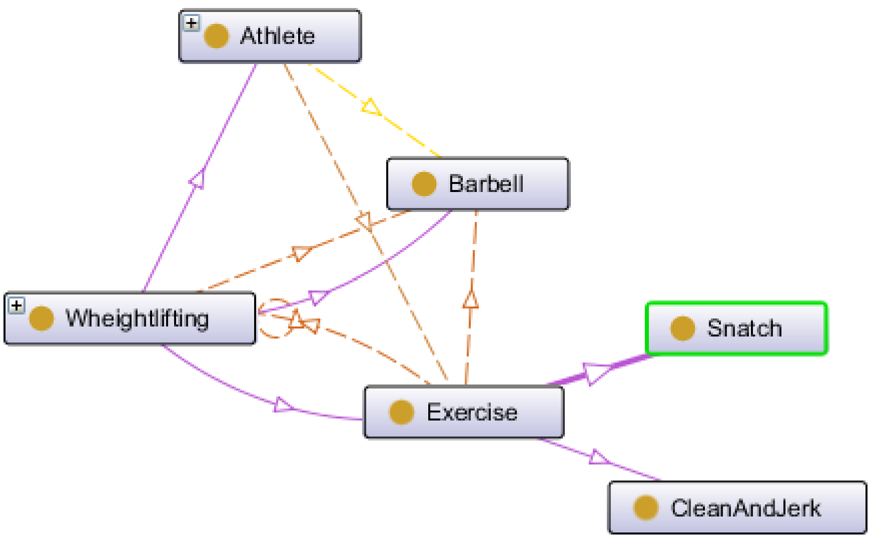


Figure 18 – Ontology Overview

Athlete class represents the athlete's profile containing its properties such as name, age, weight, etc..

Barbell class represents the barbell that is lifted during the execution of the exercise.

Exercise class represents the exercise itself and its subdivided in two other classes, *Snatch* and *Clean and Jerk* which represent each competition of the olympic weightlifting.

4.1.2. Properties

As was described above there are two types of properties. *Datatype* properties are used to describe class properties and assign concrete values to instances of class. *Objecttype* properties are used to describe the relationship between classes.

ObjectType **properties:**

- ***practices:*** relates *Athlete* class to *Exercise* class and assures that the athlete always practices the exercise.

  Domain: *Athlete*
  Range: *Exercise*
  Form: *Athlete practices Exercise*

- ***lifts:*** relates *Athlete* class and *Barbell* class.

  Domain: *Athlete*
  Range: *Barbell*
  Form: *Athlete lifts Barbell*

- ***has:*** relates *Exercise* class and *Barbell* class.

  Domain: *Exercise*
  Range: *Barbell*
  Form: *Exercise has Barbell*

- ***isPracticedby:*** is the inverse property of *practices.*

  Domain: *Exercise*
  Range: *Athlete*
  Form: *Exercise isPracticedBy Athlete*

- ***isliftedBy:*** is the inverse property of *lifts.*

  Domain: *Barbell*

  Range: *Athlete*

  Form: *Barbell isPracticedBy Athlete*

- ***isPartOf:*** is the inverse property of *has.*

  Domain: *Barbell*

  Range: *Exercise*

  Form: Barbell *isPartOf Exercise*

***Datatype* properties:**

In some cases we are working with three-dimensional variables, so what we do is creating a *mother property* and provide it with the corresponding attributes and next create three descendent properties to represent each 3D axis.

There are also properties that used on the six positions of the lifting. The *modus operandi* is the same that was used to represent the 3D axes. We create a *mother property* and then we create the descendent properties representing each one of the lifting positions.

From the merge of these two topics, the resulting property is shown of Figure 19.

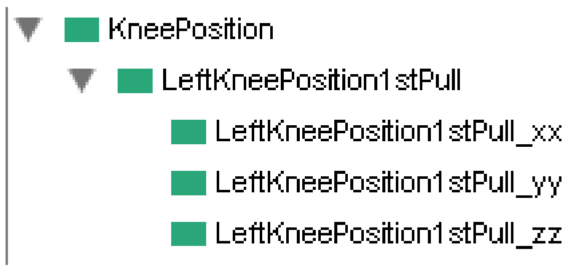

KneePosition
LeftKneePosition1stPull
LeftKneePosition1stPull_xx
LeftKneePosition1stPull_yy
LeftKneePosition1stPull_zz

Figure 19 – Descendent *DataType* Properties

4.1.3. Rules

Besides the rules are part of the ontology we choose to remove it from the core of the ontology because the rules can be changed at any moment. If the rules were built-in on the ontology every time we want to change or add some rule we need to do it directly on the ontology and if it is not done correctly we can compromise not only the rule itself but all the ontology work.

Instead, the rules are on a separated file written on XML programming language that is invoked every time the application starts to load the rules to the application memory.

The rules are after applied on the ontology in form of queries that we will describe in following sections.

4.2. Application Design

4.2.1. Application Overview

Application starts by reading and parsing the xml file and transform the rules from xml language in to SQWRL language. Following, the system loads all athletes profile and sessions saved on database.

The first window that is presented to user has a list of all athletes that were loaded from the database. When the athlete name is clicked the program shows all personal information about him. User can change the athlete's information using a window which was created for that purpose. User can also add new athletes to the program.

It is also possible to add new sessions to athlete's profile. After inserting a new session, user has the possibility to run the ontology immediately or just save session data on database.

After running the ontology is presented to user a list showing all rules. Depending if the rule was satisfied or not, meaning that athlete did the exercise correctly or not, the rules are presented with different colors.

Figure 20 presents the application overview diagram.

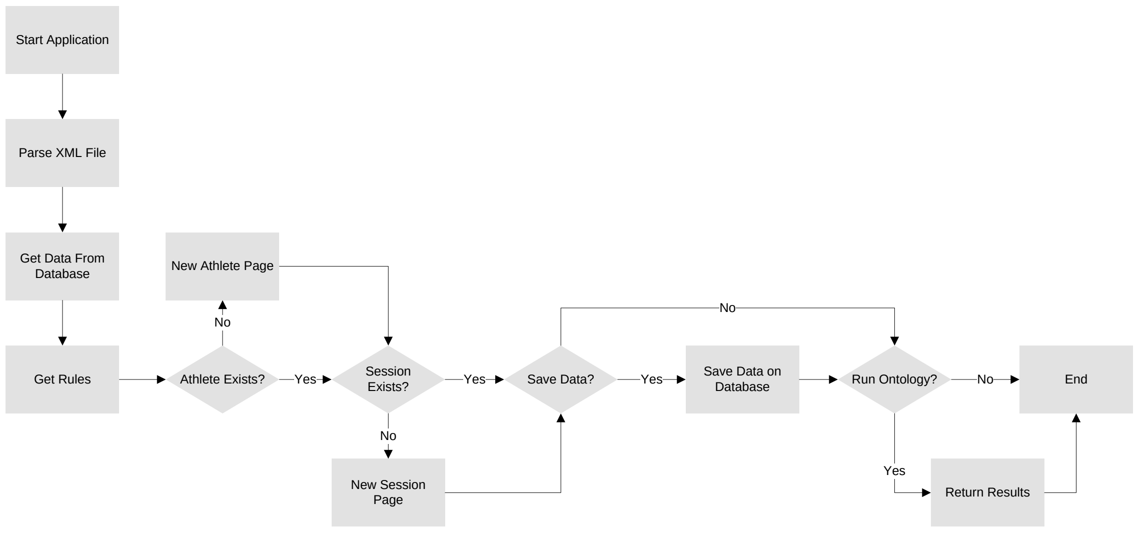


Figure 20 – Application Overview

4.2.2. Main Window Subsystem

The main window is the first window that the user will see when the application starts. Here the user can see all athletes that are on the system as well as their personal information.

There are also buttons to access to other functionalities such as creating new athletes or sessions, edit athlete's personal information and deleting existing athletes.

The Figure 21 shows the main window flowchart.

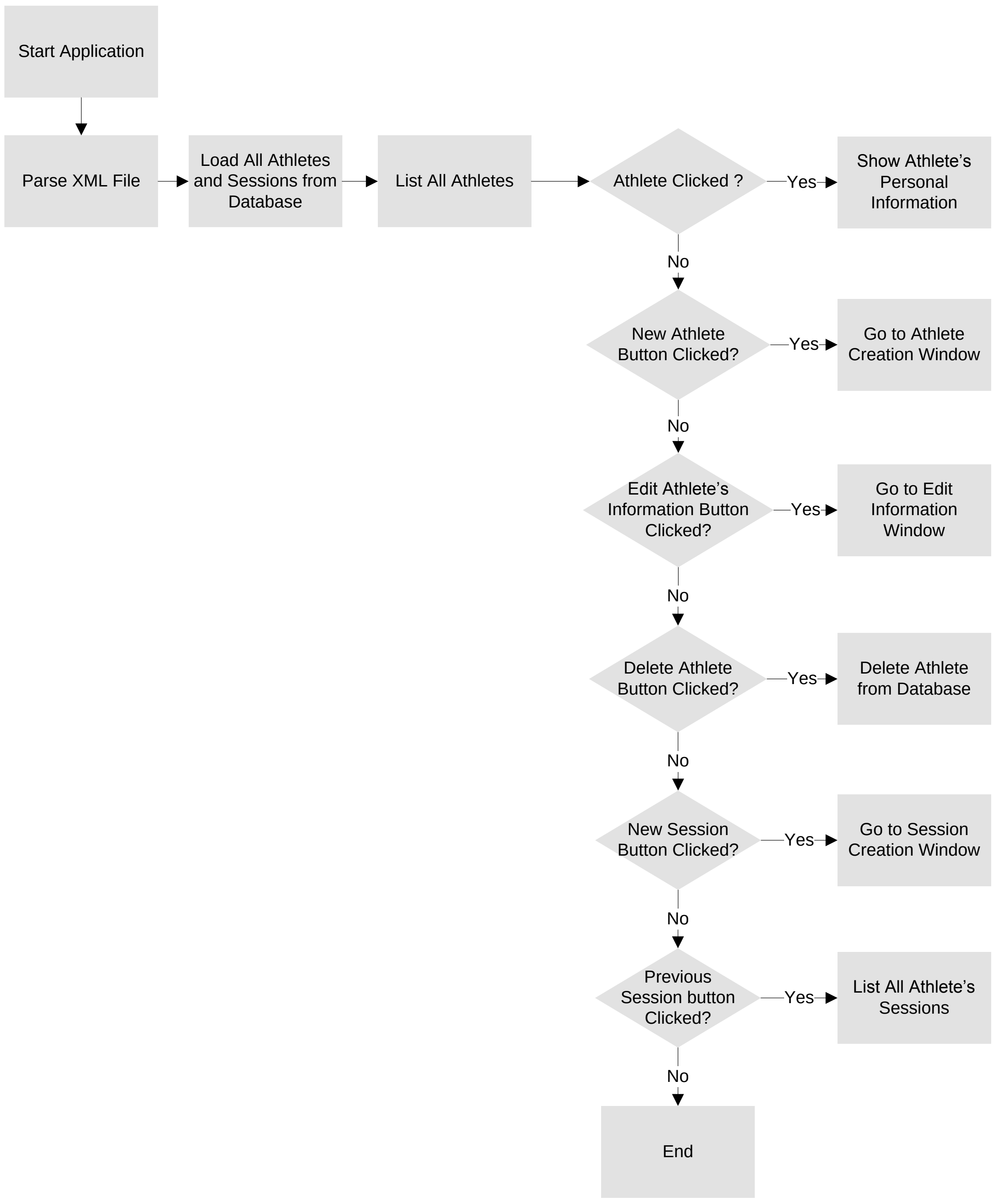

Start Application
Parse XML File
Load All Athletes and Sessions from Database
List All Athletes
Athlete Clicked ?
Yes
Show Athlete's Personal Information
No
New Athlete Button Clicked?
Yes
Go to Athlete Creation Window
No
Edit Athlete's Information Button Clicked?
Yes
Go to Edit Information Window
No
Delete Athlete Button Clicked?
Yes
Delete Athlete from Database
No
New Session Button Clicked?
Yes
Go to Session Creation Window
No
Previous Session button Clicked?
Yes
List All Athlete's Sessions
No
End

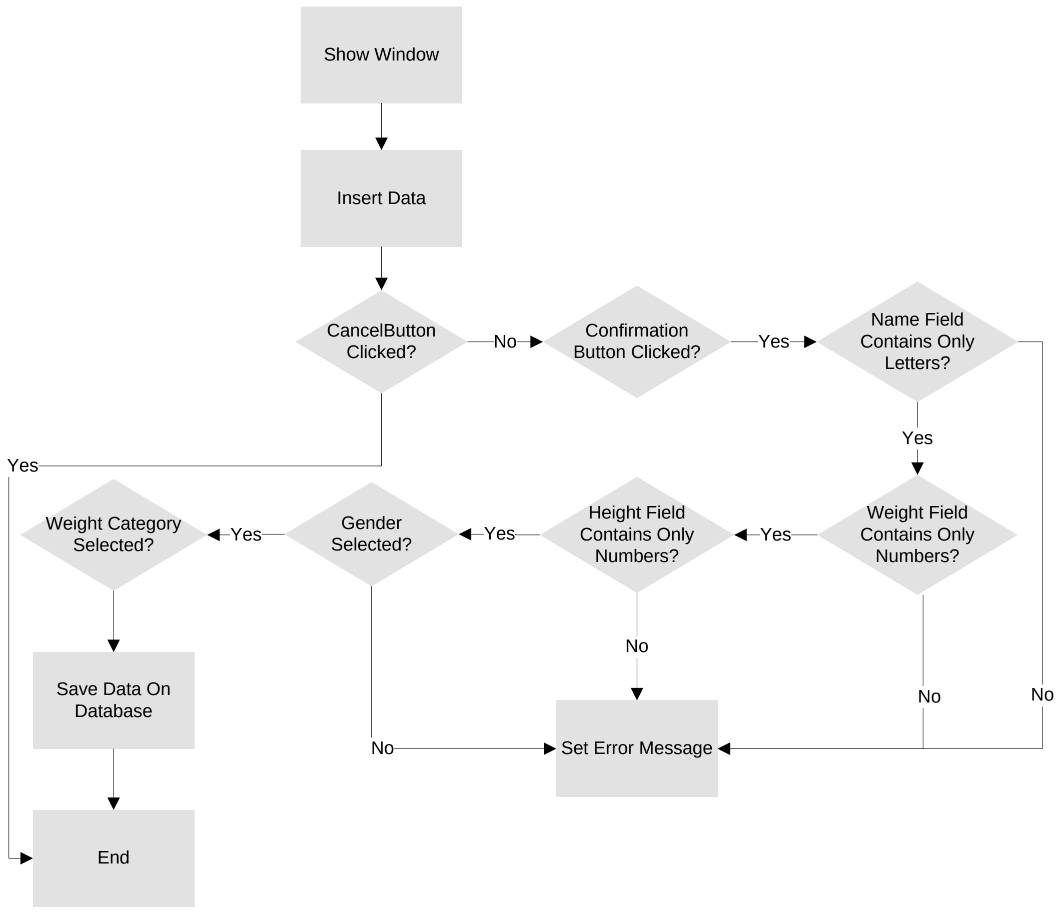
Figure 21 - Main Window Subsystem Flowchart

4.2.3. New Athlete Subsystem

On New Athlete interface window user can insert new athlete's personal information.

When information inserted is confirmed the system verifies if that information is consistent, for example the weight and height fields are only number characters. Name, weight, height, gender and weight category fields are mandatory because some rules depend on these data.

Figure 22 shows the flowchart of this subsystem.



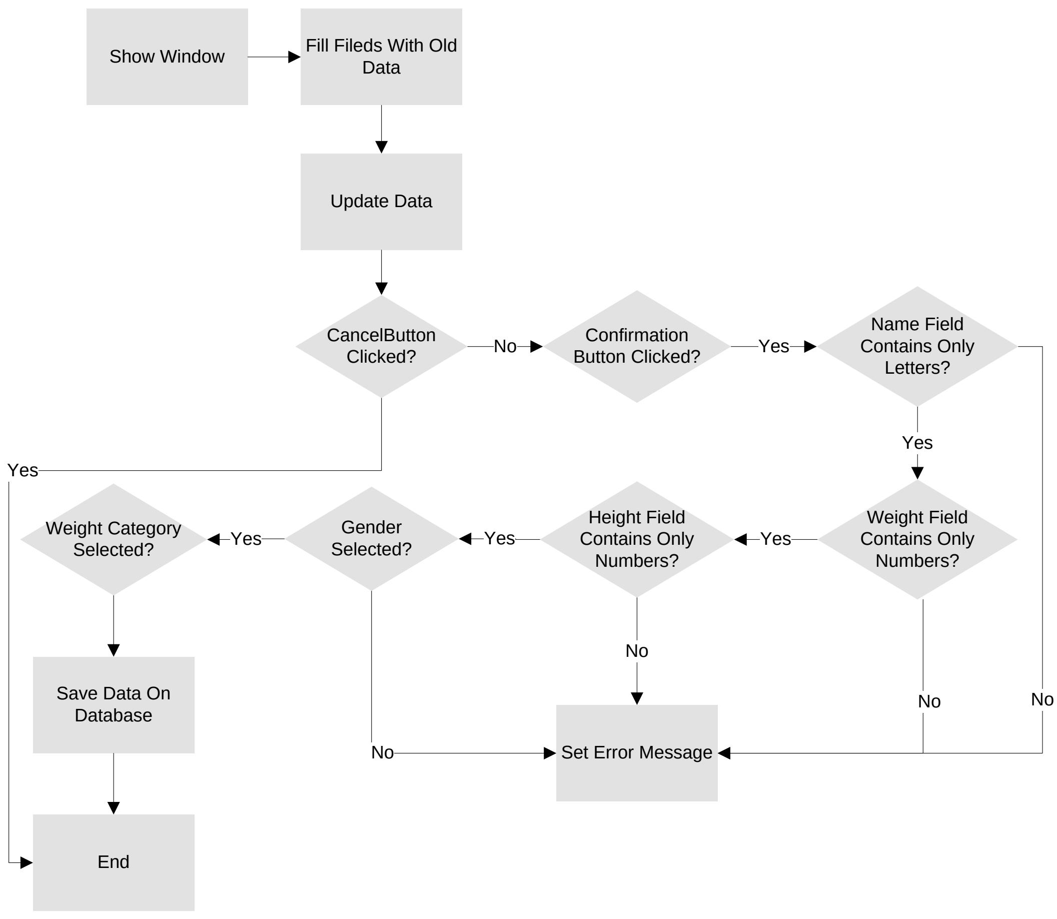
Show Window
Insert Data
CancelButton Clicked?
No
Confirmation Button Clicked?
Yes
Name Field Contains Only Letters?
Yes
Yes
Weight Category Selected?
Yes
Gender Selected?
Yes
Height Field Contains Only Numbers?
Yes
Weight Field Contains Only Numbers?
No
No
No
Save Data On Database
No
Set Error Message
End

Figure 22 - New Athlete Subsystem Flowchart

4.2.4. Edit Athlete Information Subsystem

This subsystem is much like the New Athlete Subsystem. The only difference is that the fields are filled with the athlete's information to be updated instead blank like on New Athlete Subsystem.

Figure 23 shows the flowchart of this subsystem.



Show Window
Fill Fileds With Old Data
Update Data
CancelButton Clicked?
No
Confirmation Button Clicked?
Yes
Name Field Contains Only Letters?
Yes
Yes
Weight Category Selected?
Yes
Gender Selected?
Yes
Height Field Contains Only Numbers?
Yes
Weight Field Contains Only Numbers?
No
Save Data On Database
No
No
No
Set Error Message
End

Figure 23 - Edit Athlete Information Subsystem

4.2.5. Delete Athlete Subsystem

Athlete’s profiles can be deleted. For that there is a button on main window.

When user clicks on delete button the application sends a query to database to remove the athlete with the id given.

Figure 24 represents the flowchart of this subsystem.

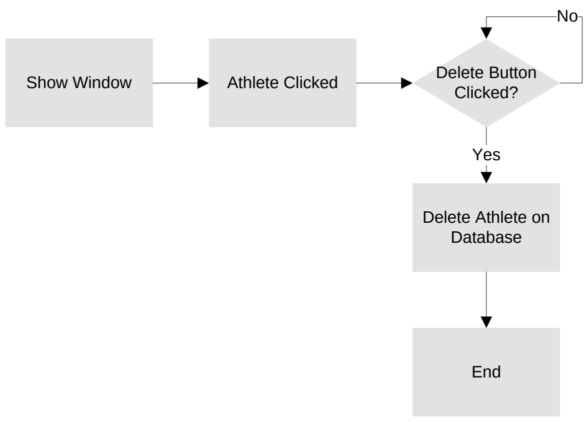


Figure 24 - Delete Athlete Profile Subsystem Flowchart

4.2.6. New Session Subsystem

The New Session Subsystem allows user to add new training sessions data to an athlete.

It was said before that the session's data is obtained from an external system on a .xls file, so our concern is only how to read the file.

The XLS file reader subsystem will be explained bellow.

The user is able to add a date and time to session and also some notes that he consider to be important.

Figure 25 shows this subsystem flowchart.

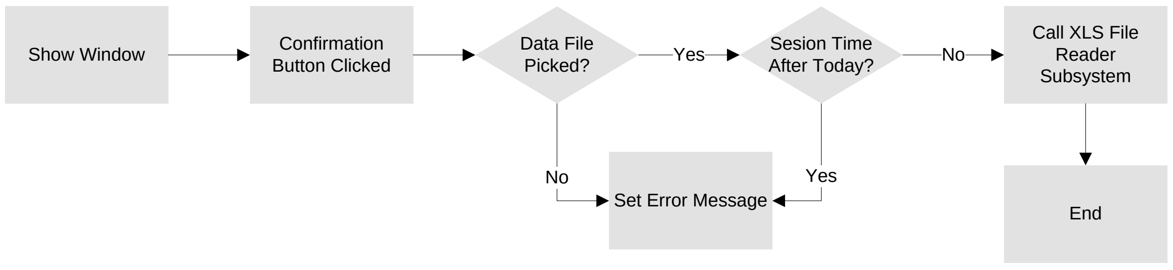


Figure 25 - New Session Data Subsystem Flowchart

4.2.7. Sessions Listing Subsystem

This subsystem is used to list all sessions of a given athlete stored on database and present it to the user.

From this list, user can select a session to run on the ontology.

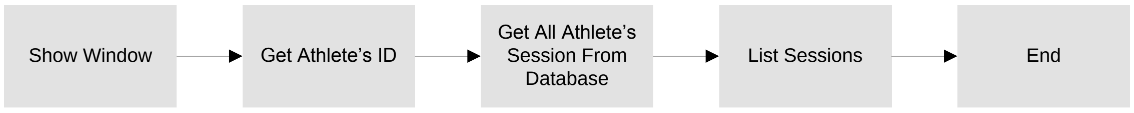


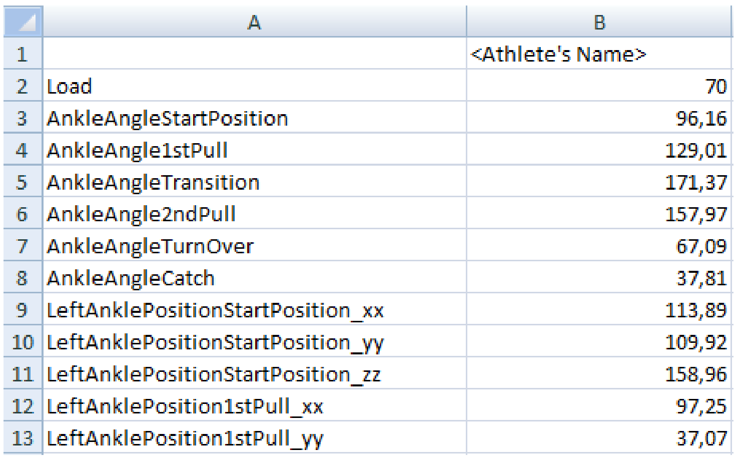
Figure 26 - Sessions Listing Subsystem Flowchart

4.2.8. XLS File Reading Subsystem

The purpose of this subsystem is to read xls files that are created by the external systems that capture the biomechanical data.

These systems doesn´t inserts the biomechanical data into the file on the same order, which is a problem because when our system reads the file can interpret the data incorrectly. The solution is to give to the user a template file where all variables are listed. The user needs to insert these variable names on external system. These systems don't export data in same order, so what our application needs to read the variable name to be able to save the correct data value.

Figure 27 shows training session's data file example.



	A	B
1		<Athlete's Name>
2	Load	70
3	AnkleAngleStartPosition	96,16
4	AnkleAngle1stPull	129,01
5	AnkleAngleTransition	171,37
6	AnkleAngle2ndPull	157,97
7	AnkleAngleTurnOver	67,09
8	AnkleAngleCatch	37,81
9	LeftAnklePositionStartPosition_xx	113,89
10	LeftAnklePositionStartPosition_yy	109,92
11	LeftAnklePositionStartPosition_zz	158,96
12	LeftAnklePosition1stPull_xx	97,25
13	LeftAnklePosition1stPull_yy	37,07

Figure 27 - XLS File Template

Figure 28 shows this subsystem flowchart.

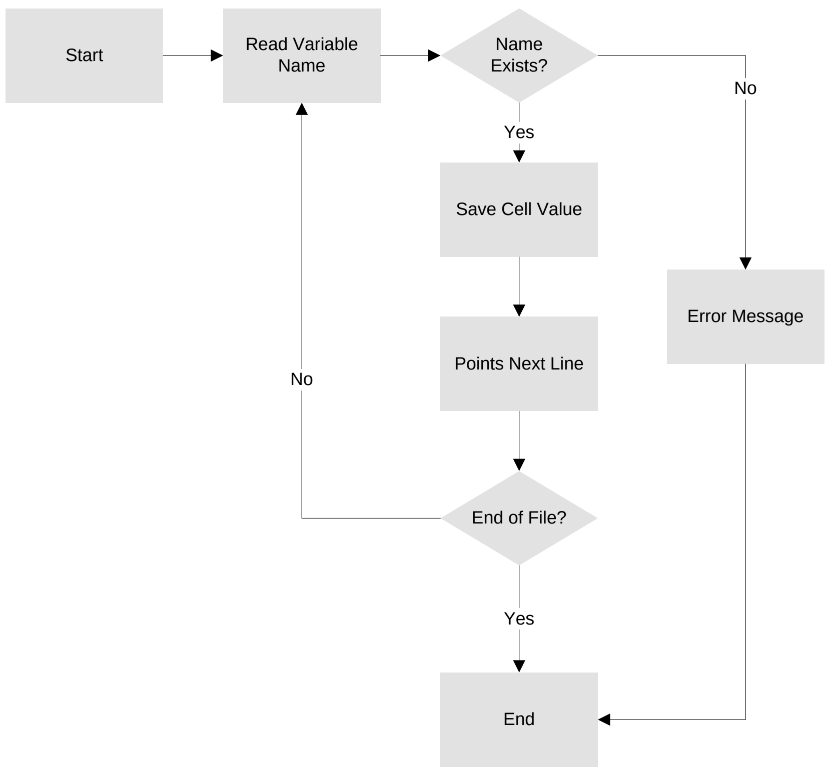


Figure 28 - XLS File Reading Subsystem Flowchart

4.2.9. Database Interface Subsystem

This subsystem is responsible to do the interface between the Java application and the database.

Here the statements are prepared with information to store on the database and queries to select, update, delete and insert data.

Figure 29 represents this subsystem flowchart.

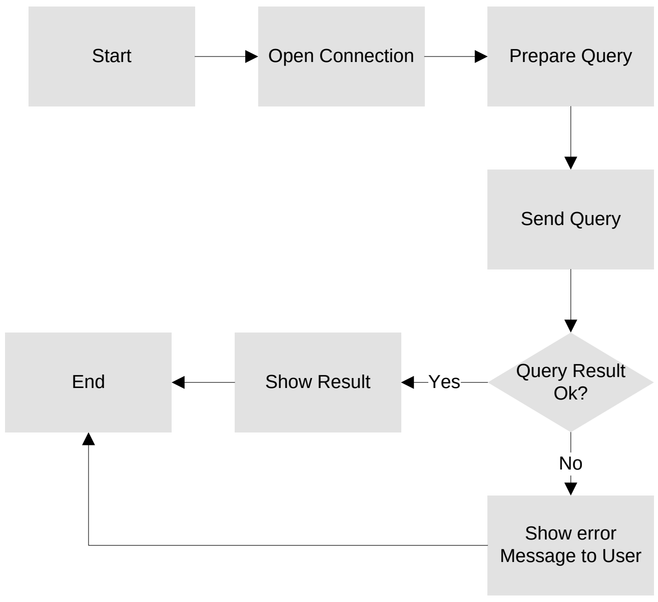


Figure 29 - Database Interface Subsystem

4.2.10. Ontology Interface Subsystem

This subsystem is responsible for interacting with the ontology.

The ontology is loaded to application's memory, it is inserted an individual and stored its properties values. These values correspond to data from the training sessions selected by user. When that is done the system queries the ontology and waits for the results.

When all results are received the systems reports to the user the results in a language that he understands.

Figure 30 represents the flowchart of this system.

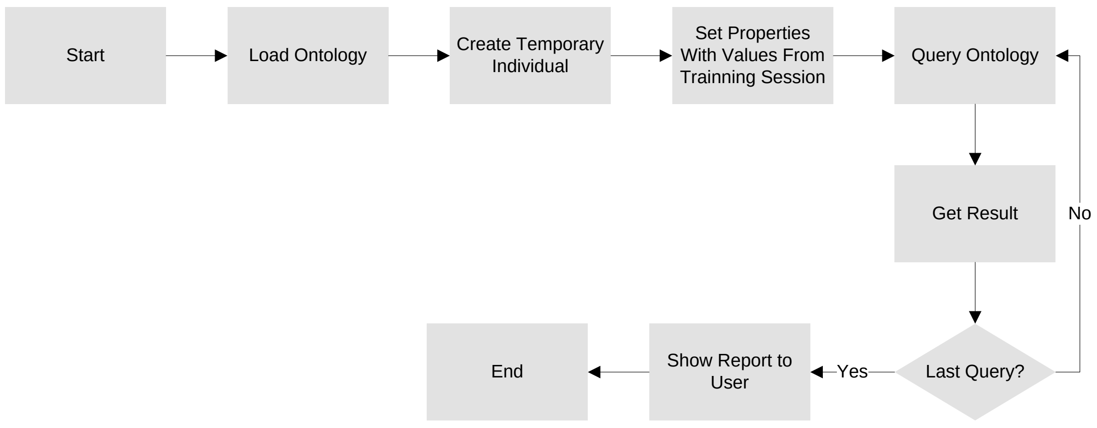


Figure 30 - Ontology Interface Subsystem Flowchart

As was mentioned before, the communication with the ontology is made using SQWRL queries.

It was also mentioned that the queries have this form:

- Comparing two properties values:

```
Athlete(?a) ^ RightKneePositionStartPosition_xx(?a,?x)^
RightAnklePositionStartPosition_xx(?a,?y) ^ swrlb:equal(?x,?y) ->
sqwrl:select(?r)


Athlete(?a) ^ HipPositionStartPosition_yy(?a, ?x) ^
RightKneePositionStartPosition_yy(?a, ?y) ^ swrlb:greaterThan(?x, ?y)->
sqwrl:select(?r)
```

- Comparing properties values with numeric values:

```
Athlete(?a) ^ RightElbowAngleStartPosition(?a, ?x) ^
LeftElbowAngleStartPosition(?a, ?y) ^ swrlb:greaterThan(?x, 175) ^
swrlb:lessThan(?x, 185) ^ swrlb:greaterThan(?y, 175) ^ swrlb:lessThan(?y,185
-> sqwrl:select(?r)
```

- Comparing the result of arithmetic operations between properties values:

```
Athlete(?a) ^ RightAnklePositionStartPosition_xx(?a, ?x) ^
HipPositionStartPosition_xx(?a, ?y) ^ LeftAnklePositionStartPosition_xx(?a,
?w) ^ swrlb:subtract(?z, ?x, ?y) ^ swrlb:subtract(?k, ?y, ?w) ^
swrlb:equal(?z, ?k)-> sqwrl:select(?r)"
```

4.2.11. Query Rule Creation Subsystem

This subsystem is responsible for parsing the xml file and constructing the query rules to send to ontology.

Because we do not do a lot documents manipulation we will use a light parser and create our own handler instead of using a powerful parser with XLST stylesheet.

System reads the element that was collected by the parser and stores its value. In the end it makes a string in a form of SQWRL query correspondent with the rule inserted on XML file.

Figure 31 represents an example of the rule xml file.

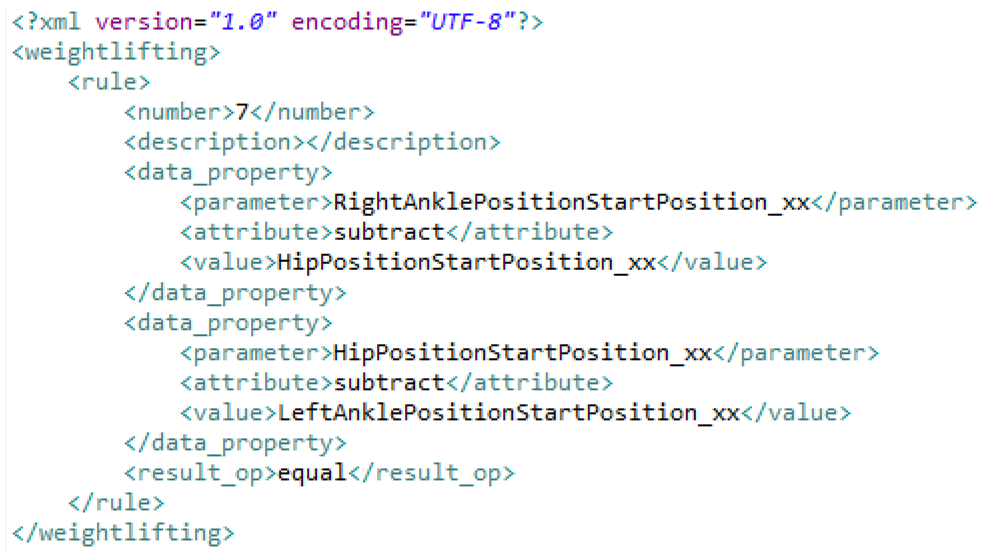

```xml
<?xml version="1.0" encoding="UTF-8"?>
<weightlifting>
    <rule>
        <number>7</number>
        <description></description>
        <data_property>
            <parameter>RightAnklePositionStartPosition_xx</parameter>
            <attribute>subtract</attribute>
            <value>HipPositionStartPosition_xx</value>
        </data_property>
        <data_property>
            <parameter>HipPositionStartPosition_xx</parameter>
            <attribute>subtract</attribute>
            <value>LeftAnklePositionStartPosition_xx</value>
        </data_property>
        <result_op>equal</result_op>
    </rule>
</weightlifting>
```

Figure 31 - Rule XML File Example

The rule of the example above means that on start position, hip must be at same distance of both ankles, which means that the result of the subtraction of the parameter *RightAnklePositionStartPosition_xx* (position of the right ankle on start position on *xx* 3D axis) by the parameter *HipPositionStartPosition_xx* (position of the hip on start position on *xx* 3D axis) and the result of the subtraction of *HipPositionStartPosition_xx by LeftAnklePositionStartPosition_xx* must be equal.

After parsing, the resulting query on SQWRL language must be:

```
Athlete(?a)^RightAnklePositionStartPosition_xx(?a,?x)^HipPositionStart
Position_xx(?a,?y)^LeftAnklePositionStartPosition_xx(?a,?w)^swrlb:subtract(?z
,?x,?y)^swrlb:subtract(?k,?y,?w)^swrlb:equal(?z,?k)^RulesOk_7(?a,?r)-
>sqwrl:select(?r)
```

Figure 32 shows the flowchart of this subsystem.

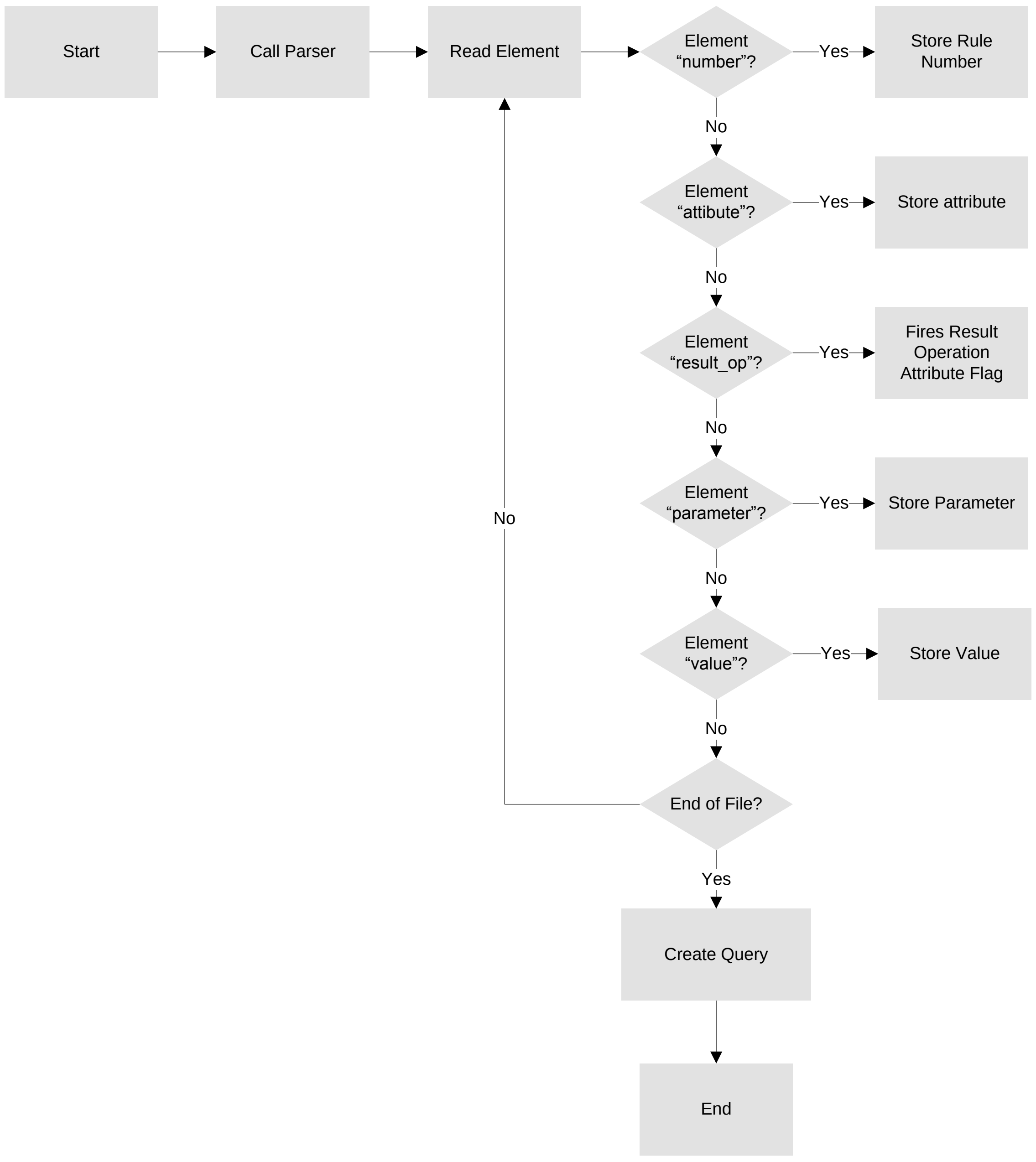


Figure 32 - Query Rules Creation Subsystem Flowchart

There are five functions from the handler class that are called automatically by parser:

- *startDocument()* when application starts parsing the file;
- *startElement()* when parser finds the beginning of an element;
- *characters()*to read each character of the value contained in the element;
- *endElement()* when parser finishes reading the element;
- *endDocument()*when parser finishes reading the document.

The function of the handler is, like the name says, handling with the information received from parser. On this case, handler verifies the syntax and stores the data.

There are three main elements: *property, attribute* and *value*. To deal with these three elements were created three arrays, each one containing information about the elements.

- *Property* array:

class_name	*property_name*	*class_argument*	*property_argument*

The firs element of the array belongs to the property's class. The second element is for the property name. The second and third elements are used to store the property arguments.

- *Attribute* array

attribute_name	*first_argument*	*second_argument*	*third_argument*	*three_argument_flag*	*result*

The first element is to store the attribute's name. If attribute has three arguments (two arguments for input and one for output) the *three_argument_flag* is fired and the three arguments are stored. If the attribute has two arguments, only the *first_argument* and *second_argument* are filled.

If the attribute compares the result of two others the *result* flag is fired and the input arguments are the output arguments of the last two attributes stored.

- *Value* array
  The *value* variable can assume two forms, one if the *value* element is a property and other if it is a number or string. The way of array is filled depends on a flag that exists on the last element of this array.

  - Property:

class_name	property_name	class_argument	property_argument	type_flag

- Numeric value and string:

value				type_flag

4.3. Writing Rules on XML File

A template of rules file was present during the design phase.

During the implementation phase were inserted fifteen rules on file.

Following is written the rules meaning, a parameterization, and the xml code:

1.

   Meaning: Right knee and right ankle should be vertically aligned during the start position;

   Paramerters: RightKneeStartPosition_xx = RightAnkleStartPosition_xx

   XML code:

```
<rule>
      <number>1</number>
      <data_property>
            <parameter>RightKneePositionStartPosition_xx</parameter>
            <attribute>equal</attribute>
            <value>RightAnklePositionStartPosition_xx</value>
      </data_property>
</rule>
```

2.

   Meaning: Left knee and left ankle should be vertically aligned during the start position;

   Simplification: LeftKneeStartPosition_xx = LeftAnkleStartPosition_xx

   XML code:

```
<rule>
      <number>2</number>
      <data_property>
            <parameter>LeftKneePositionStartPosition_xx</parameter>
            <attribute>equal</attribute>
            <value>LefttAnklePositionStartPosition_xx</value>
      </data_property>
</rule>
```

3.

Meaning: The position of hip should be higher than the position of knees;

Simplification: HipPositionStartPosition_yy >= RightKneePositionStartPosition_yy

XML code:

```
<rule>
      <number>3</number>
      <data_property>
            <parameter>HipPositionStartPosition_yy</parameter>
            <attribute>greaterThan</attribute>
            <value>RightKneePositionStartPosition_yy</value>
      </data_property>
</rule>
```

4.

Meaning: At start position shoulders should be aligned or slightly ahead of the barbell;

Simplification: RightShoulderPosition_zz >= BarPositionStartPosition_zz

XML code:

```
<rule>
      <number>4</number>
      <data_property>
          <parameter>RightShoulderPositionStartPosition_zz</parameter>
          <attribute>greaterThanOrEqual</attribute>
          <value>BarPositionStartPosition_zz</value>
      </data_property>
</rule>
```

5.

Meaning: At start position, both elbows should have an angle between 175º and 185º;

Simplification: RightElbowAngleStartPosition > 175 && RightElbowAngleStartPosition < 185 && LefttElbowAngleStartPosition > 175 && LeftElbowAngleStartPosition < 185

XML code:

```
<rule>
      <number>5</number>
      <data_property>
            <parameter>RightElbowAngleStartPosition</parameter>
            <attribute>greaterThan</attribute>
            <value>175</value>
      </data_property>
      <data_property>
            <parameter>RightElbowAngleStartPosition</parameter>
            <attribute>lessThan</attribute>
            <value>185</value>
      </data_property>
      <data_property>
            <parameter>LeftElbowAngleStartPosition</parameter>
            <attribute>greaterThan</attribute>
            <value>175</value>
      </data_property>
      <data_property>
            <parameter>LeftElbowAngleStartPosition</parameter>
            <attribute>lessThan</attribute>
            <value>185</value>
      </data_property>
</rule>
```

6.

Meaning: The back should be slightly curved at start position;

Simplification: BackAngleStartPosition ≠ 0

XML code:

```
<rule>
      <number>6</number>
      <data_property>
            <parameter>BackAngleStartPosition</parameter>
            <attribute>notEqual</attribute>
            <value>0</value>
      </data_property>
</rule>
```

7.

Meaning: On start position, hip should be at same distance of right and left ankles;

Simplification: (RightAnkleStartPosition_xx – HipPositionStartPosition_xx) = (HipPositionStartPosition_xx - LeftAnkleStartPosition_xx)

XML code:

```
<rule>
      <number>7</number>
      <data_property>
            <parameter>RightAnklePositionStartPosition_xx</parameter>
            <attribute>subtract</attribute>
            <value>HipPositionStartPosition_xx</value>
      </data_property>
      <data_property>
            <parameter>HipPositionStartPosition_xx</parameter>
            <attribute>subtract</attribute>
            <value>LeftAnklePositionStartPosition_xx</value>
      </data_property>
      <result_op>
            <attribute>equal</attribute>
      </result_op>
</rule>
```

8.

Meaning: Shoulders have to be higher than hip on start position;

Simplification: RightSoulderPositionStartPosition_yy > HipPositionStartPosition_yy

XML code:

```
<rule>
      <number>8</number>
      <data_property>
          <parameter>RightShoulderPositionStartPosition_yy</parameter>
          <attribute>greaterThan</attribute>
          <value>HipPositionStartPosition_yy</value>
      </data_property>
</rule>
```

9.

Meaning: During the first pull bar should be higher than knees;

Simplification: BarPosition1stPull_yy > RightKneePosition1stPul_yy

XML code:

```
<rule>
      <number>9</number>
      <data_property>
            <parameter>BarPosition1stPull_yy</parameter>
            <attribute>greaterThan</attribute>
            <value>RightKneePosition1stPull_yy</value>
      </data_property>
</rule>
```

10.

Meaning: During the first pull shoulders and hip should elevate at same rate;

Siplification: (RightShoulderPosition1stPull_yy – HipPosition1stPull_yy) = (RightShoulderPositionStartPosition_yy - HipPositionStartPosition_yy)

XML code:

```
<rule>
      <number>10</number>
      <data_property>
            <parameter>RightShoulderPosition1stPull_yy</parameter>
            <attribute>subtract</attribute>
            <value>HipPosition1stPull_yy</value>
      </data_property>
      <data_property>
          <parameter>RightShoulderPositionStartPosition_yy</parameter>
          <value>HipPositionStartPosition_yy</value>
          <attribute>subtract</attribute>
      </data_property>
      <result_op>
            <attribute>equal</attribute>
      </result_op>
</rule>
```

11.

Meaning: The angle of the back on first pull should be the same than on start position;

Simplification: BackAngleStartPosition = BackAngle1stPull

XML code:

```
<rule>
      <number>11</number>
      <data_property>
            <parameter>BackAngleStartPosition</parameter>
            <attribute>equal</attribute>
            <value>BackAngle1stPull</value>
      </data_property>
</rule>
```

12.

Meaning: During first pull the angle of elbows should be between 175º and 185º;

Simplification: RightElbowAngle1stPull > 175 && RightElbowAngle1stPull < 185 && LefttElbowAngle1stPull > 175 && LeftElbowAngle1stPull < 185

XML code:

```
<rule>
      <number>12</number>
      <data_property>
            <parameter>RightElbowAngle1stPull</parameter>
            <attribute>greaterThan</attribute>
            <value>175</value>
      </data_property>
      <data_property>
            <parameter>RightElbowAngle1stPull</parameter>
            <attribute>lessThan</attribute>
            <value>185</value>
      </data_property>
      <data_property>
            <parameter>LeftElbowAngle1stPull</parameter>
            <attribute>greaterThan</attribute>
            <value>175</value>
      </data_property>
      <data_property>
            <parameter>LeftElbowAngle1stPull</parameter>
            <attribute>lessThan</attribute>
            <value>185</value>
      </data_property>
</rule>
```

13.

Meaning: During the first pull bar should go backward;

Simplification: BarPosition1stPull_zz < BarPositionStartPosition_zz

XML code:

```
<rule>
      <number>13</number>
      <data_property>
            <parameter>BarPosition1stPull_zz</parameter>
            <attribute>lessThan</attribute>
            <value>BarPositionStartPosition_zz</value>
      </data_property>
</rule>
```

14.

Meaning: During the first pull knees should go backward;

Simplification: (RightKneePosition1stPull_zz < RightKneePositionStartPosition_zz) && (LeftKneePosition1stPull_zz < LeftKneePositionStartPosition_zz)

XML code:

```
<rule>
      <number>14</number>
      <data_property>
            <parameter>RightKneePosition1stPull_zz</parameter>
            <attribute>lessThan</attribute>
            <value>RightKneePositionStartPosition_zz</value>
      </data_property>
      <data_property>
            <parameter>LeftKneePosition1stPull_zz</parameter>
            <attribute>lessThan</attribute>
            <value>LeftKneePositionStartPosition_zz</value>
      </data_property>
</rule>
```

15.

Meaning: On first pull shoulders position should be aligned or slightly ahead than bar position.

Simplification: RightShoulderPosition1stPull_zz>= HipPosition1stPull_zz

XML code:

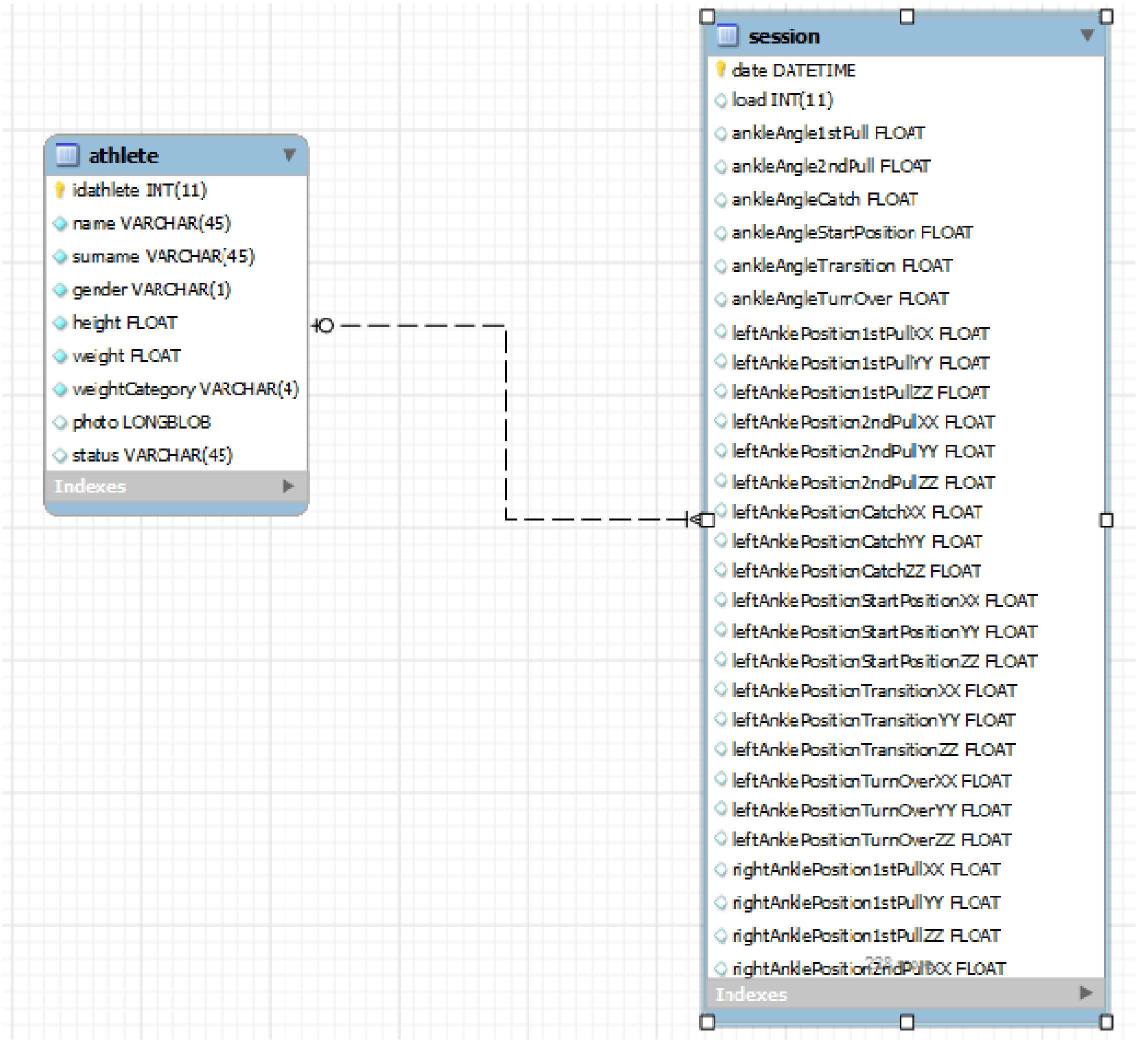
```
<rule>
      <number>15</number>
      <data_property>
            <parameter>RightShoulderPosition1stPull_zz</parameter>
```

```
                <attribute>greaterThanOrEqual</attribute>
                <value>BarPosition1stPull_zz</value>
            </data_property>
        </rule>
```

4.4. Database Design

Database will be created using MySQL. It is composed by two tables, one containing athlete's information, with the athlete's id as its primary key, and another with the data related with the training sessions, which has the session date as its primary key and the athlete id as its foreign key.

Figure 33 shows the design of the database.



athlete
idathlete INT(11)
name VARCHAR(45)
surname VARCHAR(45)
gender VARCHAR(1)
height FLOAT
weight FLOAT
weightCategory VARCHAR(4)
photo LONGBLOB
status VARCHAR(45)
Indexes
session
date DATETIME
load INT(11)
ankleAngle1stPull FLOAT
ankleAngle2ndPull FLOAT
ankleAngleCatch FLOAT
ankleAngleStartPosition FLOAT
ankleAngleTransition FLOAT
ankleAngleTurnOver FLOAT
leftAnklePosition1stPullXX FLOAT
leftAnklePosition1stPullYY FLOAT
leftAnklePosition1stPullZZ FLOAT
leftAnklePosition2ndPullXX FLOAT
leftAnklePosition2ndPullYY FLOAT
leftAnklePosition2ndPullZZ FLOAT
leftAnklePositionCatchXX FLOAT
leftAnklePositionCatchYY FLOAT
leftAnklePositionCatchZZ FLOAT
leftAnklePositionStartPositionXX FLOAT
leftAnklePositionStartPositionYY FLOAT
leftAnklePositionStartPositionZZ FLOAT
leftAnklePositionTransitionXX FLOAT
leftAnklePositionTransitionYY FLOAT
leftAnklePositionTransitionZZ FLOAT
leftAnklePositionTurnOverXX FLOAT
leftAnklePositionTurnOverYY FLOAT
leftAnklePositionTurnOverZZ FLOAT
rightAnklePosition1stPullXX FLOAT
rightAnklePosition1stPullYY FLOAT
rightAnklePosition1stPullZZ FLOAT
rightAnklePosition2ndPullXX FLOAT
Indexes

Figure 33 - Database Design

4.4.1. Querying Database

As was said before, the application communicates with database using SQL queries.

Each type of query has its own format.

- Insert query:

```
INSERT INTO `biomechanics`.`athlete`(name,gender,height,weight,
weightCategory,photo,status,age) VALUES (?,?,?,?,?,?,?,?)
```

- Update query:

```
"UPDATE `biomechanics`.`athlete` SET `name` =?, gender =?, height =?,
weight =?, weightCategory =?, photo =?, status =?, age =? WHERE idathlete =?"
```

- Select query:

```
"SELECT idathlete, name, gender, height,weight,weightCategory, photo,
status, age FROM `biomechanics`.`athlete` ORDER BY name"
```

- Delete query:

```
"DELETE FROM `biomechanics`.`session` WHERE `date` = '"+date+"' AND
`idAthlete` = "+idAthlete+""
```

5. System Implementation

5.1. Java Application

5.1.1. Main Window Interface

The main window interface is the first window to be presented to user after application starts. Here user is able to check all athletes that were loaded from database.

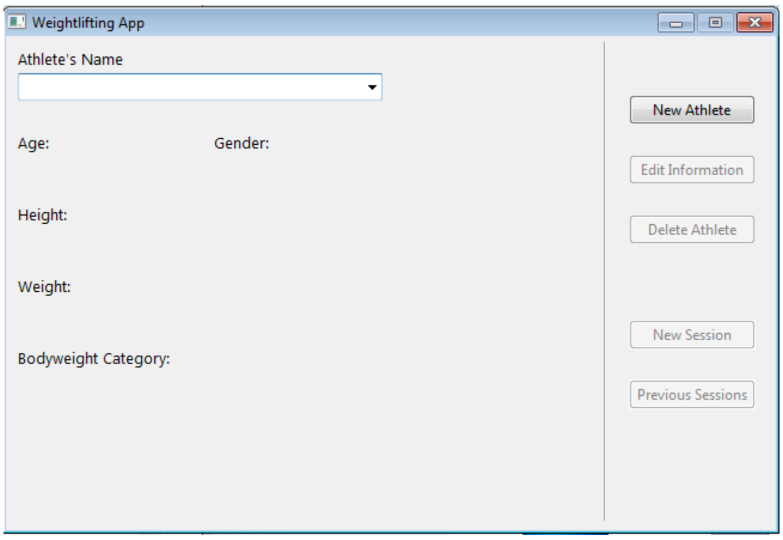
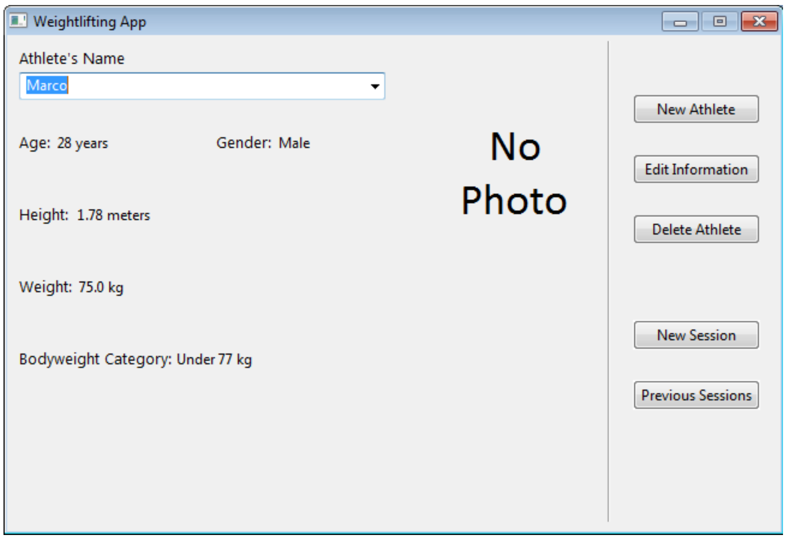


Figure 34 - Main Window Interface

From here user is able to create, update and delete an athlete profile, create a new session and list all session of the selected athlete by clicking on corresponding button. Some buttons are disabled by default.

Clicking on an athlete from the list, user can see his personal information. At this point, the disabled button pass to enable.


Weightlifting App

Athlete's Name: Marco

Age: 28 years

Gender: Male

Height: 1.78 meters

Weight: 75.0 kg

Bodyweight Category: Under 77 kg

No Photo

Figure 35 - Main Window Athlete's Personal Information

5.1.2. New Athlete Window Interface

Here user can insert a new athlete's profile. The name, weight, height, gender and weight category fields are mandatory.

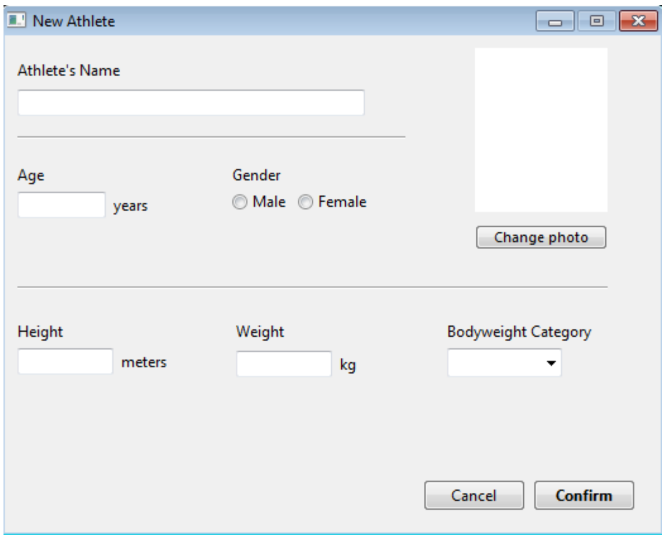
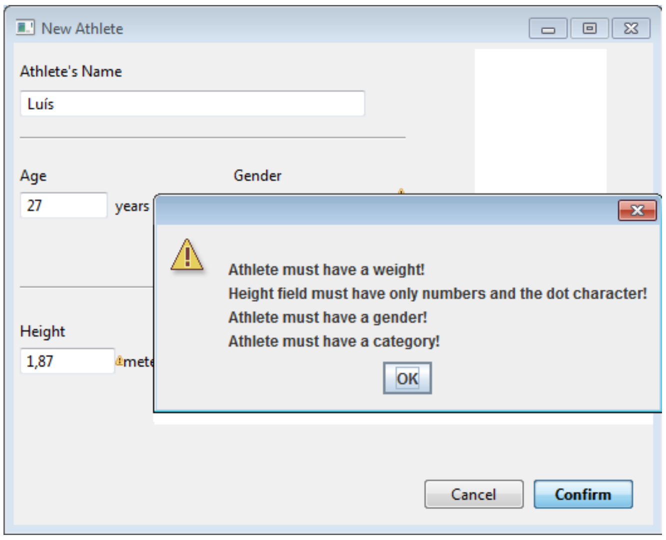


Figure 36 - New Athlete Window Interface

If those fields were left blank or filled on a wrong form an error message will show.



Athlete must have a weight!
Height field must have only numbers and the dot character!
Athlete must have a gender!
Athlete must have a category!

Figure 37 - New Athlete Error Message

If all fields were correctly filled data will be saved on database.

5.1.3. Edit Personal Information Window Interface

On this window user can change any personal information about the athlete. This page is similar to the New Athlete page and contains the same error controls.

5.1.4. New Session Window Interface

On New Session window interface, user can create a new session by inserting the training session data file, the session data and time and some comments.

When user confirms the creation of a new session, is asked to user if he wants to save the data on database. In positive case, the Database Interface subsystem is called to store data on database. Following is asked if user wants to system runs the ontology inferring rules with new data.

Figure 38 shows the interface of this subsystem.

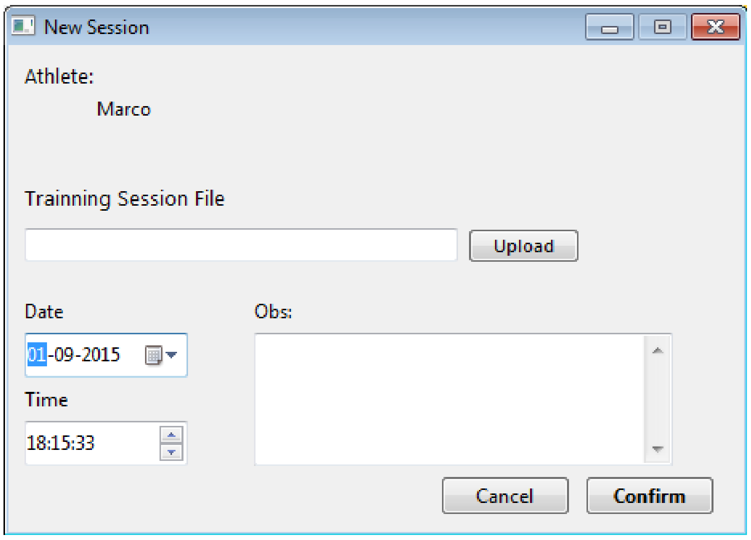


Figure 38 - New Session Window Interface

5.1.5. Athlete's Session List Window Interface

User is able to get a list of all training sessions of an athlete using this window. From this interface, user can add a new session or delete an existing one. After selecting a session, user is able to query the ontology with the data from that session.

Figure 39 shows the interface of this window.

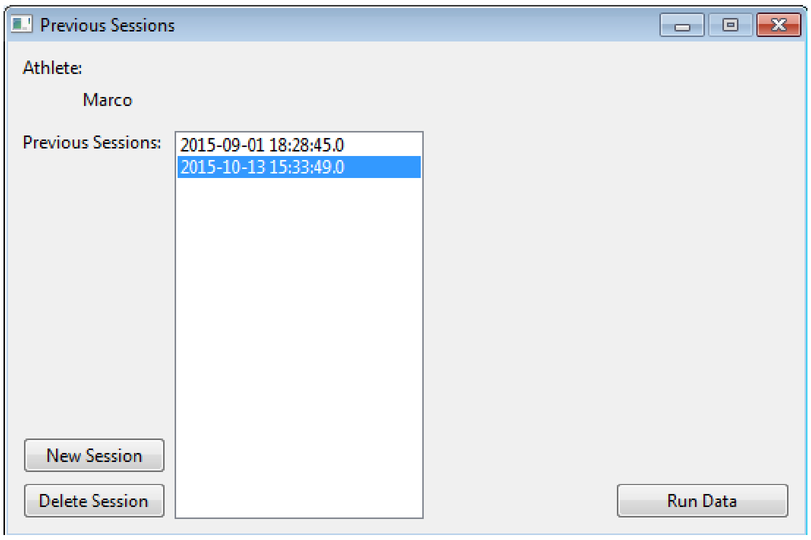


Figure 39 - Training Sessions List window Interface

5.1.6. Results Page Interface

Results page gives to user the results of inferred rules. From the results, athlete knows if he accomplished the rule's requirements.

5.1.7. Ontology Interface

After preparing queries, as was mentioned on design phase, it is necessary to load the ontology in order be able to infer rules. The API's used were provided by *Protégé* development team.

```
OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
SQWRLQueryEngine queryEngine = P3SQWRLQueryEngineFactory.create(owlModel);
```

When ontology is fully loaded we just need to send the queries and handle the results.

```
        Result  resultb=(Result)queryEngine.runSQWRLQuery  ("queryRule"+i+1+"",
query);
```

The application receives boolean values from the ontology. If the inferred rule were satisfied the application receives the *true* value, otherwise the *false* value will be received.

After application received all results the result page will be called to present them to user.

5.1.8. Creating Database Queries

Before the execution of the query, some steps need to be done.

- First we need to initialize the driver that will serve as interface to database, its *url*, credentials, connection and statement. Database was created using MySQL software, so the API's used on application are specifically for databases created on that framework.

```
static final String driver = "com.mysql.jdbc.Driver";
static final String url = "jdbc:mysql://localhost/biomechanics";
static final String user = "root";
static final String pwd = "root";
Statement smt = null;
```

- Then we have to open a connection to database and create a statement on the connection:

  ```
  conn= DriverManager.getConnection(url,user,pwd);
  smt= conn.createStatement();
  ```

- Next we prepare and execute the query and handle with the results:

  ```
  String sql= "SELECT idathlete, name, gender, height, weight,weightCategory, photo, status, age FROM `biomechanics`.`athlete` ORDER BY name";
  ResultSet rs = smt.executeQuery(sql);
  ```

- Finally, we handle with the results.

  ```
  while(rs.next())
  {…}
  ```

5.1.9. Rules Generation

The rules generation is made in three steps. First application does the parsing of the xml rules file. The API used to parse the file is the SAXPARSER API. This API has low resources consumption and is fast enough to our needs.

The first thing to do is to declare and instantiate the parser and the handler classes. The handler class receives the values provided by the parser and stores them to build the rule later.

```
XMLReader xr = XMLReaderFactory.createXMLReader();
XmlHandler handler = new XmlHandler();
xr.setContentHandler(handler);
xr.setErrorHandler(handler);
```

```
File rules = new
File("c:\\javaworkspace\\WeightLifting\\src\\weightlifting\\rules.xml"
);
InputStream inputStream= new FileInputStream(rules);
Reader reader = new InputStreamReader(inputStream,"UTF-8");
InputSource is = new InputSource(reader);
xr.parse(is);
```

When the function *parse()* from the class *XMLReader* is called the application starts parsing. This function is provided by the SAX API. The aspect of the rules file was described during the design phase.

When rule file is completely written and variables store a function to build the rule on SQWRL language is called.

This function receives all variables and put them on a specific order to build the rule correctly. For this demonstration we will implement a rule that verifies if right ankle and right knee are vertically aligned on start position.

First the function adds the class and its argument:

```
While(parameters[i][0]!=null{
        rule=rule +parameters[i][0]+"("+parameters[i][2]+") ^ ";
        i++;
}
```

Where ***parameters[i][0]*** represents the class name, for example “Athlete”, and ***parameters[i][2]*** represents the class argument “?a”. The rule can have more than one class, so the *while* cycle is needed.

The rule built until now is:

```
Athlete(?a) ^
```

After that the program adds the parameters:

```
rule=rule+parameters[i][1]+"("+parameters[i][2]+","+parameters[i][3] +") ^ ";
```

`parameters[i][1]` represents the property name, like *RightAnklePositionStartPosition_xx*, while `parameters[i][3]` represents its argument.

So, the rule built with these new variables should be:

```
Athlete(?a) ^ RightAnklePositionStartPosition_xx(?a,?f) ^
```

After this step, program checks if *value* variable has properties. On positive case, program adds it to the rule.

```
rule= rule+values[i][1]+"("+values[i][2]+","+values[i][3]+") ^ ";
```

On this case the *value* array has the same form than *parameter* array, so the form of reading is the same too.

The rule described should now have this format:

```
Athlete(?a)        ^        RightAnklePositionStartPosition_xx(?a,?f)        ^
RightKneePositionStartPosition(?a,?g) ^
```

After program inserts all parameters on rule it inserts the attributes.

Attributes can have two or three arguments, so we have to threat attributes depending of the number of arguments it has. As was referenced above, there is a flag, present on *attributes* array, that is fired in case of the argument has three arguments.

```
if(attributes[i][4]=="y")
{
	switch(values[i][4])
	{
		case "p":
			rule=rule+attributes[i][0]+"("+attributes[i][3]+","+
			parameters[i][3]+","+values[i][3]+") ^ ";
			break;

		case "n":
			rule= rule+attributes[i][0]+"("+ attributes[i][3]+","+
			parameters[i][3]+","+Float.parseFloat(values[i][0])+")^";
			break;

		case "s":
			rule= rule+attributes[i][0]+"("+attributes[i][3]+","+
			parameters[i][3]+",\""+values[i][0]+"\") ^ ";
			break;

		default:
			break;
	}
}
```

If attribute receives three arguments, the first one is the operation result argument, the second is the properties argument and the third is the *value* argument that can be a property argument, a number or a string.

In case of attribute receiving only two arguments there is no need to check the *three_argument_flag* value. The result operation argument is not inserted in this case.

```
switch(values[i][4])
{
	case "p":
		rule=rule+attributes[i][0]+"("+parameters[i][3]+","+
		values[i][3]+") ^ ";
		break;

	case "n":
		rule= rule+attributes[i][0]+"("+parameters[i][3]+","+
		Float.parseFloat(values[i][0])+") ^ ";
		break;

	case "s":
		rule= rule+attributes[i][0]+"("+parameters[i][3]+",\""+
		values[i][0]+"\") ^ ";
		break;

	default:
		break;
}
```

We want to verify is right ankle and right knee are aligned which means that their position value on *xx* 3D axis must be the same. The attribute that does that verification is the attribute *equal*. This attribute receives the property argument and the *value* argument that is a property argument as well.

So, the resulting code should be like the one described below.

```
Athlete(?a)        ^        RightAnklePositionStartPosition_xx(?a,?f)        ^
RightKneePositionStartPosition(?a,?g) ^ swrlb:equal(?f,?g)
```

After this the rule only needs an output. This output can only returns a value of variable present in the input part. We want the output return a Boolean value, so we have to add a new variable that contains boolean values. For that we insert the *RulesOk* variable. This variable has the *true* value by default. If the attribute's verification is satisfied the output returns the *RulesOk* value, otherwise it returns nothing meaning the rule was not satisfied. Now the program can finish building the rule.

```
rule= rule+"RulesOk_"+ruleNumber+"(?a,?z) -> sqwrl:select(?z)"
```

The resulting rule should now be like:

```
Athlete(?a) ^ RightAnklePositionStartPosition_xx(?a,?f) ^ RightKneePositionStartPosition(?a,?g) ^ swrlb:equal(?f,?g) ^ RulesOk_1(?a,?z) -> sqwrl:select(?z)
```

5.1.10. Training Sessions Data Reading

To start reading file we have to point to a reference cell. In this case will be cell *A1*.

```
CellReference cellRef1 = new CellReference ("A1");
```

Then, column and row variables are initialized.

```
Row row1 = sheet.getRow(cellRef1.getRow());
Iterator<Row> rowIterator1 = sheet.iterator();
int col1 = cellRef1.getCol();
```

After this application start reading the file and storing variables values.

```
while (rowIterator1.hasNext()==!errorFlag)
{
    row1 = rowIterator1.next();
    cell1 = row1.getCell(col1);
    cell2 = row1.getCell(col1+1);

    if (cell1!=null) {
        switch(cell1.getStringCellValue()) {
            case "Load":
                if(cell2!=null) {
                    session.setLoad((int)cell2.getNumericCellValue());
                }
            break;
```

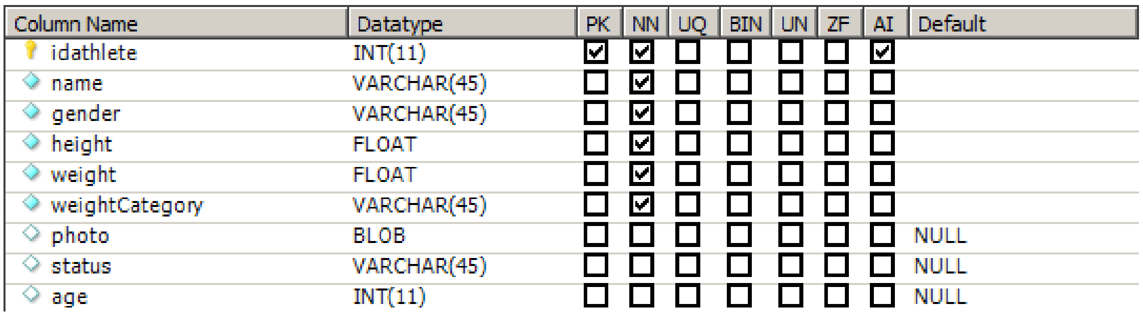
```
case "AnkleAngleStartPosition":
     if(cell2!=null) {
           session.setAnkleAngle(0,
           (float)cell2.getNumericCellValue());
    }
 break;
```

session is an instance of the class that contains information about training sessions data.

5.2. Database Implementation

Database is composed by two tables. One called *athlete* that is used to store athlete's personal information and has *idathlete* as its primary key.

Figure 40 represents shows all variables present on *athlete* table.

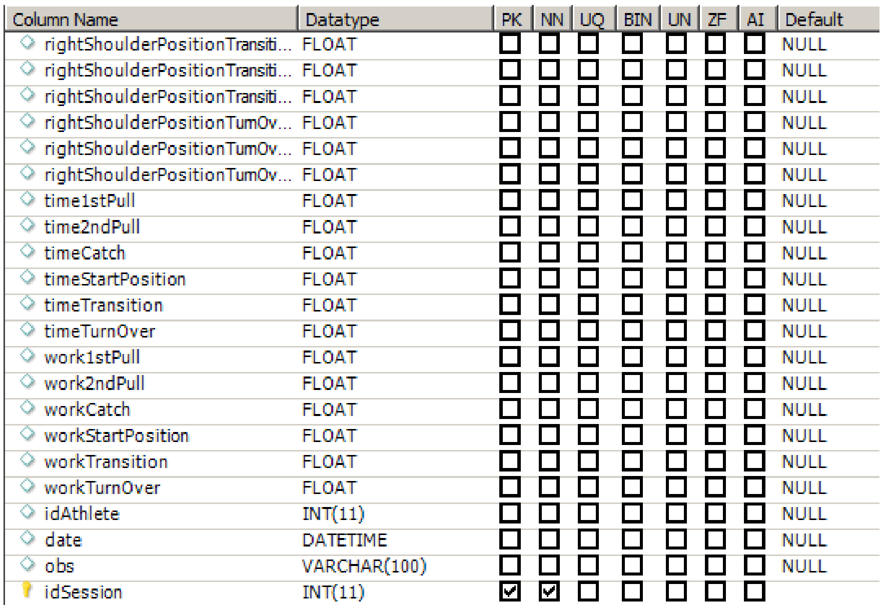


Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
idathlete	INT(11)	☑	☑	☐	☐	☐	☐	☑	
name	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
gender	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
height	FLOAT	☐	☑	☐	☐	☐	☐	☐	
weight	FLOAT	☐	☑	☐	☐	☐	☐	☐	
weightCategory	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
photo	BLOB	☐	☐	☐	☐	☐	☐	☐	NULL
status	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL
age	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL

Figure 40 - *athlete* table

The other table that is present on database stores training sessions data of all athletes. It has *idSession* as its primary key and *idAthlete* as foreign key that refers to *idathlete* on *athlete* table.

Figure 41 shows *session* table.



Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
rightShoulderPositionTransiti...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
rightShoulderPositionTransiti...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
rightShoulderPositionTransiti...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
rightShoulderPositionTurnOv...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
rightShoulderPositionTurnOv...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
rightShoulderPositionTurnOv...	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
time1stPull	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
time2ndPull	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
timeCatch	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
timeStartPosition	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
timeTransition	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
timeTurnOver	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
work1stPull	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
work2ndPull	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
workCatch	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
workStartPosition	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
workTransition	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
workTurnOver	FLOAT	☐	☐	☐	☐	☐	☐	☐	NULL
idAthlete	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL
date	DATETIME	☐	☐	☐	☐	☐	☐	☐	NULL
obs	VARCHAR(100)	☐	☐	☐	☐	☐	☐	☐	NULL
idSession	INT(11)	☑	☑	☐	☐	☐	☐	☐	

Figure 41 - *session* table

6. Results

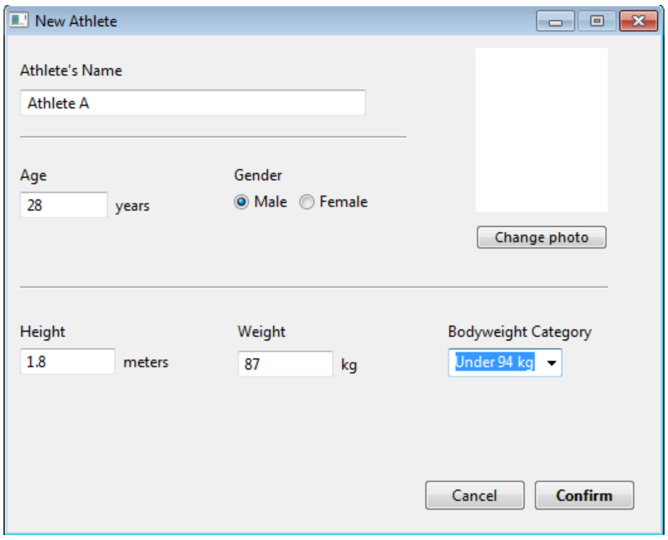
In this chapter we will test the system and comment the results obtained.

6.1. Inserting a New Athlete

We will start by creating an athlete profile on application.

We will create an athlete named *AthleteA* with 28 year of age, male, 1,80 meters of height and 87 kg of weight.

Figure 42 presents the insertion of the athlete profile.



New Athlete

Athlete's Name: Athlete A

Age: 28 years

Gender: [✓ Male] Female

Height: 1.8 meters

Weight: 87 kg

Bodyweight Category: Under 94 kg

Figure 42 - Athlete Profile Insertion

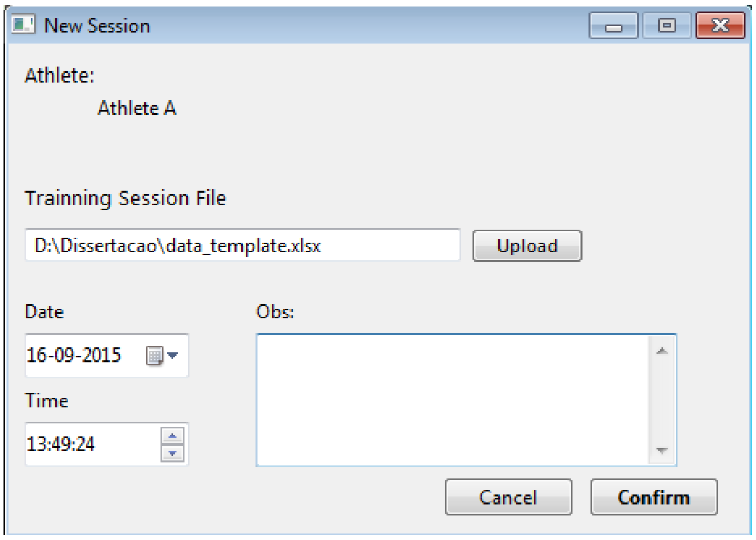
After *Confirm* button is clicked this athlete profile will be stored on database.

6.2. Inserting a Training Session Data File

After the creation of an athlete profile is possible to add files with data from training sessions. For that, on main window we choose the corresponding athlete and next we click on *New Session* button to show the window for the insertion of the file.

For this demonstration we will insert data from training session of a novice athlete.

Figure 43 shows the insertion of the file. It is possible to add the session data and some observations.



Athlete:
Athlete A

Trainning Session File
D:\Dissertacao\data_template.xlsx

Date
16-09-2015

Time
13:49:24

Obs:

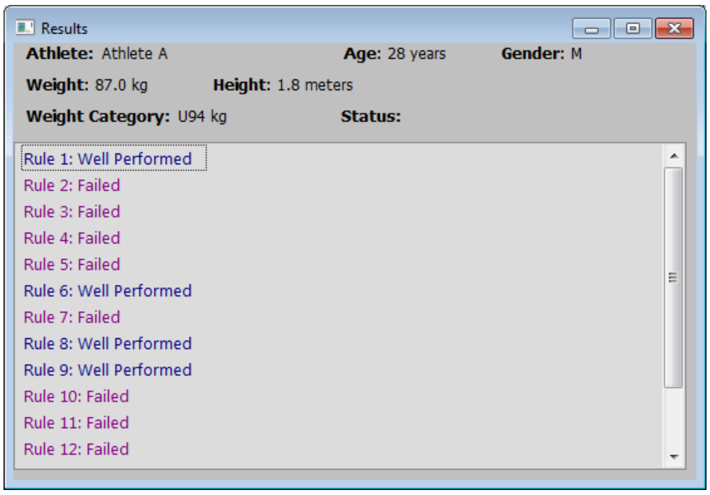
Figure 43 - Training Session File Insertion

When *Confirm* button is clicked application asks to user if he wants to store the data on database or run on ontology. If user clicks on button to run data, application starts inferring rules on ontology.

6.3. Inferring Rules on Ontology

When user runs session's data, application starts inferring rules on ontology. As was mentioned before, these rules are sent to ontology in a form of queries.

When all rules are inferred and the application received all results, a window will present to user showing the result of each inferred rule indicating if the part of the exercise that each rule refers was successfully performed.



Results

Athlete: Athlete A **Age:** 28 years **Gender:** M

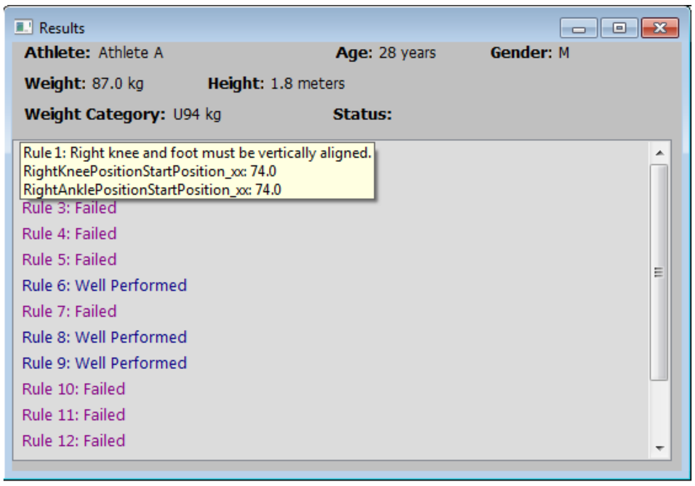
Weight: 87.0 kg **Height:** 1.8 meters

Weight Category: U94 kg **Status:**

Rule 1: Well Performed
Rule 2: Failed
Rule 3: Failed
Rule 4: Failed
Rule 5: Failed
Rule 6: Well Performed
Rule 7: Failed
Rule 8: Well Performed
Rule 9: Well Performed
Rule 10: Failed
Rule 11: Failed
Rule 12: Failed

Figure 44 - Results from Inferred Rules

Moving the mouse over a rule it is possible to see a description of that rule as well as the values of the parameters related with the rule.



Results

Athlete: Athlete A **Age:** 28 years **Gender:** M

Weight: 87.0 kg **Height:** 1.8 meters

Weight Category: U94 kg **Status:**

Rule 1: Right knee and foot must be vertically aligned.
RightKneePositionStartPosition_xx: 74.0
RightAnklePositionStartPosition_xx: 74.0

Rule 3: Failed
Rule 4: Failed
Rule 5: Failed
Rule 6: Well Performed
Rule 7: Failed
Rule 8: Well Performed
Rule 9: Well Performed
Rule 10: Failed
Rule 11: Failed
Rule 12: Failed

Figure 45 – Rule Details

6.4. Results Observation

The obtained results demonstrate the system meets the requirements. Application parses both *xml* and *xls* files successfully and the communication with database and ontology is made without errors. The rules details allow user to understand the results better and make some corrections on athlete's posture.

One of the reasons of the majority of rules has been given as failed could be the accuracy of data due to the lack of accuracy of collecting data systems. In some cases athlete's body parts should be equal positioned but they are mismatched by a small distance that is captured by the collecting data system. Inferring rules with that data, ontology notices the data is not equal and returns nothing, giving the rule as not satisfied, meaning the part of the exercise referenced by that rule was not executed successfully.

One solution for that problem should be the creation of a threshold, improving the system performance, nevertheless harming the correct execution of the exercise.

Conclusion

In this work was concluded the use of ontology is a viable way to approach creating systems that deal with knowledge-based concepts. However, comparing with databases, the waiting time for a result from a query is higher. Each query can take up to two minutes to return a result.

Setting rules out of the core system allows user to add, remove or edit rules anytime, adapting the system to the athlete and therefore increasing system's performance and gives to user freedom from the programmer. Although the purpose of this system is the weightlifting it can be used for other sports.

One constraint of system's performance is the data collection system accuracy. If a system with low accuracy is used, the collected data will also not be accurate which low our systems performance and consequently can induce athletes in error.

System's performance also depends of the number of rules and its complexity. The greater number of rules and simpler the greater will be system's performance.

Some improvements that can be done on Java application in the future are the addition of more parameters for data collection and more rules. The addition of a rule description and some useful information on results window can be done hereafter.

On ontology, the addition of *Snatch* and *Clean and Jerk's* subparts as subclasses is an implementation to do on the future.

Concluding, this project was a successful experience on knowledge-base systems area and weightlifting.

References

[1] J. D. Fortin and F. J. Falco, "The Biomechanical Principles of Preventing Weighlifting Injuries," in *Phisical Medicine and Rehabilitation: State of Art Reviews*, Philadelphia, Hanley & Belfus, Inc, 1997.

[2] A. Storey and H. K. Smith, "Unique Aspects of Competitive Weightlifting," Springer International Publishing AG, 2012.

[3] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe, Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools, Manchester: The University Of Manchester, 2004.

[4] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and Rule ML," 21 May 2014. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521. [Accessed 18 February 2015].

[5] M. O'Connor and A. Das, "SQWRL: a Query Langauge for OWL," Stanford Center for Biomedical Informatics Research.

[6] M. H.Stone, K. C. Pierce, W. A.Sands and M. E.Stone, "Weightlifting: A Brief Overview," *Strength and Conditioning Journal,* vol. 28, pp. 50-66, 2006.

[7] N. F. Noy and D. L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Stanford University, Stanford, CA, 94305.

[8] V. Oliveira, "Based Personalized Dietary Recommendation for Weightlifting," University of Minho.

[9] Protégé, "Protege3DevDocs - ProtegeWiki," 29 August 2014. [Online]. Available: http://protegewiki.stanford.edu/wiki/Protege3DevDocs. [Accessed 15 December 2014].

[10] R. Pandey and D. Dwivedi, "Ontology Description using OWL to Support Semantic Web Applications," *International Journal of Computer Applications* , vol. 14, 2011.

[11] A. Drechsler, The Weightlifting Encyclopedia: A Guide to World Class Performance, A is A Communications, First Edition,1998.

[12] J. Garhammer, "Biomechanical Profiles of Olympic Weightlifters," 1985.