University of West Bohemia
Faculty of Applied Sciences

Ontology Development in EEG/ERP Domain

Ing. Petr Ježek

Doctoral Thesis
submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
in specialization Computer Science and Engineering

Supervisor: Prof. Ing. Václav Matoušek , CSc.
Department of Computer Science and Engineering

Pilsen 2012

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Tvorba Ontologie v EEG/ERP Doméně

Ing. Petr Ježek

Disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Ing. Václav Matoušek , CSc.
Katedra informatiky a výpočetní techniky

V Plzni 2012

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji tímto, ze tuto práci jsem vypracoval samostatně, s použitím odborné, literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 20. března 2012 Ing. Petr Ježek

Abstract

Because of difficulties with neuroinformatics data/metadata storage, a new research field dealing with the development of neuroinformatics databases is gradually formed. The data within these databases are supposed to be recognizable by interested researchers. To support the development of these databases the description of specific neuroscience fields is needed. Therefore, the scientific community is intensively working on description of individual neuroscience fields by domain ontologies. These ontologies are expressed by Semantic Web languages. Since the current neuroinformatics software tools are usually based on object-oriented languages and relational databases a need for suitable mapping is emerging. Due to differences in semantic expressivity of common modeling techniques and Semantic Web languages it is necessary to fill these semantic gaps. This work is focused on developing an ontology for EEG/ERP domain that expresses EEG/ERP experiments. The developed ontology is practically implemented together with the EEG/ERP Portal. The goal of the EEG/ERP Portal is to serve as a system for storing, managing and interchanging EEG/ERP experiments. The work particularly solves semantic gaps between object-oriented code and Semantic Web languages by adding a missing semantics into the object-oriented code. The developed mapping is implemented within the presented Semantic Framework. The integration of the Semantic Framework within the EEG/ERP Portal ensures transformation of stored experiments into the ontology representation. The registration of the EEG/ERP Portal within the Neuroscience Information Framework practically validates the presented approach.

Abstrakt

Při ukládání neuroinformatických dat/metadat vzniká celá řada problémů související s vhodným popisem ukládaných záznamů. Z toho důvodu vzniká nový vědní obor zabývající se vývojem neuroinformatických databází. Data ukládaná v neuroinformatických databázích jsou určena vědecké komunitě, která věnuje úsilí vývoji databází společně s popisem jednotlivých neurovědeckých oblastí doménovými ontologiemi. Doménové ontologie jsou vyjádřeny jazyky tzv. Sémantického Webu. Současné systémy používané v neuroinformatice jsou obvykle vyvinuté s využitím objektově-orientovaných jazyků a relačních databází, proto je potřeba navrhnout vhodné mapování do jazyků Sémantického Webu. Tato práce si klade za cíl navrhnout ontologii, která popíše doménu EEG/ERP experimentů. Další část práce se věnuje řešení sémantických rozdílů mezi objektově-orientovaným kódem a jazyky Sémantického Webu a navrhuje způsob rozšíření objektově-orientovaného kódu o chybějící sémantiku. Navržené mapování je implementováno v Sémantickém Frameworku. Poslední část práce se zabývá návrhem a vývojem EEG/ERP Portálu. EEG/ERP Portál je systém navržený pro ukládání, sdílení a správu EEG/ERP experimentů. Navržená ontologie je v tomto systému implementována. Integrace Sémantického Frameworku v EEG/ERP Portálu zajišťuje automatizovanou transformaci uložených experimentů do jazyků Sémantického Webu. EEG/ERP Portál je registrován v systému zvaném "Neuroscience Informational Framework", čímž je ověřena správnost navržené transformace.

Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank Václav Matoušek, my thesis supervisor who accepted me for graduate study and gave me an opportunity to work on this very interesting topic.

Next thanks go to my colleague and friend Roman Mouček for his valuable help during my studies.

Last thanks belong to my partner Alena for her moral support and understanding.

Part I

Opening

Chapter 1

Introduction

With the gradual penetration of computer technologies into medicine a brand new scientific discipline is gradually formed. This research field is known as *neuroinformatics* because it combines research in the medicine with research in the computer science.

Neuroinformatics is concerned with the organization of neuroscience data, application of computational models and analytical tools. The usage of computers within the medicine brings an excellent support in execution of time-consuming tasks (e. g. signal processing) or collection of large data sets (obtained during patients examinations or experimental research).

Major challenge for software engineers is to prepare software tools, analytical methods or data standards usable in medical practice. Such software should be based on unified interfaces, open-source technologies and clearly defined protocols in order to knowledge should be distributed among medical laboratories. Despite a lot of tools is being developed the potential benefits that the dissemination of knowledge among laboratories offers are still not used. The main reason is that the medical software is supplied by various mostly commercial vendors who are not interested in developing reusable standards.

On the other hand, the suitable medium for sharing experimental data should be the Internet. Despite the popularity of the Internet, how it is growing, it contains of a huge amount of information with practically no classification. Such not classified data are not suitable for sharing. As a solution an extension called the Semantic Web that is intended to give the data semantic meaning is being developed.

Due to high complexity that relates to processing of medical data there are developed specific databases intended for preservation of medical data. The

sustainability of these databases is very important field in the neuroscience.

1.1 Problem Overview

Our research group specializes in the research of brain activity, especially attention of drivers and seriously injured people is investigated. We widely use the methods of electroencephalography (EEG) and event related potentials (ERP). According to difficulties mentioned in Chapter 1 we will discuss several difficulties related to EEG/ERP experiments.

EEG/ERP experiments usually take a long time and produce a lot of data. With the increasing number of experiments we had to solve their long-term storage and management.

There is no widely spread and generally used standard for EEG/ERP data files within the community. Data formats are usually closed and bounded with specific hardware supplied by the specific vendor.

Results (interpretations) of EEG/ERP experiments are usually more important than obtained data. Some researchers even declare that experimental data have a low value when they are interpreted.

There is no reasonable and easily extensible tool for long-term EEG/ERP experiments storage and management. The general practice is to organize data and metadata in common file directories.

There is no general habit to share and interchange experiments between EEG/ERP laboratories. Some researchers suppose that EEG/ERP experiments are secret or unimportant to share them.

Several organizations that aims is to participate on solving difficulties with sustainability of neuroscience databases were formed. Probably the most important is International Neuroinformatics Coordinating Facility (INCF). INCF develops an infrastructure that serves to interested researchers in developing of a partial infrastructure of each specific neuroscience domain.

The system that facilitates storing, interchanging and managing EEG/ERP experiments is needed. Currently the domain description using a specific ontology is discussed in many scientific fields. Ontologies are considered one of the pillars of the Semantic Web. The Semantic Web is suitable for machine processing hence a suitable ontology that expresses EEG/ERP experiments should help with EEG/ERP experiments interpretation.

However, current software systems are usually object-oriented and they oper-

ate over large data collections usually stored in relational databases. Because fundamental differences between semantics of object-oriented code and Semantic Web languages, it is necessary to ensure a suitable mapping.

Because expressive capabilities of the Semantic Web languages are richer than in the case of object-oriented systems it is necessary to investigate the ways to fill these semantic gaps.

1.2 Document Structure

In this work we bring an introduction into EEG/ERP experiments (Chapter 2) and describe the most often data formats. Although the presented formats were supposed as a promising attempts to standardize EEG/ERP data formats serious disadvantages stay. (Chapter 3). We present the description of these formats with their most pressing disadvantages.

Since data and their relationships can be expressed by various modeling systems we describe the differences among individual systems (Chapter 4). Since we want to design the ontology describing EEG/ERP experiments we need to extract data from common data structures and transform them into Ontology Web Language (OWL). To support this idea we investigate and describe features that provide common data structures in comparison with the Semantic Web structures (Chapter 7).

Already neuroscience databases exist, we selected the most representative set of them (Chapter 6). Some of them we use for a direct cooperation with our developed system while others we used only as an inspiration to support our idea to develop the database of EEG/ERP experiments. We also selected the most important recommendations for creating a neuroscience database that were issued by INCF. These recommendations were respected when our system was designed.

Since semantic gaps between common modeling techniques and the Semantic Web exist we needed to describe them thoroughly and investigate their removal. We present the proposed mapping and its implementation that extends common object oriented code by missing semantics (Chapter 9 and Chapter 10). The proposed mapping we practically implemented within a custom Semantic Framework.

The ontology that precisely describes the EEG/ERP experiment is presented (Chapter 11). Our effort to provide a system for storing and management of EEG/ERP experiments resulted in a custom solution called the EEG/ERP

Portal (Chapter 12).

In Chapter 13 and Chapter 14 the designed ontology is evaluated with emphasis to its usability, syntactical validity and performance.

1.3 Aims of the Dissertation Thesis

According to mentioned difficulties the aim of this thesis can be summarized in the following points:

1. To propose ontology that represents EEG/ERP domain.
2. To propose and implement an expression of proposed ontology by the Semantic Web languages.
3. To propose and implement extension of a mechanism that transforms data stored in our database into the Semantic Web.
4. To register the solution in NIF to evaluate the design and the practical contribution of the approach.

Part II

Background and State of The Art

Chapter 2

EEG/ERP Research

2.1 Introduction to EEG/ERP

Before discussing experiments it is necessary to introduce electroencephalography (EEG), event-related potentials (ERP) and how the brain works.

2.1.1 Biological Background

The core component of the nervous system (including brain, spinal cord, and peripheral ganglia) is a neuron. It is an electrically excitable cell that processes and transmits information by electrochemical signaling via connections with other cells called synapses. A neuron is basically an on/off switch. It is either in a resting state or it is shooting an electrical impulse down an axon. On the very end of axon path there is a little part that shoots out a chemical. This chemical goes across a gap (called synapse) where it triggers another neuron to send a message. Figure 2.1.1 shows a structure of a typical neuron [3].

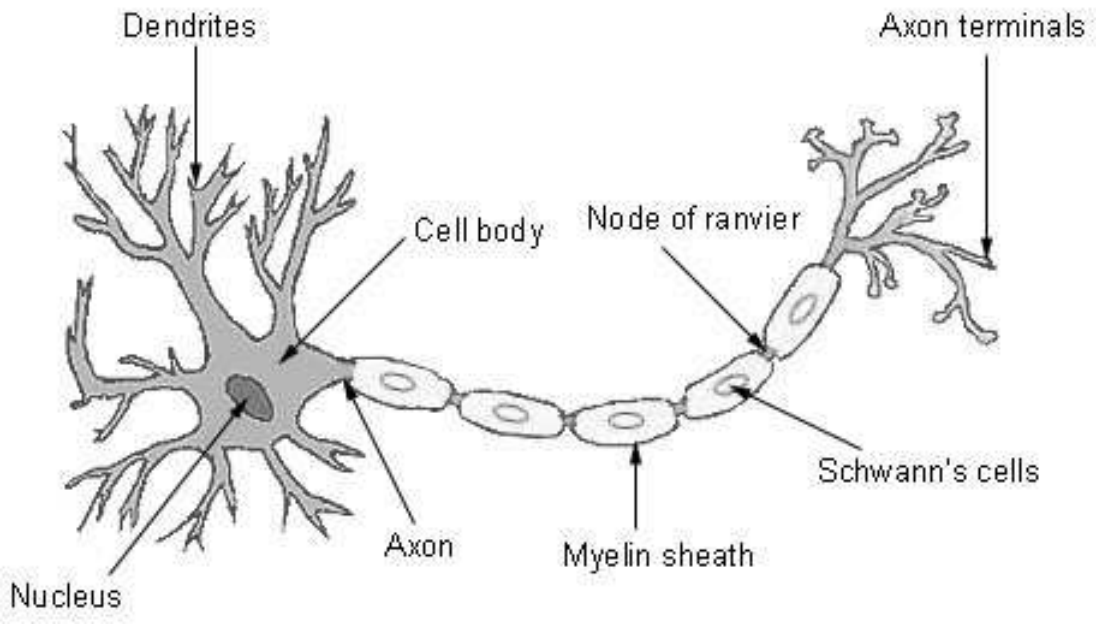

Figure 2.1.1: Neuron structure [3]

2.1.2 Electroencephalography

Electroencephalography (EEG) is a technique for recording and interpreting the electrical activity of the brain. It is a non-invasive method. The nerve cells of the brain generate electrical impulses that fluctuate rhythmically in distinct patterns. To record the electrical activity of the brain, pairs of electrodes are attached to the scalp. Each pair of electrodes transmits a signal to one of several recording channels. This signal consists of the difference in the voltage between the pair. The rhythmic fluctuation of this potential difference is shown as peaks and troughs on a line graph by the recording channel dependence on time. This graph is named electroencephalograph [1].

2.1.3 Event-Related Potentials/Evoked Potentials

Event-related brain potentials or Evoked Potentials[1] (ERP resp. EP) are derived techniques from EEG. The methods are non-invasive, they measure

[1]In this work it supposed that terms Event-Related Potentials and Evoked Potentials have the same meaning

a brain activity during e.g. the cognitive processing. The transient electric potential shifts (so-called ERP components) are time-locked to the stimulus onset (e.g. the presentation of a word, a sound, or an image). Each component reflects brain activation associated with one or more mental operations. In contrast to behavioral measures such as error rates and response times, ERPs are characterized by simultaneous multi-dimensional on line measures of polarity (negative or positive potentials), amplitude, latency, and scalp distribution. Therefore, ERPs can be used to distinguish and identify psychological and neural sub-processes involved in complex cognitive, motor, or perceptual tasks. Moreover, unlike next technique used for registering brain activity as magnetic resonance imaging (MRI) or functional magnetic resonance imaging (fMRI) (even Event-Related fMRI, which precludes the need for blocking stimulus items) is, ERP provides extremely high time resolution, in the range of one millisecond [2].

2.1.4 ERP Components

The method of averaging is used for obtaining ERP from EEG. When ERP experiment is recorded simultaneously with brain activity a position of stimulus is stored (by creating markers in the signal). The single-trial waveforms create averaged ERP waveforms for each type of stimuli at each electrode site. By doing this averaging at each time point following the stimulus its end up with highly replicable waveforms for each stimulus type.

The resulting averaged ERP waveforms consist of a sequence of positive and negative voltage deflection, which are called components.

The components are designated by letters P, N or C. P is used for positive signal, N for negative signal and C for components which are not completely positive or negative but their polarity vary. The letter is typically followed by a number which quantifies latency of the wave in milliseconds. For instance there is a component named P300 which is very often used in experiments based on the oddball paradigm described in Subsection 2.2.1. It signifies the component with positive amplitude detected after 300ms on stimuli onset. A notation of components is sometimes shorten so that we can see P3 instead of P300 but the meaning is the same [4].

2.2 EEG/ERP Experiments

2.2.1 Oddball Paradigm

The experiments based on the oddball paradigm typically contain two stimuli. Stimuli are presented in a random series such that one of them occurs relatively infrequently. The first one presented more often is called *non-target* and the second one is called *target*. Stimuli could be audio (two different tones, beeps or voices) or video (two different signs, pictures, letters or digits on the screen). The rate between stimuli is approximately 20 percent for target to 80 percent for non-target. The tested subject is instructed to be concentrated on the target stimuli or to do nothing [4, 5].

2.2.2 Simple Example Experiment

This section describes a simple EEG/ERP experiment used for demonstration of obtaining a P3 component from the EEG signal. This experiment was done in our laboratory. The experiment is based on the experiment described in [4].

The experiment is a variant on the classical oddball paradigm. Subjects view sequences of 80 percent letters Os (non-target stimuli) and 20 percent Qs (target-stimuli) and they calculate how many times Q (target stimuli) occurs. Each letter is presented on a video monitor for 100ms, followed by a 1 400ms blank interstimulus interval. While the tested subject performs this task, EEG from several electrodes embedded in an electrode cap is recorded. The EEG is converted into the digital form and stored. Whenever a stimulus is presented the stimulation computer sends a marker code to the EEG digitization computer, which stores them along with EEG data.

A simple signal averaging procedure is performed continuously during the session after each stimulus subside. Averaging extracts the ERPs elicited by the Os and the Qs. Specifically, the segment of EEG surrounding each Q and each O is extracted and lined up these EEG segments with respect to the marker code.

Figure 2.2.1 shows ERP signal for a non target stimulus O. Onset of stimulus is inserted into the coordinates of the origin.

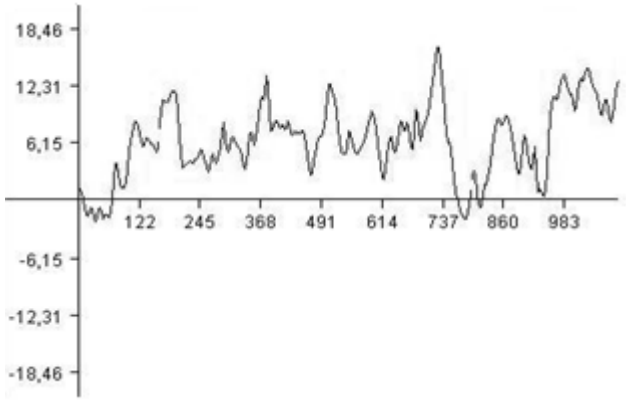

Figure 2.2.1: Graph Segment for Non-target Stimulus

Figure 2.2.2 shows ERP signal for a target stimulus Q. Onset of stimulus is inserted into the coordinates of the origin as well. Approximately after 300ms post stimulus a positive peak with much higher amplitude then neighboring extremes is present.

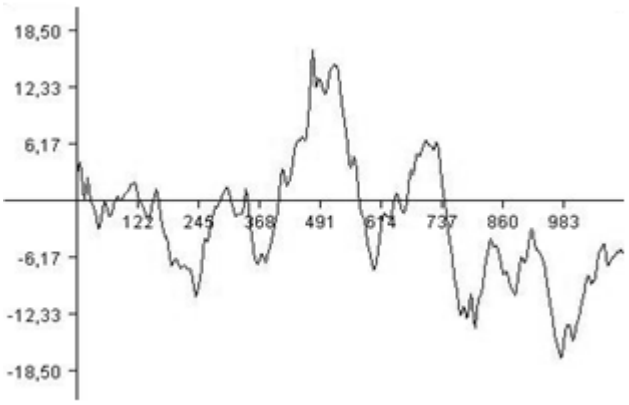

Figure 2.2.2: Graph Segment for target Stimulus

2.3 EEG/ERP Laboratory

A specialized laboratory is used for performing ERP/ERP experiments. We use 32-channels EEG recorder *BrainAmp* with *BrainVision* recording software and a custom software for presenting experimental scenarios. We use two computers. The first one is used for playing scenarios and the second one for storing EEG/ERP experimental data and watching progress of the

experiment. Both computers are connected together by a USB adapter in order to storing markers from the scenario. The tested subject sits on the seat, he/she has an EEG cap on the head and he/she is watching the scenario of the experiment on the computer screen. The attendant person is present during the experiment in order to instruct the tested subject. Laboratory equipment is presented in Figure 2.3.1.

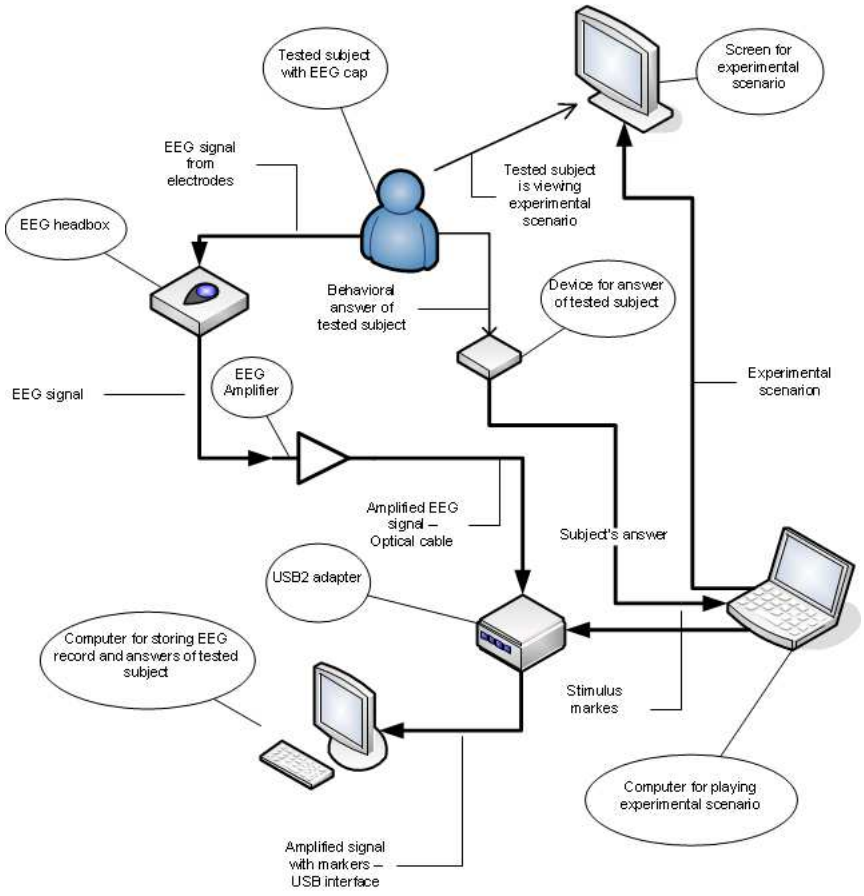

Figure 2.3.1: Laboratory Equipment [22]

Chapter 3

EEG/ERP Data Formats

3.1 Formats Overview

When EEG potentials are obtained from the scalp of a tested subject, they have to be digitalized. A special device called the analog-digital converter converts data into the digitalized form. Producers of these converters are responsible for the output format specification. Since there are many of producers of the EEG recording devices and they profit from selling a custom solution there is no general endeavor to make formats open or compatible with solutions of other producers.

The formats most often used for storing EEG data and metadata are described in this section. Some formats were developed by commercial companies. Reading or storing data usually requires using a supplied commercial software. Other described formats are open.

3.2 European Data Format

The European Data Format (EDF or EDF+ for its extension)[7] is a simple format for exchange and storage of multichannel biological and physical signals. It was developed by a few European medical engineers in 1987 who met on international Sleep Congress in Copenhagen. With the support of professor Annelise Rosenfalck, the engineers initiated the European project on Sleep-Wake analysis (1989-1992). They wanted to apply their sleep analysis algorithms to each others data and compare the analysis results. So, in Leiden in March 1990, they agreed upon a very simple common data format.

This format became known as the European Data Format first introduced in 1992 and published in [8].

3.2.1 Specification

One data file contains one uninterrupted digitized polygraphic recording. The data file consists of a header record followed by data records. The first part of header contains a set of metadata that identify tested subject, time information about the recording, the number of data records and finally the number of signals in each data record.

The first part of header is 256 bytes length and it is followed by the second part of header record that specifies type of signal, amplitude calibration or number of samples in each data record. The length of the second part is 256 bytes for each signal so total header length is possible to express by (3.2.1). The header is followed by data record where each sample is represented by two bytes integer.

$$header\,length = 256b + (ns * 256b)\,;\, ns = signals\,count \tag{3.2.1}$$

Although this format is used in some commercial (e.g. Walter Graphtek [9] or xltech [10]) and in many of open source readers and writers (e.g. Brainlab [11] or OpenXDF [12]) it has several disadvantages.

3.2.2 Disadvantages

Raw data and metadata are in one file together. Metadata contain only a restricted set of information about the tested subject. The format is not intended for ERP experiments directly because does not provide a possibility to store markers to the signal. Information about the experimental scenario is totally missing.

Despite its drawback this data format has been probably the most hopeful attempt to standardize description of EEG data.

3.3 Vision Data Exchange Format

Vision Data Exchange Format (VDEF) [13] is produced by BrainAmp device designated for reading EEG/ERP. This format is used by the Vision

Recorder developed by BrainProduct company [13]. This software and hardware equipment is used in our EEG/ERP laboratory. The Vision Recorder has the following features.

- User can control different amplifiers, software also enables an integration of new EEG/ERP formats.
- The number of channels is restricted only by the amplifier that is in use. The internal structure supports an unlimited number of channels.
- Segmentation based on event markers is available to reduce the space required by EEG/ERP files.
- Averaging based on event markers is available to form ERP during recording.
- The data can be filtered separately for display, for segmentation or averaging and for storage.

3.3.1 Specification

The format consists of three files (the header file, the marker file and the raw data) that have to be stored in one folder. The header file has an ASCII format with the extension ".vhdr". It has normally the same base name as the raw data EEG/ERP file that is described in it. It also contains the name of marker and raw data files, data format, number of channels, sampling interval and for each channel number, reference channel name, channel name, resolution and resolution unit. The format of the header file is based on the Windows INI format. The marker file, contains the name of data file, used encoding and for each marker their number, type, description, position, size and channel number.

3.3.2 Disadvantages

Although this format solves many disadvantages of EDF data format, especially data and metadata are stored separately into diverse files and the format is directly used for ERP experiments (it provides possibility to store markers), several disadvantages remain open.

The format does not define metadata about a scenario of the experiment or other a custom set of experimental metadata. Because the format is a

commercial its acceptance as a standardized EEG/ERP experimental format is questionable because its usage is bounded with the need to use it together with the specific hardware equipment from the specific vendor.

3.4 Attribute-Relation File Format

Attribute-Relation File Format (ARFF) is used internally by the Weka Machine Learning Project (WEKA)[14]. WEKA is a collection of machine learning algorithms for data mining tasks written in Java. It contains tools for regression, association rules, clustering, data pre-processing, classification, and visualization. It is also suitable for developing a new machine learning schemes.

3.4.1 Specification

ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF file contains two sections; Header and Data. Header part is marked by *header* annotation. It contains the name of the relation and the list of attributes and their types.

The data part is marked by *data* annotation and contains a set of values separated by comma. Attributes in the header part have to be ordered; they define the name of the attribute and its data type. The order of the attributes define the column position in the data section of the file. For example, if an attribute is the third one declared then WEKA expects that all attribute values will be found in the third comma delimited column in data section.

3.4.2 Disadvantages

Although ARFF is an open source format published under the General Public License[1] as well as whole WEKA project so it could be more extended, it is used only in the WEKA project.

The format does not provide possibilities to store metadata of experiments. Searching and seeking in the text file is problematic because data from each channel are stored together with metadata in one text file. Moreover, there is no possibility to store markers from ERP experiments.

[1] http://www.gnu.org/copyleft/gpl.html

3.5 Conclusion

The unification of available data formats is not without difficulties as follows from the previous text. The reason is that a format specification is defined by the vendor of a measuring device. The main disadvantage of the presented formats is that metadata description is almost missing. The data without metadata precisely describing an experiment are almost useless. We could say that interpretation of data is almost more important then data itself.

We are not able to unify existing formats until there is not an effort from the vendors of measuring devices to make an internal formats based on open standards. We describe the EEG/ERP experiment by clearly defined metadata and we design a system that will provide a possibility to store raw data obtained from the measured device and the metadata description defined in Chapter 11.

The described formats store data within the common files stored in the computer hard disc. It has several disadvantages related to long term sustainability of these data.

Chapter 4

Common Systems for Data Modeling

4.1 Introduction

Since data obtained during the EEG/ERP experiments are raw data we need to extend data by metadata description. We will investigate possibilities of the data/metadata modeling in this Section. Data/metadata have to be further processable by software tools. It includes searching, interchanging or managing.

There are several ways how data/metadata can be represented. Data/metadata are usually stored in a relational database. On the other hand, tools that work with data/metadata are usually based on the object oriented programming. Each approach uses a different way to access data. These approaches are described in the following text in detail.

4.2 Relational Databases

Relational databases are based on relational model represented by a collection of data items organized as a set of formally-described tables. Data can be accessed or reassembled in many different ways without having to reorganize the database tables. The standard user and application program interface to a relational database is the structured query language (SQL).

Each table contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns.

Next six definitions describe constitution of relational model [26].

Definition 1. (Domain)

A domain is non empty set of values with unique name commonly referred to as a data type.

Definition 2. (Scheme)

A scheme for Relation R of arity n is a list of unique attribute names A where

R=$\{A_1, \ldots A_n\}$.

Definition 3. (Relation)

A relation r on scheme R is a subset of the Cartesian product

$R \subseteq A_1 \times ... \times A_n$

We can say that R has arity n.

Definition 4. *(Relational Database)*

A relational database DB is a finite set of relations R_1, R_2, ..., R_n. *The schema for* R_1, R_2, ..., R_n *comprise the database schema for DB.*

Definition 5. (Key)

We will suppose key K, relation r and schema R.

A key for relation r in schema R is subset of R such that, for any two tuples in r, they are the same if they have same value for K.

Definition 6. (Attribute)

For a relation R of arity k, each element $X_i (i \leq 1 \leq k)$ *of some tuple t* ϵ *R can be referenced either by the ordinal value* ($X_i = t[i]$), *or by some predefined string* s_i *called an attribute* ($x_i = t[i] = t[s_i]$). *Because elements can be referenced by attribute value in this way, a relation is often called a table.*

4.3 Unified Modeling Language

4.3.1 UML Scope

The primary goal of UML is to advance the state of the industry by enabling object visual modeling tool interpretability. UML meets the following requirements:

- A formal definition of a common metamodel that specifies the abstract syntax of the UML. The abstract syntax defines the set of UML modeling concepts, their attributes and their relationships. It also defines the rules for combining these concepts to construct complete UML models.
- A detailed explanation of the semantics of each UML modeling concepts. The semantics define, in technology independent manner, how the UML concepts are realized by computers.
- A specification of the human-readable notion elements for representing the individual UML modeling concepts.

One reason UML has become a standard modeling language is that is programming language independent. UML notation set is a language not a methodology.

UML provides several types of diagrams that increase understanding an application under the development.

4.3.2 UML Diagrams Description

Use case illustrates a unit functionality provided by the system. This diagram visualizes the functional requirements of a system. It describes the relationships with actors to essential processes or the relationships among different use cases.

Class diagram shows how the different entities (data) relate to each other. It shows the static structure of the system.

Sequence diagram shows a detailed flow for a specific use case or even just part of a specific use case. It shows the calls between the different objects in their sequence.

StateChart diagram models the different states that a class can be in and how the class transitions from state to state.

Activity diagram shows the procedural flow of control between two or more class objects while processing an activity.

Component diagram provides a physical view of the system. It shows the dependencies that the software has on the other software components.

Deployment diagram shows how a system will be physically deployed in the hardware environment. It shows where the different components of the system will physically run and how they will communicate with each other.

The detailed description of the diagrams is in [43].

Because the aim of this work is to solve storage and interpretation of experimental data/metadata the suitable diagram for describing relations between data and its metadata is the *Class Diagram*.

4.3.3 Class Diagram

A class in the class diagram is represented as a rectangle with three horizontal sections. Let suppose an example with two data classes (Experiment and Person) presented in Figure 4.3.1. The upper section describes a class name. The class attributes are in the middle section. The bottom section is intended for class operations (methods) (some methods are omitted in order to keep the diagram readable). Solid lines express associations between classes. There is a few associations because one person has several relationships related to an experiment (experiment owner, member, tested or testing person, ...).

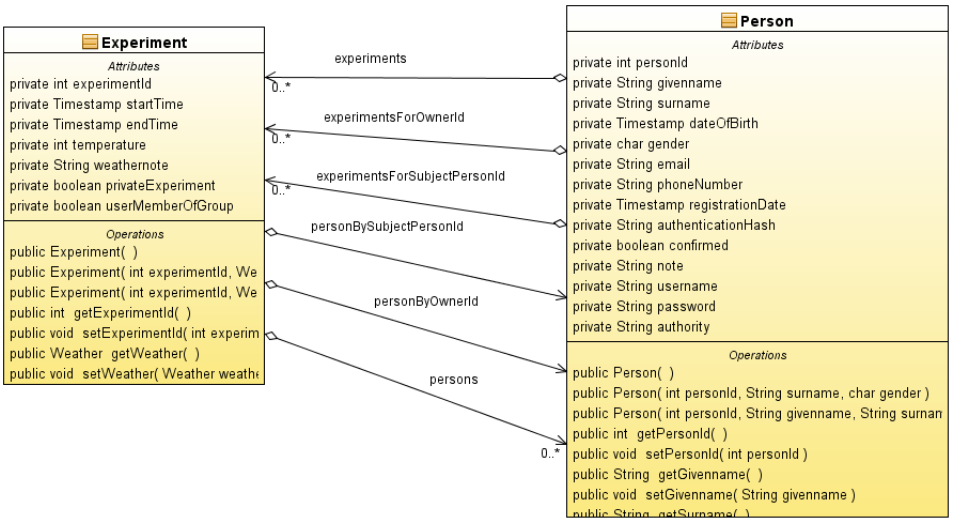

Figure 4.3.1: Example of Objects in Class Diagram

Class Diagram Kernel

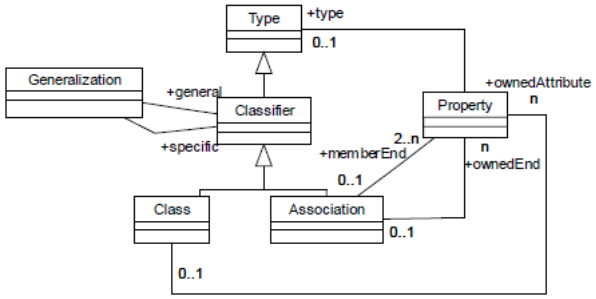

Figure 4.3.2: Key Aspects of UML Class Diagram [44]

Figure 4.3.2 shows representation of a metamodel of the UML class diagram. From this metamodel we can observe the following [44]:

- There is no direct linkage between *Association* and *Class*. The linkage is mediated by *Property*.
- A Property is a structural feature (not shown), which is typed. The model is built from structural features.
- Both *Class* and *Association* are types.

- A *Class* can have properties that characterize instances of the *Class*.
- A property may or may not be owned by a class. A property may be either navigable or not navigable[1]. Associations ends are properties.

We can express properties and their types from Figure 4.3.1 presented in Table 4.1. Table shows properties for *Experiment* and *Person* classes (only several properties are selected).

Property	**Type**
experimentId	int
startTime	TimeStamp
endTime	TimeStamp
privateExperiment	boolean
personId	int
givenname	string
surename	string

Table 4.1: Properties and Their Types in Simple Model

The classes in Table 4.2 are represented by sets of *ownedAttribute* properties.

Class	**Owned Properties**
Experiment	experimentId, startTime, endTime, privateExperiment
Person	personId, givenname, surename

Table 4.2: Classes and Owned Properties in Simple Model

Associations from Figure 4.3.1 are: *experiments, experimentsForOwnerId, experimentsForSubjectPersonId, personBySubjectPersonId, personByOwner, persons.*

If classes are implemented as in Table 4.2 the association can be modeled as the disjoint union of the owned attributes of the two classes (see Table 4.3).

[1]A navigable property is owned by a class while a non-navigable is not (it is an atomic type e.g.: integer, string, ...)

Association	Implementation
experimentForOwnerId	experimentId, startTime, endTime, privateExperiment, personId, given-name, surename

Table 4.3: Implementation of Association in Simple Model

If we know that the property *experimentId* identifies instances of *Experiment* and the property *personId* identifies instance of *Person* we can express an implementation of association as Table 4.4:

Association	Implementation
experimentForOwnerId	experimentId, personId

Table 4.4: Implementation of Association in Simple Model Using Unique IDs

4.4 Object Oriented Modeling

4.4.1 Overview

UML diagrams (mainly class diagram) describe object oriented concepts that most appropriately describe the real world. These concepts have their roots in object oriented programming. Object oriented programming languages use the concepts in practical sense. This Section describes the process of applying object oriented concepts to data modeling.

4.4.2 Data Concepts

The main construct in object-oriented modeling (OOM) is an object. Since entities and their relations exist in relational databases the similar representation exist in OOM as well. Entities are represented by objects and its relations are represented by their association ends represented by class attributes. In the OOM we need to identify objects of the system and identify the operations for these objects.

4.4.3 Objects Identity

An object can be a physical or abstract thing that encapsulates the behavior of the thing that is being modeled. All objects should be unique so every object has an identity. Similar mechanism as a key in relational database is used in OOM (see Definition 5). This identity is represented by an object identifier (OID) that distinguishes it from all other objects. OID is generated and used internally by the system to identify each object uniquely and to create and manage inter-object references. Its value is not visible to user.

The OID has the following features:

- It is immutable: The value of OID for a particular object should not change.
- It is used only once: Even if an object is removed its OID is not assigned to other objects.
- Its value does not depend on any object attribute value. If some attribute value is changed OID has still the same value.
- Its value should not depend on a physical address value in the memory. There is a mechanism for generating OID value.

The internal structure of an object includes specification of its fields. These fields hold the values, that define a state of the object.

We define objects as follows:

Definition 7. (Object Definition) *Consider a following triple t(i, c, v); where*

- i is an OID

- c is a constructor (that is an identification of how the object value is constructed)

- v is an object value (state)

4.4.4 Object Constructors

Objects are created by constructors. The constructor can be of several types depending on specific Object Oriented (OO) system. Typical constructors are: *atom, tuple, set, list, array.* The atomic value is typically *integer, real*

number, character string, boolean, dates. Different systems can also support other data types.

The object value v is interpreted on the basis of the value of the constructor c in the triple *(i, c, v)* that represents the object according to the algorithm 4.1.

Algorithm 4.1 Constructing of Object

Input:
triple *(i, c, v)* from Definition 7
Output: $c \leftarrow$ construed object
if $c = atom$ **then**
$v \leftarrow$ an atomic value from D
else
if $c = set$ **then**
$v \leftarrow$ a set of objects identifiers$[i_1, i_2, ..., i_n]$
end if
else
if $c = tuple$ **then**
$v \leftarrow$ a tuple of the form $(a_1 : i_1, ..., a_n : i_n)$
end if
else
if $c = list$ **then**
$v \leftarrow$ an ordered list of object identifiers $[i_1, ...i_n]$ of the same type
end if
else
if $c = array$ **then**
$v \leftarrow$ an array of object identifiers
end if
end if

4.4.5 Operations Encapsulation

The encapsulation related to the aim to hide the internal structure of the object. The object is accessible only through a small predefined set of operations. Some operations are used to create object (constructor), destroy objects (destructor), other operations may update the object, retrieve object value or apply some calculations to the object.

The external users access the object through the interface. The interface defines the name and arguments of the operations that the user can execute on the object. A real implementation of the method is hidden from the external users.

The interface operation is called the signature and the operation implemen-

tation is called the method. Next two terms *Inheritance* and *Polymorphism* relate with encapsulation.

Inheritance

Inheritance is a way of reusing any existing code. Inheritance is deriving objects from existing objects. The derived objects inherit properties and operations from the parent object. It is true that the derived object is a specific subtype of the parent object.

Polymorphism

Polymorphism allows the programmer to access attributes and operations with the same name from different objects using the same interface.

4.4.6 Objects Persistence

In order to store data in the relational database there is a mechanism for persisting the object into the database systems. This mechanism involves giving the object a unique persistent name though which it can be retrieved.

The mechanisms for persisting objects into the database system is called Object Relational Mapping (ORM). ORM is a mechanism that makes possible to address, access and manipulate objects without having to consider how those objects relate to their data source. ORM lets programmers maintain a consistent view of objects over time, even as the sources that deliver them, the sinks that receive them and the applications that access them change.

ORM manages the mapping details between a set of objects and underlying relational databases, while simultaneously hiding the often changing details of related interfaces from developers and the code they create [54].

Chapter 5

Semantic Web Modeling

5.1 Introduction

The World Wide Web (WWW) resp. the Internet[1] is the largest knowledge database which is available for human readers over the world. Its boom changed the way of people communication.

Currently the Internet consists of a huge amount of information, with practically no classification. It is extremely difficult to effectively handle this enormous amount of information.

Today's Web content is mostly suitable for human readers. It typically involves people's seeking and making use of information, searching for or getting in touch with other people, reviewing catalogs of on-line stores and ordering products by filling out forms and so on. Main tools used for finding relevant information are search engines such as Google, Yahoo or Alta Vista. Although search engines are widely used they have several important disadvantages:

- They have low precision. Even if the main relevant pages are retrieved, relevant or irrelevant documents are also retrieved.
- Low or no recall. Users does not often get any relevant result for their request.

[1]Between meaning of the Internet and WWW phrases is a difference. The Internet is a network of all subnetworks over the world against WWW is the way of accessing information over the Internet. Nevertheless for this work the difference between these two terms is irrelevant.

- Results are highly sensitive to vocabulary. Offered user's initial keywords don't get the results they want because relevant documents use different terminology from original query.
- Results are single Web pages. If user needs information that is spread over various documents, they have to initiate several queries to collect the relevant documents.

One possible solution is to develop increasingly sophisticated techniques based on artificial intelligence and computational linguistics.

An alternative approach is to represent Web content in a form that is more easily machine-processable and to use intelligent techniques to take advantage of these representation. One of these approaches is the Semantic Web [23].

The Semantic Web is not a separate web but it is an extension of the current one. The phrase Semantic Web was firstly introduced by inventor of WWW, URIs, HTTP and HTML sir Tim Berners-Lee [24].

The idea is to enrich the web content by semantic metadata that describe the content in order to be computer-understandable. Metadata should by expressed by special languages intended to represent data that could be understood by various kinds of software tools (often called software agents). Ontologies and set of statements translating information from various data sources into common terms and rules have to be defined. Data formats, ontologies and software agents should operate as one big application on the World Wide Web.

Although the Semantic Web is difficult for people to understand it consortium W3C[2] is working to improve, extend and standardize the system of tools, languages, publications etc. to make the Semantic Web easy to use.

Data in the WWW are typically stored in relational databases. Databases are made available in several forms on the Web where users or applications are end-users. In such cases, the semantics of data has to be made available along with the data. For human readers appropriate formats (e.g. HTML) are available but for application programs the data semantics has to be provided in a formal and machine processable form.

Data from databases are typically translated from relational model into object oriented model using object-relational mapping. The transformation of data from relational database to an Ontology can be done by two parallel approaches. The first approach includes the direct transformation of relational

[2]World Wide Web Consortium is the main international standards organization for the World Wide Web founded and headed by Tim Berners-Lee.

database into Ontology. The second approach transforms object oriented representation. Both approaches are described in Chapter 7.

5.2 Languages and Technologies

The Semantic Web is based on a layered architecture, often represented using a diagram first proposed by Tim Berns-Lee. Typical diagram representation is in Figure 5.2.1. This schema is quite old (proposed in 1999) but it can still serve as a simple illustration of the Semantic Web architecture.

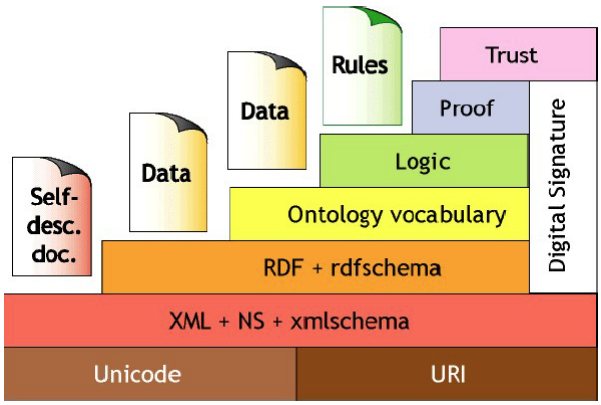

Figure 5.2.1: Semantic Web Layered Architecture [23]

Description of layers is:

- *UNICODE and URI*: Unicode is the standard for computer character representation, URI is the standard for identifying and locating resources.
- *XML*: XML and its related standards, such as Namespaces, and Schema, form a common means for structuring data on the Web but without communicating the meaning of the data.
- *Resource Description Framework*: RDF is a simple metadata representation framework, using URIs to identify Web-based resources and a graph model for describing relationships between resources. Several syntactic representations are available, including a standard XML format.

- *RDF Schema*: a simple type modelling language for describing classes of resources and properties between them in the basic RDF model. It provides a simple reasoning framework for inferring types of resources.
- *Ontologies*: a richer language for providing more complex constraints on the types of resources and their properties.
- *Logic and Proof*: an automatic reasoning system provided on top of the ontology structure to make new inferences. Thus, using such a system, a software agent can make deductions as to whether a particular resource satisfies its requirements and vice versa.
- *Trust*: The final layer of the stack addresses issues of trust that the semantic web can support. This component has not progressed far beyond a vision of allowing people to ask questions of the trustworthiness of the information on the Web, in order to provide an assurance of its quality.

In next Section technologies from Figure 5.2.1 up to the Ontologies layer are described. The layers above ontologies as well as downmost layers are beyond the scope of this work.

5.2.1 Resource Description Framework

Resource Description Framework (RDF) is a standard model for data interchange on the Web. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as two ends of the link (this relation is called triples). More formal definition follows.

Definition 8. (RDF triples)

Assume an infinitive set of RDF URI references marked U;

an infinitive set of blank nodes marked B where $B = \{b_j, j \epsilon \mathbb{N}\}$ *;*

and an infinite set of RDF literals marked L

A triple (v_1, v_2, v_3) $\epsilon(U \cup B) \times U \times (U \cup B \cup L)$ *is called an RDF triple.*

Because the linking structure of RDF triples is directed, labeled graph, from the definition 8 could be inferred Definition 9.

Definition 9. (An RDF document is directed labeled graph)

$G = (N, E, l_N, l_E)$

E (Edges) represent named links between two resources,

N (Nodes) represent resources,

l_N,l_E *represent their labels.*

Graphical representation of RDF graph looks like in Figure 5.2.2. Ellipses represent URI-identified resources, rectangles are literals and arcs are URI-identified predicates.

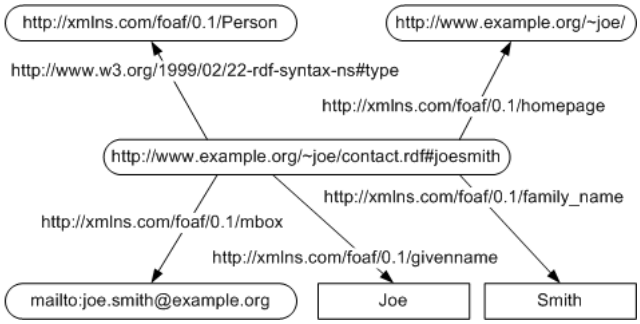

Figure 5.2.2: Example of RDF Graph[25]

Although RDF is essentially a data-model it needs any syntax. Since XML provides a uniform framework, that structure could be validated according to DTD or XSD[3], to RDF model has been given a XML syntax. As a result

[3]DTD or XSD schema languages express a set of rules to which an XML must conform in order to be valid.

RDF/XML language with XML benefits and with possibilities to express RDF triples was introduced. Other but not so common and not XML based formats are e.g. N3 or Turtle. The formal grammar for the syntax is annotated with actions generating triples of the RDF graph.

RDF contains several elements and attributes. Basic primitives are: *rdf:Resource, rdf:type, rdf:Description, rdfs:Class, rdfs:SubClassOf, rdfs:Domain, rdfs:Range, rdfs:Literal, rdfs:Property, rdfs:ConstraintResource* etc. These primitives provide possibilities to describe classes, their data types, restrictions etc.

Moreover, there are two layers; RDF and RDFS. RDFS describes a structure of classes compared to RDF which describes instances of these classes. The example is in Figure 5.2.3. The schema contains classes: *lecture, academic staff member, first-year courses* and properties: *taught by, involves, phone, employee id*. Properties are blocks, ellipses above the dashed line are classes and ellipses below the dashed line are instances.

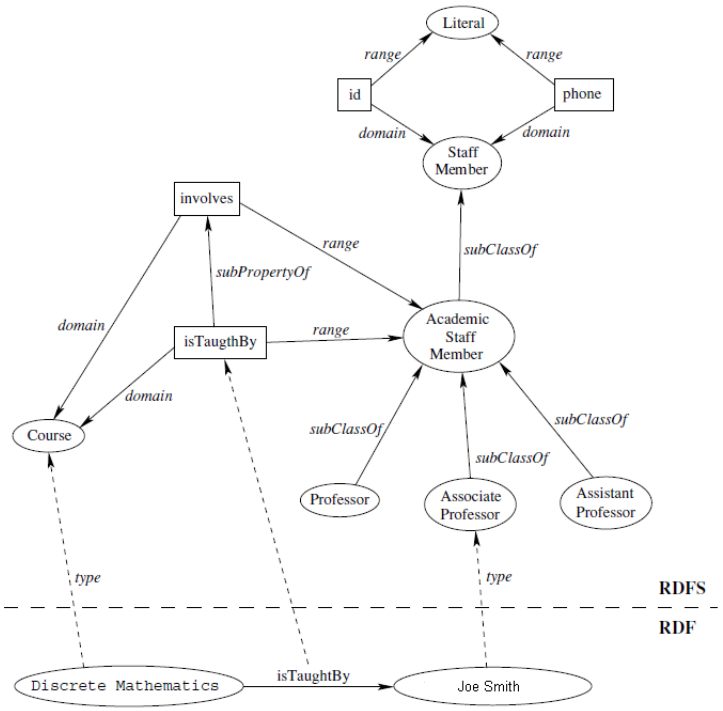

Figure 5.2.3: RDF and RDFS layers [23]

Limitations of the Expressive Power of RDF

RDF contains primitives to concern about classes and subclasses, properties and subproperties, subclasses and subproperties relationships, domain and range restrictions and instance of classes. However, a number of features are missing.

There are no properties with local range. The range of property using *rdf:range* can be defined for all classes but not only for a set of classes. Let suppose the property range "eat". It is not possible to define that cows eat only plants, while other animals may eat meat too.

Definition of disjointness classes is not possible. For example, we could say that male and female are disjoint. RDF only enables to define that female is a subclass of *Person* while male is a subclass of *Person* as well.

Further boolean combinations of classes like union, intersection or complement are not available, therefore it is not possible to define that class *Person* is a union of the classes male and female.

RDF does not solve special characteristics of properties. It is not possible to say that property is transitive, unique or the inverse of another property.

More limitations of RDF are described in [23].

5.2.2 Ontology Web Language

The expressivity of RDF is limited. RDF schema is limited to a subclasses hierarchy and properties hierarchy with domain and range definition. Because more semantic expressivity than RDF offers is required W3C defined a more powerful language named Ontology web language (OWL). The OWL has more facilities for expressing meaning and semantics than RDF. The OWL also facilitates greater machine interpretability of the web content than RDF.

There are various syntaxes available for OWL; the most often used is OWL/XML. It is the only one that is mandatory to be supported by all OWL tools.

Against the RDF the OWL allows users to write explicit formal conceptualizations of domain models. The main requirements are a well-defined syntax, efficient reasoning support, a formal semantics, sufficient expressive power and convenience of expression.

Well-defined syntax is necessary for the machine-processing of information. The formal semantics describes the meaning of knowledge precisely. It means

that the semantics does not refer to subjective intuitions, nor it is open to different interpretations by different people or machines.

The formal semantics and reasoning support are usually provided by mapping an ontology language to a known logical formalism, and by using automated reasoners that already exist for those formalisms.

Reasoning support is important because it allows one to check the consistency of the ontology and the knowledge, to check for unintended relationships between classes and automatically classify instances in classes. OWL is a richer vocabulary description language for describing properties and classes, such as relations between classes (e.g., disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes. Relationships between classes are described using following Definitions.

Definition 10. (Class membership)

We will suppose two classes A and B

If we suppose that there is x an instance of A.

Next we will suppose that B is a subclass of A ($B \subset A$)

$\Rightarrow$*x is an instance of B.*

The next Definition deals with transitivity of classes.

Definition 11. (Equivalence of classes)

We will suppose three classes A, B and C.

If A is equivalent to B and

B is equivalent to C then

$\Rightarrow$*A is equivalent to C*

The next Definition deals with consistency of the ontology

Definition 12. (Ontology Consistency)

We will suppose x which is instance of class A and

A is subclass B and C ($A \subset B \cap C$) *and A is subclass of D* ($A \subset D$),

B and D are disjoint

$\Rightarrow$*Ontology inconsistency because A should B empty, but has the instance x*

$\Rightarrow$*We have indicated an ontology error*

The next Definition deals with the classification of an individual to the class

Definition 13. (Classification)

We will suppose that certain property-value pairs are a sufficient condition for membership in a class A then

if an individual x meets all such conditions

$\Rightarrow$*x must be an instance of A*

Three Sublanguages of OWL

Because the full set of requirements for an ontology language is extensive, W3C defines OWL as three different sublanguages, each geared toward fulfilling different aspects of this full set of requirements.

1. OWL FULL - OWL full uses all the OWL language primitives. It also allows the combination of these primitives with RDF scheme. OWL full is fully compatible with RDF
2. OWL DL - OWL DL is a sublanguage of OWL Full. OWL DL for example restricts how the constructors from OWL and RDF may be used.
3. OWL Lite - OWL Lite contains more restrictions than OWL DL. For example, OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality. The advantage of this ontology is that is easy to use and implement.

Relation among OWL languages is in Figure 5.2.4. The full description of available constructs is in [55]. When we construct an ontology for EEG/ERP domain (Chapter 11) and investigate an automatic mapping from the object oriented model (Chapter 9) we express ontology at least in OWL DL semantic expressivity.

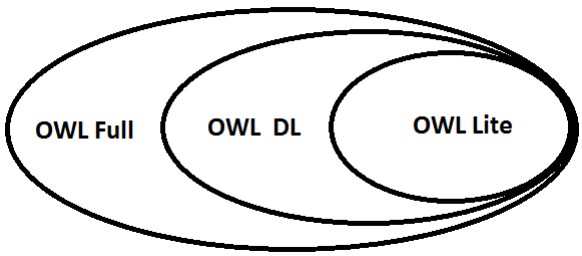

Figure 5.2.4: OWL Variants

Chapter 6

Neuroscience Databases

6.1 Introduction

Because of many problems that exists with organizing the research within the neuroscience, in January 2004 the ministers of research of the OECD countries endorsed the recommendation from the Neuroinformatics Working Group of the OECD Global Science Forum to start a global neuroinformatics initiative to coordinate international research and resources in the field. As the result, the International Neuroinformatics Coordinating Facility (INCF) was established in August 2005.

INCF develops and maintains database and computational infrastructure for neuroscientists. Software tools and standards for the international neuroinformatics community are being developed through the INCF Programs, which address infrastructures issues of high importance to the neuroscience community.

The INCF collects and makes available neuroinformatics tools in the INCF Software Center, where anyone can upload documentation, executables and related files; track use of their software; create a wiki; and establish development teams.

INCF also deals with the sustainability of neuroscience databases. The INCF formulated several recommendation important for this work. This Section summarizes the most important of them.

6.2 Sustainability

Neuroscience databases are young and dynamic field with many developments still have to be done. Databasing already gives a new flavor to the term neuroinformatics emphasizing high-throughput technologies for data generation, systematic large-scale data collection and presentation, and the development of computational tools that allow researchers to extract features and relation ships among ever-grooving amounts of data.

Neuroscience databases are provided by a diverse collection of neuroscientists. These databases provide a set of analytical tools or computational models and some of them provide possibilities for storing raw data and metadata of experiments. These resources could be useful in new research, development of methods and scientific education. The development of these databases requires several years of work focused on researchers needs with active researchers cooperation.

Nowadays there is a question how these databases sustain their activities in the long therm. INCF organized the 1st INCF Workshop on Neuroscience Database Sustainability. The goal of this workshop was to discuss issues related to the sustainability of neuroscience databases, identify problems and discuss solutions or approaches to these problems, and formulate recommendations [15, 16].

6.2.1 INCF Recommendations

INCF formulated several recommendations that should be followed when neuroscience databases are created in order to ensure long term sustainability. Extraction of recommendations useful for this work follows [15]:

- Clearly define the community (audience for the resources), identify roles and needs of each, provide mechanisms for incorporating feedback (wiki, bulletin, boards, etc.).
- Develop focused but flexible standards, follow best practices, make standards open to community.
- In developing of infrastructure for data sharing and sustainability it is critical to understand how neuroscience community is organized and how it works with data.

- Data can be safely expressed in relational schema. A comprehensive data model, integrating datasets, documents and annotations are needed. Large neuroscience datasets should be isolated.
- To use open source solutions in the maximal range, including XML, Web-Services or semantic web technologies, adherence to standards (ISO) is important.
- Datasets could be replicated at the central site, have to be formulated on ethical and patent/copyright issues, and users identificational requirements for integrated datasets.
- Technical issues include grid and web service security, access control, single sign on, etc. should not be missed out.
- INCF could identify the data resources with highest information value, and the interconnections between these resources. Then, INCF can specify which resources shall be preserved and at which schedule, which resources are not sustained, and which resources have a low information value and do not need to be sustained.
- Databases should be based on defined ontologies and schemata that are portable (in visible formats). They should allow import/export of database data in exchange formats. Query engines must be integral to databases and be defined explicitly. Languages and source code specifications must be provided for database applications.
- Data should have a markup language with metadata info for formats, experimental information, granularity, description of terminology, and minimal standards. It should be portable, scalable and extensible, and needs an ontological framework on which the data is based.
- The Web should be taken as a standard for interfaces (user interface). Each interface must have a defined API, with specifications for graphical interfaces, portability, query, and use cases.

Generally INCF should establish and moderate web-based infrastructure, identifying specific types of data/databases and investigate existing neuroscience data. The complete text of recommendations is available in [15]. Our effort in this work is respecting the maximum of these recommendations.

6.3 Available Databases

This section introduces databases developed for storing and sharing neuroscientific data.The main advantage of introduced databases is that they provide possibility to register a custom data source. Such registered resource can serve as a recognizable data source.

6.3.1 CARMEN Portal

CARMEN is a project funded by the Engineering and Physical Sciences Research Council (UK) [17]. The system CARMEN has been designed to allow neuroscientists to share data and programs from neurophysiological experiments amongst collaborators, in a secure and formally annotated manner. Core of the CARMEN is a data storage resource which is available to end-user through web interface.

The portal provides to user a set of the following objectives:

- To search achieved data
- To upload, annotate and store own experimental data
- To run processes and routines on the stored data on the CARMEN computers.

Searching the data stored in the portal is possible by using the search box in the system. This search box provides the text field where user puts entry key words; a relevant set of results is obtained. The owner can sign data as private or as public. Not logged user can see only public data.

The system provides a possibility to show the metadata associated with archived data and download data for a local processing.

Registered users can upload experimental data to the CARMEN system. The uploading process consists of several steps. One step requires completion of data by metadata description.

The Portal also enforces a privacy of archived data. Through a simple user interface the end user can specify who can access the uploaded data/metadata.

In addition, it is possible to store and run analysis tools that were used in data processing. It allows collaborators to share tools, methods and algorithms,

and provides means to run the analysis tools on the CARMEN computer resources. Uploaded tools are implemented as web-services hence they could be called locally from user's computer without their downloading. The access control list defines who can call particular services. Services could not be uploaded directly by the user but the user has to contact the CARMEN system support staff.

6.3.2 Neuroscience Information Framework

Neuroscience Information Framework (NIF) is a dynamic inventory of Web-based neuroscience resources. It includes data, materials, and tools accessible via any computer connected to the Internet.

An effort of NIF is to advance neuroscience research by providing possibilities to access public research data and tools through the Internet with requirements to use open source.

NIF is created by several participant universities including University of California, San Diego, California Institute of Technology, George Mason University, Yale University Medical College, and Washington University.

A comprehensive vocabulary for annotating and searching neuroscience resources is developed. The vocabularies are available for download as OWL files and also through the NCBO BioPortal [21]. The community news and Neuro Wiki that informs about news are published. The tools available in the current version of NIF are described in [19].

The possibility to register a custom data source is probably the most useful feature. Registered resources are actively seeking to be available through NIF. The goal of NIF is to enable users to register his/her data source within the portal. NIF indexes registered data sources. When an interested user wants to search some data he/she accesses NIF, put key words into and NIF searches data within the registered databases. Therefore the interested user can search over a lot of databases by using a uniform interface. NIF does not maintain any resource locally.

The registration within NIF is based on the three following levels:

1. Level 1 - Registration requires providing URL of user's data source and basic information about the type of data source. This level places data source into the NIF registry where is available through NIF web portal but it does not provide a direct access to the dynamic content.

2. Level 2 - It uses a XML-based script to provide a wrapper to a web

site that allows searching for key details about a requested data source including dynamic content. Content wrapping is ensured by a special tool named DISCO[1].

3. Level 3 - This level knits independently maintained databases into a virtual data federation by registering of schema information and databases views within NIF. This concept maps tables fields and values into the NIFSTD ontology[2]. Data within a source database can be combined with other databases by defining an integrated view across databases. It means that individual databases may be small but user access this data source as one virtual large database.

6.3.3 INCF Japan Node - Portal of Neuroinformatics

The Japan Node of the INCF (JNode) coordinates neuroinformatics activities in Japan and represents Japanese efforts in INCF. The Japan Node mainly domestics neuroinformatics research and directions, advises on Intellectual Property Rights and protects experimental subjects, develops and publishes brain science databases, coordinates database management, disseminates neuroinformatics information via the web portal, develops the infrastructure for brain science information and neuroinformatics and supports the development and diffusion of neuroinformatics technology.

Activities of the Japan node with relation to INCF are shown in Figure 6.3.1 [18].

Except the mentioned activities JNode has developed the portal of neuroinformatics. Within this portal it is possible to find links to the web sites of organizations that participate in the neuro research.

[1]It is the tool used as a gateway to the neuroscience database, it provides machine understandable information to integrator servers (developed by Dr. Luis Marenco at Yale University)[20].

[2]NIF Standard Ontology is composed of a collection of OWL modules covering distinct domains of biomedical reality.

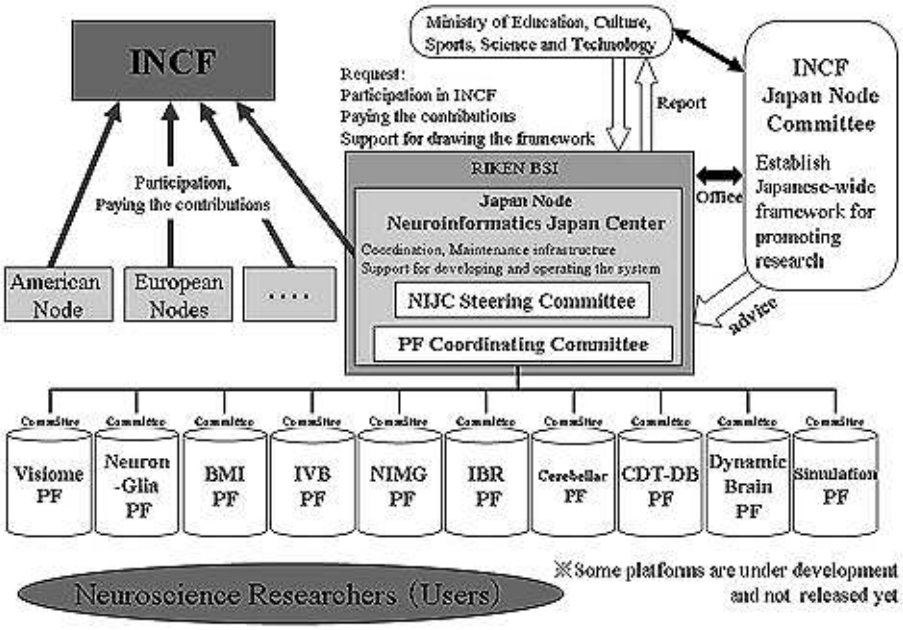

Figure 6.3.1: JNode Activities

6.4 Conclusion

Several well known databases in neuroscience were introduced in this Section. CARMEN is the well-designed portal where the user can make custom user account and share data from experiments. When the user uses some additional tools for data processing he/she can provide them as a web service. Data and services can be public or private according to the owner decision. The portal provides a suitable solution for users who do not have a proprietary solution for data sharing and they want to share a custom experiments. The portal is also suitable for users who are interested in neuroresearch but they do not have their own laboratory but they are interested e.g. in data processing. The disadvantage of CARMEN portal is that it does not provide the possibility to register users data source.

Japan portal provides a set of useful information and news from neuroscience. It contains several links to the existing data sources and a set of available software tools therefore could serve as a good guidepost. Although the possibility to add a custom data source exists, it is not possible to do it automatically (e.g. by registering a custom ontology).

The most promising project seems to be NIF where the user can find a lot of useful information, tools and data from neuroscience. The main idea of NIF is not to serve as a global database but it enables users to register a custom database. These partial databases are maintained by their owners but data are available using the unified interface (through the NIF registry). Common users have possibility to only register URL and provide description of a custom data source; while advanced users can register their OWL structure. Data in databases registered within NIF are searched by a full text search engine. Despite all advantages this solution is not addressed to users who do not have a custom data source. Such users can use NIF for obtaining available experiments, but not for sharing their experiments.

Part III

Comparison and Mapping of Data Models

Chapter 7

Comparison of Concepts

7.1 OWL and UML

7.1.1 Introduction

This section compares the features of OWL with the features of UML. First it looks at the features the two have in common, then the features in one but not the other.

A description of the translation from a model expressed in UML to an OWL expression is given in the features in common. Because there are features that exist only in OWL but not in UML the transformation has several difficulties.

The description of these difficulties is based on *Ontology Definition Metamodel* [44].

Since this work is focused on a one side transformation from Object Oriented Code to the Semantic Web structures the backward transformation from OWL to UML is not presented.

7.1.2 Similar Concepts

Classes and Properties Mapping

Both OWL and UML are based on classes. The class in OWL is a set of zero or more instances while the class is a more general construct in UML. The set of instances associated at a particular time with a class is called the class' *extent*.

In UML the extent of a class is a zero or more instances. The instance is associated with one or more classifiers while in OWL the extent of a class is a set of individuals identified by URIs. The individual is defined independently of classes. A universal class `Thing` exists in OWL. All individuals have extent of class `Thing` and all OWL classes are subclasses of `Thing`. In OWL an individual may be an instance of `Thing` and not necessary of any other class, so it could be used as a stand alone entity outside the system.

If we suppose the same example as in Figure 4.3.1 the OWL class for `Person` class can be expressed as follows:

Listing 7.1: OWL Class Example

```
<owl:Class rdf:ID="Person"/>
```

Let suppose existence of an instance of the `Person` class (see Table 7.1). `PersonId` value is an OID (Subsection 4.4.3).

Classifier	**personId**	**givenname**	**surname**	**dateOfBirth**
Person	101	Jan	Novak	1980/08/09

Table 7.1: Example Person Instance

An individual is essentially a construct with a unique name (an instance). So the instance from Table 7.1 could be expressed as the individual as follows[1]:

Listing 7.2: OWL Individual Example

```
<owl:Thing rdf:ID="Person_101"/>
```

Relationships among OWL classes are called *properties*. Properties are not necessarily tied to a specific class. By default, a property is a binary relation between `Thing` and `Thing`. The relationship among classes could be represented by two OWL constructs `owl:ObjectProperty` and `owl:DatatypeProperty`.

The UML `ownedAttribute` from Figure 4.3.2 instance is translated to `owl:ObjectProperty` if the type of *Property* is UML Class and `owl:DatatypeProperty` otherwise.

Table 7.2 shows translation of classes from Table 4.2 from OOM to OWL.

[1]Because ID has to be unique for each individual we generate an ID using class name and its id (separated by underscore). However, a different strategy for generation the individual ID in different applications can be used.

Class	**Owned Property**	**Property Type**	**OWL representation**
Experiment	experimentId	int	<owl:DatatypeProperty rdf:ID="ExperimentId"> <rdfs:domain rdf:resource="Experiment"/> <rdfs:range rdf:resource= "http://www.w3.org/2001/ XMLSchema#integer"/> </owl:DatatypeProperty>
	startTime	TimeStamp	<owl:ObjectProperty rdf:ID="StartTime"> <rdfs:domain rdf:resource="Experiment"/> <rdfs:range rdf:resource="TimeStamp"/> </owl:ObjectProperty>
	endTime	TimeStamp	<owl:ObjectProperty rdf:ID="EndTime"> <rdfs:domain rdf:resource="Experiment"/> <rdfs:range rdf:resource="TimeStamp"/> </owl:ObjectProperty>
	private Experiment	boolean	<owl:DatatypeProperty rdf:ID= "privateExperiment"> <rdfs:domain rdf:resource="Experiment"/> <rdfs:range rdf:resource= "http://www.w3.org/2001/ XMLSchema#boolean"/> </owl:DatatypeProperty>
Person	personId	int	<owl:DatatypeProperty rdf:ID="PersonId"> <rdfs:domain rdf:resource="Person"/> <rdfs:range rdf:resource= "http://www.w3.org/2001/ XMLSchema#integer"/> </owl:DatatypeProperty>
	givenname	string	<owl:DatatypeProperty rdf:ID="GivenName"> <rdfs:domain rdf:resource="Person"/> <rdfs:range rdf:resource= "http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>
	surname	string	<owl:DatatypeProperty rdf:ID="Surname"> <rdfs:domain rdf:resource="Person"/> <rdfs:range rdf:resource= "http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>

Table 7.2: Translation of Simple OOM Associations to OWL

An OWL property is also represented by UML *association* that is translated directly to an `owl:ObjectProperty`. The translation of Table 4.3 is in Table 7.3.

Association	**Owned end**	**Member end**	**OWL representation**
experiment ForOwnerId	Experiment	Person	<owl:DatatypeProperty rdf:ID= "experimentForOwnerId"> <rdfs:domain rdf:resource="Experiment"/> <rdfs:range rdf:resource="Person"/> </owl:DatatypeProperty>

Table 7.3: Translation of OOM Associations to OWL

While the UML association is always between types the OWL association has domain and range specified.

Both UML and OWL have subclasses (in OWL `owl:SubclassOf`) and support subproperties.

Advanced Concepts

There is a number of concepts appearing in both UML and OWL. Mapping of this similar concepts is straightforward, so it is not described more in depth.

Both support a separation into modules, called *package* in UML and *ontology* in OWL. Both can use a system of namespaces.

Both support fixed enumeration of elements OWL `owl:OneOf` and UML *enumeration*.

UML has two ends for associations that can be *navigable* or *non-navigable* (described in Subsection 4.3.3). OWL properties have also two ends called *domain* and *range*. When an UML association has one navigable and one non-navigable ends this association is translated into a OWL property which domain is the navigable end. The UML association with two navigable ends is translated into a pair of OWL properties where one is *inverseOf* the other.

UML properties have scope only in the class and its subclasses where they are defined while all OWL properties are properties of one superclass *Thing* therefore they have a global scope. So when UML properties have the same names in different classes they have a different semantics in each class. Names of OWL properties have to be unique because they have the same semantics everywhere they appear.

Both languages allow the class to be a subclass of more than one class, but it appears impractical in a common usage. Therefore current OO languages usually do not support the multiple inheritance.

OWL properties can be constrained by cardinality restrictions (`owl:minCardinality` and `owl:maxCardinality`). UML also supports cardinality restrictions but in OO languages possibilities for expressing cardinality are restricted. We usually express such cardinalities: only one, one or more, but usually not a specific restricted range.

In UML the association with its multiplicity is generally declared only once, whereas the OWL property can have different cardinalities for different classes.

OWL allows properties to be declared symmetric (`owl:SymmetricProperty`) or transitive (`owl:TransitiveProperty`). UML uses OCL [45].

The OWL property permits declaration of a value to be the same for all instances of a class `owl:AllValuesFrom`.

UML properties can be derived from other model constructs (generalization). Operations can be also derived in UML. The UML derivation rules cannot in general be represented in OWL. Derivation rules in UML are expressed in OCL. There is no translation of OCL into OWL.

The classifiers in UML are *private*, *protected*, *default* or *public*. OWL constructs are always *public*.

Two different objects modeled in UML may have dependencies that are not represented by UML named elements. OWL does not have a comparable feature, but RDF permits an `RDF:property` relation between very general elements classified by `RDFS:Class`. Therefore, a dependency relationship between a supplier and client UML model element is translated to a reserved name `RDF:Property` relation which domain and range are both `RDF:Class`.

7.1.3 Different Concepts

OWL Predicates

OWL class may be defined as the set of individuals which satisfy a restriction expression. These expressions can be a boolean combination of other classes (*intersectionOf, UnionOf, complementOf*), or property value restriction on properties.

The following class definition defines the class `Experiment` as a subclass of the property `experimentOwner`.

Listing 7.3: OWL Subclass Example

```
<owl:Class rdf:ID="Experiment">
  <owl:EquivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:Resource="#experimentOwner" />
      <owl:AllValuesFrom rdf:resource="#Person" />
    </owl:Restriction>
  </owl:EquivalentClass>
<owl:Class>
```

It defines individuals for which the range of `experimentOwner` is in the class `Person`. So if we know the individual to be an instance of `Experiment` we infer that it has the property `experimentOwner` and all its values associated with this property are instances of `Person`. Conversely if we have an individual which has the property `experimentOwner` and all of its values associated with that individual are `Person`, we can infer that the individual is an instance of `Experiment`. In contrast to OOM we cannot infer a class data type according to its attributes.

On the basis of mentioned above we can define the following:

Definition 14. (OWL Restriction as a Predicate Logic)

When it is possible to infer from the properties of an individual that the individual is a member of a class then we can revolve the classes and properties restriction as a sort of predicate logic.

OWL Names

In the common OOM the names within one namespace always refer to the same object, and different names always refer to a different object (*unique name assumption*). Names in OWL do not by default fulfill unique name assumption. Although the same name always refers to the same object, the object may be referred by several different names.

So, in the unique name assumption, if we have a set of names we can infer that these objects refer to the same number of objects whereas in OWL a set of objects does not warrant the inference that the set of OWL names refers to the same number of objects.

There are several constructs to discipline names. The unique name assumption can be declared using `owl:AllDifferent`. If two or more names indicate the same object they can be declared using `owl:SameAs`. When one name refers to something different referred by any other names there is `owl:DifferentFrom`.

Behavioral Concepts

UML allows to define a behavioral features using classes operations. This features can e.g. calculate property values. UML classes with their operations are essentially programs. Facilities of UML supporting programs include *operations*, *responsibilities*, *static operations*, *interface classes*, *abstract classes*. In contrast OWL is intended to only represent data and additional data semantics that enables to infer an additional data meaning.

7.1.4 Summary

Difficulties with mapping UML constructs to OWL constructs have been described in this Section. The features which are in similar are compared in Table 7.4. This table shows the individual UML constructs and their OWL equivalents. In addition, the table expresses the representation of described constructs in Java that is used in this work. The selection of this language is described in 6.2.1. UML features are grouped in clusters that are translated to a single OWL feature or a group of related OWL features.

Almost satisfactory mapping from UML constructs to OWL constructs and vice versa exists. Java mapping also exists for the most of described features. Nevertheless some constructs without straightforward mapping stay. These constructs were described in Section 7.1.3. The tables bellow clearly summarize the features of UML, OWL and Java where a satisfactory mapping does not exist. Table 7.5 summarizes the features that exist in UML but there is no straightforward way to map them into OWL. Table 7.6 shows OWL constructs with no UML mapping. Table 7.7 describes OWL constructs that are particulary possible to map in UML but a Java equivalent does not exist.

UML	**OWL**	**Java**	**Comment**
Class, atomic type, property ownedAttribute	owl:Class	class	
instance	individual	class instance	OWL owl:individual class independent
owned attribute, association	owl:property	class attribute	OWL has only global attributes
subclass, generalization	owl:subclass, owl:subproperty	*extends*	Java doesn't support multiple inheritance
enumeration	owl:oneOf	enum	
disjoint	owl:disjointWith, owl:unionOf	In Java one object is always an instance of exactly one class, but we should pay attention to class inheritance	
multiplicity	owl:MinCardinality, owl:MaxCardinality	There is no Java representation	
package	ontology	package	
dependency	RDF:property	methods parameters or return value	

Table 7.4: OWL and UML Features Comparison

navigable, non-navigable
derived
abstract classifier
classes as instances

Table 7.5: UML Features with no OWL Mapping

Things, global properties, autonomous individual
owl:allValuesFrom, owl:someValuesFrom
owl:SymetricProperty, owl:TransitiveProperty
classes as instances
rdfs:Comment, rdfs:SeeAlso, rdfs:Label, rdfs:isDefinedBy
owl:versionInfo
owl:disjointWith, owl:complementOF
owl:hasValue
owl:Cardinality
owl:inverseOf

Table 7.6: OWL Features with no UML Mapping

owl:EquivalentClass
owl:EquivalentProperty
owl:SameAs
owl:DifferentFrom, owl:AllDifferentFrom
owl:intersectionOf, owl:unionOf, owl:complementOf
rdfs:Comment, rdfs:SeeAlso, rdfs:Label, rdfs:isDefinedBy
owl:versionInfo
owl:SymetricProperty, owl:TransitiveProperty
owl:inverseOf
owl:AllValuesFrom, owl:SomeValuesFrom
owl:MinCardinality, owl:MaxCardinality
rdfs:someValuesFrom

Table 7.7: OWL Features with no Java Mapping

We can summarize that Semantic Web technologies associate three types of features used in the object oriented world. They describe reality in the conceptual level independent to technological restrictions so they are similar to UML representations in OOP. They also constitute database schema for the base of facts (RDF). Eventually they are processed by software tools in the implemented application so they are part of the implementation.

At the first sight there are several similarities between OOP (expressed by UML) and OWL. They both have classes, instances or inheritance. Both also enable defining cardinality restrictions etc.

However, in detailed view there are many differences. The substantial difference is a meaning of properties and individuals. In the UML instances and properties are removed from classes, in the OWL properties are double types; object and datatype properties. The first one links an individual to an individual and the second one links individuals to data values. The UML also does not provide support for describing anonymous classes. Ontologies are static so they do not provide possibilities to reflect changes in the time while in the UML it is possible to use the state model.

The differences are particulary caused by a different focus on the Semantic Web compared to UML and OOP. Mainly the Semantic Web is based on predicate logic and an open world assumption while object oriented systems are based on close word assumption. Several UML constructs without an equivalent in common programming languages exist. For instance a cardinality restriction does not exist in the Java Language.

The aim of the following sections is to describe a possible mapping of common programming structures into the semantic web technologies with an emphasis to cover the most of the semantic gaps.

However, when we want to investigate an automatic mapping mechanism we have to fill these semantic gaps by a suitable extension of common programming language. Since we use the Java language as the main programming language within our research the following task is to develop a mapping that translates this Java extension to appropriate OWL constructs.

7.2 Relational Schema and RDF

There are several approaches that try to solve a mapping of common data structures to the Semantic Web structures. Some of these approaches exist only as initial proposals or prototypes described in scientific papers, while some of them have been really implemented as available frameworks.

RDF triple can describe a simple fact such a relationship between two things where the predicate names the relationship, and the subject and object denote the two things. A familiar representation of such a fact might be as a row in a table in a relational database. This table has two columns, corresponding to the subject and the object of the RDF triple. The name of the table corresponds to the predicate of the RDF triple. In this table each row represents a unique instance of the subject. Such a row has to be decomposed for representation as RDF triples. Such designed table must be further normalized in order to be at least in the third normal form [56, 26].

Furthermore in RDB model, every table has a primary key. This key is typically an additional column with unique row id. The form of mapping from a row of a table to RDF triples is defined as follows:

- The primary key value corresponds to the common subject of collection of triples and the subject has an *rdf:type* property whose value is the table name.
- The column name of each table corresponds to the predicate of the RDF triple.
- The value in the cell corresponds to the object.
- A more complex fact is expressed in RDF using a conjunction of simple binary relationship.

The algorithm obtains an equivalent RDF model from the relational model, (1) creates an RDF class for each entity-table, (2) converts all primary keys

into IRI[2] class, (3) assigns a predicate IRI to each non-primary key attribute, (4) assigns an *rdf:type* predicate for each row, linking it to an RDFS class IRI corresponding to the table, (5) for each column that is neither part of primary or foreign key, constructs a triple containing the primary key IRI as the subject, the column IRI as the predicate and the column's value as the object.

The framework OntoGrate [29] combines ontology-based schema representation, first order logic inference, and some SQL wrappers. There are defined several mapping rules from the first order logic to relational scheme needed for developing SQL wrappers.

By using the set of developed features described in [29, 30, 31] it is possible to express simple ontologies by using first-predicate logic and according to mentioned rules to transform it to the relational schema. In addition, there is described merging of ontologies consisting of common elements from a source and target ontology. Given merged ontology between two sources it is again expressed in the first order logic language. There is also defined a data integration model where integration of ontologies is done in two steps.

- Query Translation: The process of extracting data expressed by one schema to answer a query posed using another schema, also known as query answering.
- Data Translation: Translating data from a source schema to a target (or integrated) schema for the purpose of information exchange.

A promising approach for mapping from RDB to RDF migration is D2RQ [36]. This framework uses a declarative language to describe mappings between relational database schema and RDF. D2RQ Platform provides possibilities to query a non-RDF database using the SparQL [34] query language, to access information in a non-RDF database using the Jena API or the Sesame API [37], to access the content of the database as Linked Data over the Web or to ask SparQL queries over the SparQL Protocol against the database. Further D2RQ consists of D2RQ engine, a plug-in for the Jena and Sesame, which uses the mappings to rewrite Jena and Sesame API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks. The last part of D2RQ platform is D2R Server, HTTP server that can be used to provide a Linked Data view, a HTML view for debugging and a SparQL Protocol endpoint over the database.

[2]IRI - Internationalized Resource Identifier is generalization of URI but may contain Unicode characters against URI that can contain only ASCII characters.

METAMorphoses [38] is data transformation processor from RDB into RDF uses the mapping described in the template XML document. The processor employs an algorithm based on author´s data transformation model, which is maintained to have a higher performance than similar solutions in the field. The tool is designed to hide the complexity of the Semantic Web technologies into the schema mapping layer, while exposing the simple template layer to the programmer.

SquirelRDF [35] is a tool which allows relational databases to be queried using SparQL. It is just an implementation of RDB to RDF mapping, thus ontology is not considered.

7.3 Existing Tools

The most of tested tools are based on Jena [33]. It is Java Framework for building the Semantic Web applications. It provides a program environment for RDF, RDFS, OWL, SparQL and includes a rule-based inference engine. Jena is open source and has grown out of work with the HP Labs Semantic Web Programme. The Jena Framework includes: A RDF API Reading and writing RDF in several formats (RDF/XML, N3 and N-Triples), The OWL API, In-memory and persistent storage or SparQL query engine. Jena is a parser which is able to read/write mentioned formats and store them into the internal model (described in Section 9.4). This model could be read by encapsulated frameworks.

7.3.1 Tools with Common Semantic Expressivity

Mapping of OWL classes to Java Interfaces is described in [28]. Every OWL class is mapped into a Java Interface containing just the accessor/mutator method declarations (set/get methods) for properties of that class. Using an interface instead of a Java class to model an OWL class is the key to expressing the multiple inheritance of properties OWL, because Java class is simple inheritance language. The corresponding Java class that embeds each interface (corresponding to the OWL class) wherein there are explicitly defined the fields (properties of the class) and the acceding methods. These interfaces allow to mapp a various set of OWL operators like *subClassOf, intersectionOf* and *oneOf*.

Properties in OWL assume multiple-cardinality so `Collection` type is used for expressing Java fields. But in Java each variable can be of one type.

It contrasts with the permitted multi-range properties in OWL. For avoiding this Java insufficiency the special set of listeners with range checkers is implemented.

Back transformation is described in [32] where an OWL processor SWCLOS3 is developed. It is on top of Common Lisp Object System (CLOS). CLOS allows lisp programmers to develop Object-Oriented systems, and SWCLOS allows lisp programmers to construct domain and task ontologies in software application fields.

A resource node in the RDF graph is represented by a CLOS object in SWCLOS. A labeled arc from a node to another is represented by a slot that belongs to an arrow-tail node and has an arrow-head node as the slot value, but *rdf:type* relation is replaced with the instance-class relation and *rdfs:subClassOf* relation is replaced with the class-superclass relation in CLOS.

The situation with OWL mapping is better because OWL representation is much more likely for objects. Especially, the property restrictions that provide the local constraints on property values for a specific domain may be straightforwardly implemented by CLOS slot definitions that belong to a class.

Sommer [39] is a very simple library for mapping Java Objects to RDF graphs and back. It uses XML/RDF template in the input. This template is extended by information from input objects.

Java2OWL-S is a tool which is able to generate OWL directly [40]. It uses two transformations. The first transformation is from JavaBeans into WSDL (Web Service Description Language). The input of this transformation is formed by Java class and the output is temporary WSDL file. The second transformation transforms temporary WSDL file into OWL (four OWL documents are created).

JenaBean [41] is a similar tool, it is a flexible RDF/OWL API to persist JavaBeans. It takes an unconventional approach to binding that is driven by the Java object model rather than the OWL or RDF schema. By default JenaBean uses typical JavaBean (see Section: 9.1) conventions to derive RDF property URI's. JenaBean against Sommer does not need any input template but generates RDF/XML representation according to JavaBean structure.

There exist several syntaxes for representation of ontologies. The OWL API [42] is a Java API and reference implementation for creating, manipulating and serializing OWL Ontologies. It includes a number of components including RDF/XML, OWL/XML; Turtle parsers and writers.

7.3.2 Tools with Additional Semantic Expressivity

Concerning one side transformations the tools described in 7.3.1 work quite satisfactorily because object-oriented code has poorer semantics than OWL. However, if we want to use more capabilities of OWL, we have to enrich object-oriented code by missing semantics.

There are several frameworks and tools that try to enrich object-oriented code by additional semantics that appears in the OWL output structure.

The ActiceRDF [46] is a library for accessing RDF data from Ruby programs. It provides a domain specific language for RDF models; it can address RDF resources, classes and properties programmatically without using e.g. SparQL queries.

This tool solves only a part of OWL and OOP mismatches due to the usage of Ruby that is a dynamic interpreted language. Namely developed framework does not need strictly typed classes and properties. Types are evaluated in runtime and can be changed dynamically. An availability to add additional semantics into source codes is missing.

The Semantic object framework (SOF)[47] utilizes embedded comments in source codes to describe semantic relationships between classes and attributes. Heterogeneous data sources could be processed using implemented parsers.

This approach seems to be promising but programmer has to insert RDF/OWL keywords into source code comments directly. It can be an obstacle for object-oriented developers. Moreover, source comments should be used for description of the class meaning, not for insertion of different language syntax. In addition, common systems are usually distributed in compiled forms while comments are presented within the source codes. Such semantics can not be processed at runtime.

The eClass [48] is a solution that changes Java syntax to embed semantic descriptions into the source code. The eClass contains data attributes, methods, inference rules and presentations. It can be implemented as an extension of an existing object-oriented programming language.

However, when a common programming syntax is changed, it affects compilers and virtual machines. It is an obstacle to use it in common systems and computers with common software installation.

Chapter 8

Mapping Improvement

8.1 Motivation

Possibilities for mapping Object Oriented Languages (especially Java) into the Semantic Web technologies were described in Section 7.3. Similarities and differences between interpretations and the practical mapping of OWL to OOP in the existing software solutions were described.

Although tools for development of ontologies did a progress in the last years (from text editors till graphical user interfaces), current tools still do not provide such user comfort as existing tools for object oriented modeling. One reason is that ontology discipline was later formalized but the larger problem is the complicated essence of ontological models.

Since the description of a specific domain by a suitable ontology is desirable construction of domain ontologies is still not well managed. Construction of an ontology can be divided in to two main parts. Firstly, we have to determine specific metadata for the described domain. Secondly, we have to solve construction of an ontology and its future maintenance in the case of future changes with the elimination of manual work. Further, we have to ensure the consistency of the ontology.

The Semantic Web languages suppose to be description languages than programming languages. It means that they are suitable for describing data but not for storing data records. Relational databases better cope with sets of extensive data collections. Since there are semantic gaps between these approaches (described in Section 7.1.3) we need to investigate a way to fill these gaps by adding missing semantics into available data sets. Because data within relational databases are accessed typically by tools often written

in an object oriented language the data layer is the suitable place where to place additional semantics.

Section 7.3 described existing tools that are able to transform an object oriented code into the OWL representation to a certain extent.

Because we are focused on to one side transformation from OO code to OWL the selected tools (described in Subsection 7.3.1) work quite satisfactory. In spite of the fact that the described tools are able to transform an input OO code to an output OWL structure the generated output is usually semantically poor due to poorer semantic expressivity of OO code against the semantic expressivity of OWL.

We investigated several approaches described in 7.3.2 that particulary add missing semantics into the input OO code. The most of tested frameworks are difficult to use either because added semantic information is insufficient or confused, or the usage of the modified compiler or interpreter is required.

8.2 Context

According to difficulties mentioned above we proposed a custom approach which allows us to map and transform common object-oriented language (Java) into an OWL output.

We suppose that the proposed solution will be also used by software engineers and not only by experts in the Semantic web field. Thus a framework based on common programming technologies is preferred.

We expect that our solution could serve a wide community of researchers that use object oriented systems and need to generate ontologies. To support this idea we decided to use only the standard syntactic structures of used programming language for data input which is extended by missing semantics.

Part IV

Construction of Ontologies

Chapter 9

Generation of Ontology from JavaBeans

9.1 JavaBean Definition

In [49] a JavaBean is defined as follows:

> A Java Bean is a reusable software component that can be visually manipulated in builder tools.

More precisely, JavaBeans are named Java classes with class attributes which are accessed only by get/set methods. The name convention is to concatenate prefix get (resp. set) with an attribute name. When an attribute is boolean type the prefix *is* is used instead of *get*. The JavaBean is often called by an abbreviation POJO.[1]

9.2 System Requirements

Before we started with the development of the custom solution we had defined several requirements to the system. Firstly, the system based on open source programming technologies is preferred. We selected the programming language Java because the majority of current open source tools are based on Java as well. The system has to comply the INCF recommendations described in Section 6.2.1 in the maximal range. The output of the system

[1] Plain Old Java Object

should be well-formed according to the OWL W3C specification with the input of the system based on common JavaBean. Because JavaBeans are common programming structures for persisting data we can apply mapping rules defined in Chapter 7. We also suppose the possibility to add missing semantics into the common JavaBean structure. This additional semantics should not require a modification of a common JavaBean syntax in order to be deployed on common operation systems[2]. Such system is compiled using a standard Java compiler and run on a standard Java Runtime Environment[3].

These requirements ensure that such system could be immediately deployed on the computers with the standard software installation. It is necessary for common users of the system. This system will be easy to use by software engineers which usually work with the familiar programming language. In general, software engineers are used to develop object oriented systems but usually they are not familiar with the development of ontologies.

9.3 Tools Overview

Because several tools for transforming a data structure from common systems to the Semantic Web representation were described in 7.3.1 we decided not to implement a custom solution from scratch but come out from an existing solution that we improve on in order to meet our requirements. We selected JenaBean with OWL API from described tools. JenaBean internally uses Jena framework that is responsible for creation of an internal semantic model. Using Jena framework facilitates integration of frameworks because the internal model is clearly defined and RDF parsers are already implemented. Using Jena model ensures abstraction of managed data.

JenaBean itself serves as a simple superstructure that provides construction of the model from the input based on common JavaBean.

JenaBean generates only the RDF/XML output in one syntax. Serialization of the output into various syntax is ensured by OWL API. We can take result of JenaBean transformation as the input of OWL API.

[2]It means operation systems where it is possible to run standard Java installation.

[3]In the time of writing this work JDK 6 is available. Due to backward compatibility in the future version of Java the system should work as well.

9.4 Jena Models

9.4.1 Introduction

Jena has become as a system with several different kinds of models [31]. The simplest way to create a model is calling the factory method `createDefaultModel()` of `ModelFactory`. It returns the interface `Model` that is a plain RDF model stored in the memory with no ontology interface.

The Jena gives a possibility to manage more than one model simultaneously using Interface `ModelMaker`. Various models are created with unique names. Such models could be saved and looked up by the specific name if needed.

The model maker can create models stored within the memory or serialized/deserialized within the file using `FileBasedModel`.

In addition Jena allows models to be created within the database by the API called RDB Maker. Models from the RDB Maker are the same as in-memory models, but they can be much larger and they could be stored permanently. Nevertheless access to these models could be significantly slower.

A specific model can be also inferred from a base model. Inference models are constructed by applying reasoners to base models. The statements deduced by the reasoner from the base model then appear in the inferred model alongside the statements from the base model itself.

But probably the most comprehensive and the most useful model for this work is an *Ontology Model* described later in Section 9.4.3.

9.4.2 RDF Model

The Jena Framework provides an extended Java API for creating and manipulating RDF graphs. Jena has object classes to represent graphs, resources, properties and literals represented by Java Interfaces.

An algorithm for creating an RDF model is in Algorithm 9.1.

9.4.3 Ontology Model

The Jena provides a Jena ontology API for reading or creating ontologies regardless of which specific Semantic Web language is used. Jena provides the interface `OntoClass` that can represent a RDFS class as an OWL class.

Algorithm 9.1 Constructing RDF Graph

Input:
m$\in \mathbb{N}$ - count of requested resources
n$\in \mathbb{N}$ - count of properties for each request
Output: *model* - Ontology Model

```
1: Model model ← ModelFactory.createDefaultModel();
2: for i = 1 → m do
3:    Resource r ← model.createResource(resourceURI);
4:    for i = 1 → n do
5:       Resource r2 ← r.addProperty(property, propertyName);
6:    end for
7: end for
```

The differences between various representations is solved using *profiles* that define permitted constructs.

To represent the differences between the various representations each of the ontology languages has a profile which lists the permitted constructs. A used profile is defined when the ontology model is created and it is bound up wit the ontology model. Jena encodes all information within construed ontology as RDF triples in the RDF model and provides possibility to add an additional semantic information available in the specific OWL profile. The ontology API does not change the RDF representation of ontologies. For manipulation with ontology predicates Jena provides a set of accessor methods on the Java classes in the Jena API. Actually when user adds an subclass, this subclass is asserted as an additional RDF statement into the model. Conversely, when user wants to e.g. gets a list of superclasses of a specific class, Jena retrieves information from the underlying RDF statement.

When we want to build an ontology based application we can use reasoners. Reasoners are able to derive an additional truth about the concepts modeled within the ontology. Jena includes a support for variety reasoners through the *Inference API*. When Jena reasoner is enabled, it creates a new RDF model that contains RDF triples from the base model and new RDF triples that are derived from reasoning. The new model is still a Jena Model so they can be used wherever the base model can be used. Figure 9.4.1 visualizes how the Jena API works.

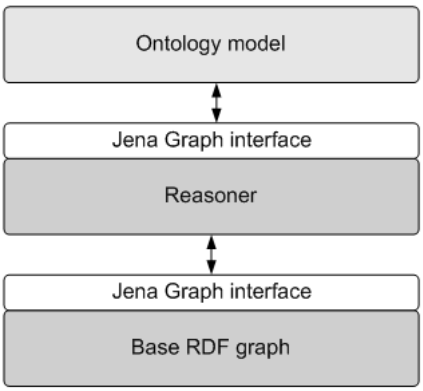

Figure 9.4.1: Jena Layers [33]

The base statements that can appear in the output ontology are held in Base RDF graph. Additional statements can be inferred from the base graph using the selected in-build reasoner. The complete set of statements resulted in the output ontology. Because both RDF graph and reasoners are accessed through the Jena Graph interface it allows us to build ontology models with or without a reasoner. In addition, the RDF graph can be represented by in-memory storage, database storage, or some other storage structure.

Resource called `OntoResource` exists in Ontology model. Some of the common attributes of ontology resources are: *versionInfo*, *comment*, *label*, *seeAlso*, *isDefinedBy*, *sameAs*, *differentFrom*.

Because classes, properties and advanced ontology expressions on properties are the building blocks of the ontology Jena provides a set of classes and methods for handling individual constructs.

The ontology class `OntoClass` accesses related classes by calling methods: *subClass*, *superClass*, *equivalentClass*, *disjointWith*.

The properties are manipulated by methods: *subProperty*, *domain*, *range*, *equivalentProperty*, *inverse*.

Advanced ontology expression on properties are represented by classes: `Restriction` (*hasValue*, *allValuesFrom*, *someValuesFrom*, *cardinality*), `Boolean`, `Intersection`, `Union`, `Complement`, `Enumeration`, etc.

The structure of Jena Ontology Model is in Figure 9.4.2.

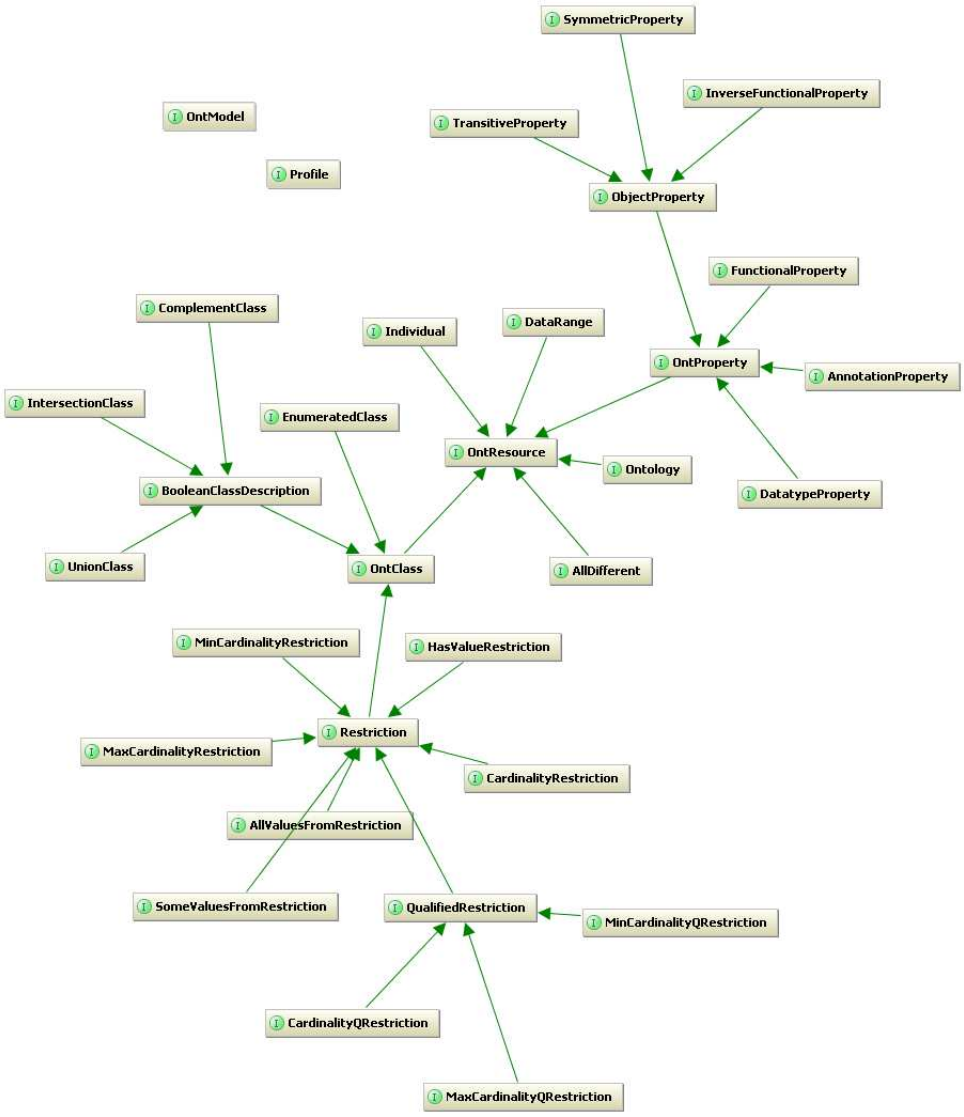

Figure 9.4.2: Jena Ontology Model

9.4.4 Conclusion

The Jena provides an abstraction of supported models. We can use the ontology model that allow us to construct the RDF triples extended by the additional semantics provided by the OWL. The Jena API hides manual construction of the semantic constructs; the user only creates the model calling the prepared API. The main interface is `OntResource`. The main subclasses

are `OntClass` and `OntProperty`. `Restriction` is inherited from `OntClass`. Additional subrestrictions are inherited from the `Restriction`. An additional restriction can be added into the model by implementing `Restriction` interface.

Jena can be easily encapsulated within various that are supposed to work with ontologies.

9.5 JenaBean - Jena Model Superstructure

Jena provides API for constructing domain ontologies. When we want to create an ontology from JavaBeans we use JenaBean modified for our needs.

JenaBean provides API for the processing of input JavaBeans and calling the JenaBean API internally. JenaBean transforms the input JavaBean structure (classes with their properties) to RDF triples. The output of this transformation is semantically poor with only a fraction of possibilities that OWL provides due to poor semantic expressivity of common JavaBean. When we want to use more capabilities of OWL we need to provide semantically enriched JavaBeans. It requires investigation and definition of semantic enrichment of the common JavaBean structure with modification of JenaBean transformational mechanism. The proposed modifications are described in Chapter 10.

9.6 JavaBean Structure Extraction

Transformation of the JavaBean into the OWL is defined as follows.

Definition 15. (JavaBean structure extraction process)

The process is the transformation of set of JavaBeans JS to an ontology O that satisfies:

For each JavaBean J_i exists OWL class OC_i in the ontology O.

For each JavaBean J_j that is a superclass of J_i exists OWL class OC_j that is a superclass of OWL class OC_i in the ontology O.

For each field of JavaBean Jf_i exists an equivalent class OC_i in the ontology O its extent is an DataType property in the ontology O when Jf_i extent is an atomic type.

For each field of JavaBean Jf_i exists an equivalent class OC_i in the ontology O its extent is an Object property when Jf_i extent is an class J_i.

For each instance of JavaBean J_i and for each field Jf_{ij} exists an equivalent OWL literal OL_{ij} in the ontology O with a corresponding value of field Jf_{ij}.

9.6.1 Mapping JavaBean to OWL Class

If it is possible to transform UML class to an OWL class (described in Chapter 7) we can analogically transform JavaBean into OWL class as well. If we suppose a persistent object `Experiment` its JavaBean representation is in Listing 9.1.

Listing 9.1: Java Class Definition

```
package cz.zcu.kiv;

public class Experiment{

}
```

Because Java classes with the same semantic meaning are grouped using packages and OWL classes are grouped using namespaces, we can simply transform the package name to the namespace name and the Java class name to the OWL class name. OWL uses the character '#'as a separator for the namespace and the class name.

Listing 9.2: OWL Class Equivalent

```
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment">

</owl:Class>
```

9.6.2 Mapping JavaBean Property to OWL Property

When we can express an UML class property as a JavaBean property we can map a Javabean property to an OWL property according to mapping described in Listing Chapter 7. In 9.3 we extend the JavaBean from Listing 9.1 by two class properties.

Listing 9.3: Java Class Property Definition

```
package cz.zcu.kiv;

public class Experiment {

    private Person testedSubject;
    private int experimentId;

    public Person getTestedSubject() {
        return testedSubject;
    }

    public void setTestedSubject(Person testedSubject) {
        this.testedSubject = testedSubject;
    }

    public int getExperimentId() {
        return experimentId;
    }

    public void setExperimentId(int experimentId) {
        this.experimentId = experimentId;
    }
}
```

One JavaBean with two class attributes is defined in Listing 9.3. The first attribute `testedSubject` is the reference (association relation) to the other class `Person` while the second one is the atomic value (integer type). The underlying differences in mapping of JavaBean properties are that `experimentId` is mapped as `DataTypeProperty` while `testedSubject` as `ObjectProperty` according to mapping described in Section 7.1.2.

Listing 9.4: OWL Property Mapping

```
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment">
</owl:Class>
<owl:ObjectProperty rdf:about="http://cz.zcu.kiv#testedSubject">
  <rdfs:domain rdf:resource="http://cz.zcu.kiv#Experiment"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="http://cz.zcu.kiv#experimentId">
  <rdfs:domain rdf:resource="http://cz.zcu.kiv#Experiment"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:DatatypeProperty>
```

9.6.3 Mapping JavaBean Instance to OWL Individual

Since JavaBeans are used to persist data we have to investigate a mapping of data from JavaBeans instances. As a suitable structure for storing a JavaBean value OWL individuals can be used. OWL individual is a typed property with the value. This property can represent the JavaBean property

value. When we suppose JavaBean from Listing 9.3 with values *[experimentId=1,testedSubject="Jan Novak"]* we can infer the following OWL structure:

Listing 9.5: OWL Individual Instance

```
<cz.zcu.kiv:Experiment rdf:about="http://cz.zcu.kiv#Experiment_1421876889">
  <cz.zcu.kiv:testedSubject>
    <cz.zcu.kiv:Person rdf:about="http://cz.zcu.kiv#Person_511811991">
      <cz.zcu.kiv:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Jan Novak</j.1:name>
    </cz.zcu.kiv:Person>
  </cz.zcu.kiv:testedSubject>
  <cz.zcu.kiv:experimentId rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
 >1</cz.zcu.kiv:experimentId>
</cz.zcu.kiv:Experiment>
```

Listing 9.5 expresses a part of OWL document with expressed values of the instance of JenaBean defined in Listing 9.3.

Serialization of *DataTypeProperty* is straightforward, we directly serialize the property value. When we want to serialize an *ObjectProperty* the transformation is more difficult because the extent of *ObjectProperty* can be a complex data type.

In this case we serialize the value that `toString` method[4] returns. When a JavaBean persists a large collection of raw binary data we do not serialize them into the output OWL document, we place there only the link that points to the data within the document.

[4]It is a method of each JavaBean thats returns a string that represents the current object.

Chapter 10

JavaBeans Semantic Extension

10.1 Prerequisites

The mapping of simple JavaBean to an OWL structure was described in Chapter 9. We presented transformation of Java classes to OWL classes and their attributes to OWL *DataType* and *Object* properties. We successfully used OWL literals for expression of instance values. However, the OWL provides much more expressivity. If we want to use more capabilities of OWL we have to enrich object-oriented code by missing semantics.

Several concepts where a straightforward transformation from OOM to OWL does not exist is presented in Section 7.1.3. In this Section we describe addition of the missing semantics into the input OOM in order to map most of OWL concepts.

When we were looking for suitable way for adding missing semantics into input JavaBeans we decided to develop a solution based on Java Annotations[1].

Java Annotation is a special form of syntactic metadata that can be added to the Java source code. *Classes, methods, variables, method parameters* and *packages* may be annotated. Java annotations may be reflective so they can be embedded within the compiled code and retrieved at runtime.

This approach has several benefits. Firstly, we can directly execute a compiled code in the transformation input (in contrast with the solutions that uses source code comments e.g. [48] described in 7.3.2). Secondly, since Java 5.0 Annotations are the part of Standard Java Development Kit, they can be processed immediately. Finally, the Java Annotations are used in current

[1] http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html

software development (by several common frameworks e.g. Spring, Hibernate, Java Persistent API), hence the software developers can work with the Java Annotations without difficulties.

Such JavaBean extension can be immediately used by software engineers. It does not matter if they are familiar with the development of the Semantic Web or not.

10.2 Mapping Realization

Annotations themselves may be annotated to indicate where they can be used. We can define if the annotation is applied to a class, a field, a method, a constructor or a package.

There are some OWL language resources that can express a class restriction and other resources that can express a property restriction.

When we were realizing an annotations mapping to individual OWL constructs we divided the set of proposed annotations to annotations with class or field scope accordingly to scope of individual OWL constructs that they describe.

An example of annotated JavaBean is showing in Listing 10.1. We demonstrate the annotated class `Experiment` with the class attribute `testedPerson` with the `Person` extent. The example demonstrates using two annotations. The first one defines an equivalent class to the `Experiment` class. The second one shows definition of an equivalent property of the `testedSubject` property. The serialization of such JavaBean is showing in Listing 10.2.

Listing 10.1: Annotated Java Bean Example

```
package cz.zcu.kiv;

@EquivalentClass("http://cz.zcu.kiv/Measurement")
public class Experiment {

  @EquivalentProperty("http://cz.zcu.kiv/TestedSubject")
  private Person testedPerson;

}
```

Listing 10.2: Annotated Java Bean OWL Serialization

```
<owl:Class rdf:about="http://cz.zcu.kiv/Measurement"/>
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment">
  <owl:equivalentClass rdf:resource="http://cz.zcu.kiv/Measurement"/>
```

```
</owl:Class>
<owl:ObjectProperty rdf:about="http://cz.zcu.kiv#testedPerson">
<owl:equivalentProperty rdf:resource="http://cz.zcu.kiv/TestedSubject"/>
  <rdfs:domain rdf:resource="http://cz.zcu.kiv#Experiment"/>
</owl:ObjectProperty>
```

10.3 Implementation of OWL Elements

We described the set of OWL concepts (Table 7.7) for that a Java mapping does not exist. This section brings a brief description of these concepts. The concepts vary according to their semantic meaning [52]. The concepts can be properties of classes restrictions, class axioms, they can define a relation of property to other properties, they can be a part of ontology header, etc. The semantic meaning and its proposed implementation is described.

10.3.1 Basic Elements

owl:equivalentClass

This property expresses that two classes have the same class extension (the same set of individuals), but not necessarily the same concepts[2]. It means that although two equivalent classes have the same instances, they does not have to be equal.

In OWL Full class equality can be expressed using *owl:sameAs*. OWL Lite or DL cannot express class equality.

Implementation: Class annotation `@EquivalentClass`. Its value must be a well-formed URI (referencing the equivalent class).

10.3.2 Property Axioms

owl:equivalentProperty

This statement links properties to be equivalent, which means they have the same extension (the same values). It does not necessarily mean that they are equal (have the same concepts).

[2]Classes in OWL provide an abstraction mechanism for grouping resources with similar characteristics. Every OWL class is associated with a set of individuals, called the class extension. It corresponds to the concept in description logic .

In OWL Full properties can be treated as individuals that is why we can describe equal properties using *owl:sameAs*. OWL Lite or DL cannot express property equality.

Implementation: Field annotation `@EquivalentProperty`.

owl:inverseOf

This statement links two properties that are inverse each other. It means they describe the same relation from the other side (some parent has a child, this child has that parent).

Implementation: Field annotation `@Inverse`. Its value must be a well-formed URI (referencing the inverse property).

owl:SymmetricProperty

This element is used to specify a property to be symmetric. Symmetric property means that the subject and the object from the triple can be interchanged and the statement is true as well. In other words, if the pair (X, Y) is an instance of a symmetric property, then the pair (Y, X) is an instance of this property too. It follows that its domain and range must be the same. *Owl:SymmetricProperty* is a subclass of *owl:ObjectProperty*.

Implementation: Implemented as the field annotation `@Symmetric` without arguments. If a field is marked by this annotation, the resulting property will be specified as an *owl:SymmetricProperty* instance. JenaBeanExtension does not check if the annotation is used properly (domain and range must be the same), it must be arranged by a programmer.

owl:TransitiveProperty

This element is used to specify a property to be transitive. It is useful primarily for inferencing. Transitive property means that if we have two pairs (X, Y) and (Y, Z) as instances of a transitive property, then the pair (X, Z) is also an instance of this property. *Owl:TransitiveProperty* is a subclass of *owl:ObjectProperty*.

Implementation: Field annotation `@Transitive` without arguments. If a field is marked by this annotation, the resulting property will be specified as an *owl:TransitiveProperty* instance.

10.3.3 Property Restriction

owl:allValuesFrom

This property restricts values of some property inside a restriction class. Its meaning is to give a range to the property under consideration, but unlike the *rdfs:range* property this constraint concerns only the restriction class. Property under consideration has no value constraint outside this class. In contrast, the *rdfs:range* property asserts that the property's values must always belong to a defined range. The restricting value can be either a class or a data range.

Implementation: Field annotation `@AllValuesFrom`. The containing java class turns into the owl's restriction class, where this constraint is valid.

owl:someValuesFrom

This property restricts values of some property inside a restriction class. Its meaning is to give a range to the property under consideration. It is similar to *owl:allValuesFrom*, but this one is less restrictive. It describes all individuals for which at least one value of the property under consideration has the defined range. This constraint concerns only the restriction class as well as *owl:allValuesFrom*. Property under consideration has no value constraint outside this class. The restricting value can be either a class or a data range.

Implementation: Field annotation `@SomeValuesFrom`.

owl:cardinality

This is a property restriction. It indicates that all individuals of the restriction class have exactly N (where N$\in$ $\mathbb{N}$) different values of the concerned property. In fact, this statement has exactly the same meaning as using both *owl:maxCardinality* and *owl:minCardinality* with the same value (N).

This element has to be used inside a *owl:Restriction* class which is usually anonymous. Except the *owl:cardinality* this class contains *owl:onProperty*. The anonymous restriction class can be used afterwards e.g. as a superclass of another class for which we want to use the restriction.

Implementation: Field annotation `@Cardinality` with an integer argument.

The difficulty of this implementation is the difference between the object code and the RDF ontology. We can say that some property's cardinality is N using `@Cardinality(N)` for the field under consideration. But we ca not create instances in Java with the cardinality greater then one (because every instance has exactly N different values of that field at the same time).

We suppose usage of this annotation for collections or arrays. The meaning is then "number of elements" of the collection.

owl:maxCardinality and owl:minCardinality

This is a property restriction. It indicates that all individuals of the restriction class have at most (res. at least) N (where N$\in \mathbb{N}$) different values of the concerned property. If both *owl:minCardinality* and *owl:maxCardinality* are used, it defines an interval to which the number of property's different values must belong.

This element must be used inside a restriction class (*owl:Restriction*) which is usually anonymous. Except *owl:maxCardinality* (resp. *owl:minCardinality*) this class also contains an *owl:onProperty* element which determines the restricted property. The anonymous restriction class can be used afterwards e.g. as a superclass of another class for which we want to use this restriction.

Implementation: Field annotation `@MaxCardinality` (resp. `@MinCardinality`).

We also suppose usage of these annotations only for collections or arrays due to the same difficulty as in the case of `@Cardinality` restriction.

10.3.4 Individuals

owl:AllDifferent

This element indicates that individuals in a group are mutually different. It has the same meaning as *owl:differentFrom*, but this property describes only two individuals. This is a more convenient way to describe a group of mutually different individuals.

Owl:AllDifferent is in fact a special class with the *owl:distinctMembers* property. This property contains a list of individuals that are pairwise different.

In OWL Full classes they can be treated as individuals. That is why we can describe different classes using this property. However, such a statement

cannot be used in OWL Lite or DL.

Implementation: Class annotation `@AllDifferent`. Usage of this annotation is inaproppriate and we currently deprecated it, because in a static code we cannot annotate individuals, but classes (OWL Full).

owl:differentFrom

This property indicates that two individuals are not the same. We can use it to express that two URIs refer to different individuals. It is the opposite to *owl:sameAs*.

In OWL Full classes can be treated as individuals that is why we can describe different classes using this property. However, such a statement cannot be used in OWL Lite or DL.

Implementation: Class annotation `@DifferentFrom`. Usage of this annotation is also deprecated, because we cannot annotate individuals, only classes, in a static code (OWL Full).

owl:sameAs

This property states that two individuals are equal. It is the opposite of *owl:differentFrom*. We can use it to express that two different URIs refer to the same thing.

In OWL Full classes and properties can be treated as individuals. That is why we can express class equality and property equality using *owl:sameAs*. However, such a statement cannot be used in OWL Lite or DL which implies that there cannot be expressed class equality or property equality in OWL Lite or DL. This property is similar to *owl:equivalentClass*, but it does not have the same meaning, likewise *owl:equivalentProperty*.

Implementation: Class annotation `@SameAs`. Use of this annotation is deprecated, because in a static code we cannot annotate individuals, but classes (OWL Full).

10.3.5 Mapping

The transformation in Section 9.6 describes a process of mapping a simple JenaBean structure into an OWL structure. When we want to extend JavaBean by the set of annotations we need to extend the described algorithm.

The following definition describes a process of transforming Java annotations into the corresponding OWL resource.

Definition 16. (Java annotation extraction process)

The process is the transformation of a set of Java annotations JA to an ontology resources OR in the ontology O that satisfies:

For each JavaBean annotation JB_{Ai} exists an equivalent OWL class resource OR_i in the ontology O when JB_{Ai} is a class annotation.

For each JavaBean annotation JB_{Ai} exists an equivalent OWL property resource OR_i in the ontology O when JB_{Ai} is a property annotation.

10.3.6 Design an Implementation of Annotations

We defined and implemented a set of Java Annotations that we map to OWL constructs. Table 10.1 shows the set of currently supported annotations and their mapping to the OWL constructs. Most of designed annotations are parameterizable with an input string parameter. The parameters shown in Table 10.1 are demonstration examples; they can be changed according to the needs of a specific domain. The set of supported annotations is still extended. New annotations are gradually proposed and implemented.

Java Annotation	**OWL construct**	**Scope**
@EquivalentClass ("http://www.kiv.zcu.cz/Person")	<owl:equivalentClass rdf:resource= "http://www.kiv.zcu.cz/Person"/>	class
@EquivalentProperty ("http://www.kiv.zcu.cz /first_name")	<owl:equivalentProperty rdf:resource= "http://www.kiv.zcu.cz /first_name"/>	field
@Symmetric	<rdf:type rdf:resource= "http://www.w3.org/2002/07/owl# SymmetricProperty"/>	class/field
@Inverse ("http://www.kiv.zcu.cz/ givenname")	<owl:inverseOf rdf:resource= "http://www.kiv.zcu.cz /givenname"/>	class/field
@AllValuesFrom ("http://www.kiv.zcu.cz/ #Persons")	<owl:allValuesFrom rdf:resource=" http://www.kiv.zcu.cz/#Persons"/>	field

@Transitive	<rdf:type rdf:resource= "http://www.w3.org/2002/07/owl# TransitiveProperty"/>	class/field
@AllDifferent ("http://www.kiv.zcu /Experiment")	<rdf:type rdf:resource="http://www.w3.org/ 2002/07/owl #AllDifferent"/>	class
@DifferentFrom ("http://www.kiv.zcu.cz /Experiment")	<owl:differentFrom rdf:resource="http://www.kiv.zcu/ Experiment"/>	class
@SameAs ("http://www.kiv.zcu.cz /Experiment")	<owl:sameAs rdf:resource="http://www.kiv.zcu/ Experiment"/>	class
@Cardinality(1)	<owl:cardinality rdf:datatype="http://www.w3.org /2001/XMLSchema#int" > 1 </owl:cardinality>	field
@MaxCardinality(1)	<owl:maxCardinality rdf:datatype="http://www.w3.org /2001/XMLSchema#int"> 1 </owl:maxCardinality>	field
@MinCardinality(1)	<owl:minCardinality rdf:datatype="http://www.w3.org /2001/XMLSchema#int"> 1 </owl:minCardinality>	field
@SomeValuesFrom ("http://www.kiv.zcu/Person")	<owl:someValuesFrom rdf:resource= "http://www.kiv.zcu/Person"/>	field

Table 10.1: OWL Mapping of Java Annotations

10.4 Semantic Framework

10.4.1 Modules Structure

The theoretical background described in previous sections resulted in the Semantic Framework that we have implemented in order to provide an advanced Java API for the Semantic Web technologies.

We provide the Semantic Framework as the library that user can integrate into the custom system. The user can use the Framework as a black box with input in the form of the set of JavaBeans and output in the form of an ontology document. The ontology document can be serialized into the several supported Semantic Web languages syntaxes. We currently support RDF/XML, OWL/XML, Turtle, Abbreviated OWL/XML formats.

Figure 10.4.1 shows the component diagram of the Semantic Framework. The framework contains three main subcomponents. The first subcomponent is the modified JenaBean. We extend the current JenaBean so that the output corresponds to the mapping described in Chapter 9. Moreover, we added the processing of Java annotations into the processing of JavaBeans structure so that we are able to transform the set of annotations described in Chapter 10. The output of the Extended JenaBean component is an internal model representation.

This internal model representation is submitted to the second, Ontology Model Creator, subcomponent. This subcomponent creates an Ontology model using an Ontology Model Factory. The internal JenaBean model is processed and an ontology document is created by calling Jena API methods.

The result model can be further processed by the last subcomponent, OWL API, that is able to transform an ontology model into the supported ontology formats.

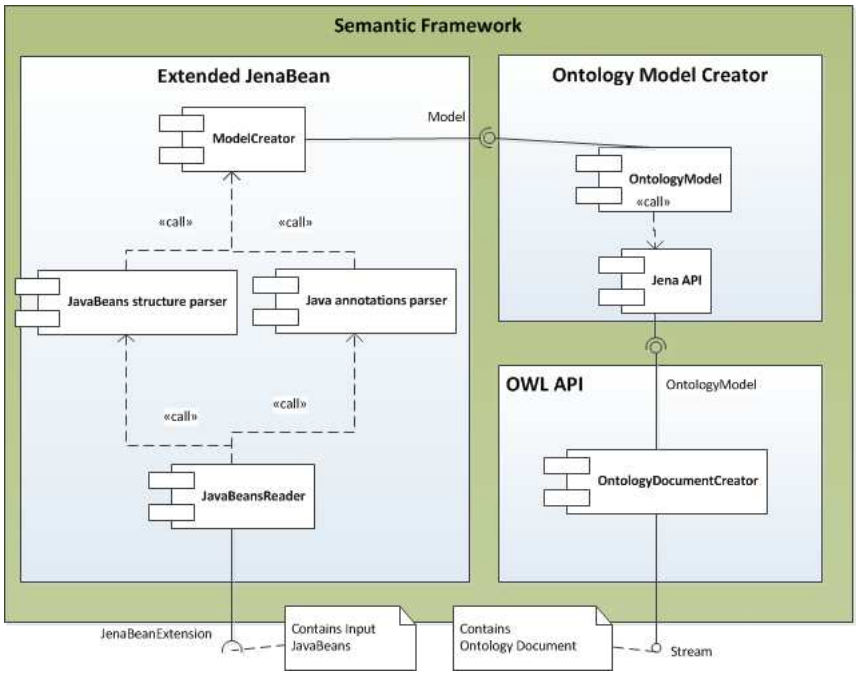

Figure 10.4.1: Component Diagram of Semantic Framework

10.4.2 Running Semantic Framework

When a user wants to use the Semantic Framework within a custom system he/she has to add the released library into his/her project. When the library is used the user needs to know only one interface (see Listing 10.3). This interface has only one overloaded method `getOntologyDocument`. The method in the implemented class returns the ontology document generated from the set of JavaBeans. We currently provide one implemented class `JenaBeanExtensionTool` where the list of input JavaBeans is used as a constructor parameter. This approach ensures that an advanced developer can use this interface as well if e.g. he/she wants to write his/her own implementation that reads the input from another structure than the set of JavaBeans is.

The provided interface gets the ontology document in the `InputStream`. The user can process this Stream according to his/her needs. It does not depend if he/she serializes it to the file, or e.g. exposes it directly on the web using e.g. the HTTP protocol.

The transformation of the obtained ontology document into a specific ontology format is provided another interface (see Listing: 10.4). This interface

provides the method `getOntologyDocument` that returns the document in the requested format.

The supported Java Annotations are included within the library as well. The user can use a defined annotation and annotate a custom JavaBeans as well. When the user annotates a custom JavaBeans with annotations from Table 10.1, his/her output document will be supplemented by additional semantics.

Listing 10.3: Semantic Framework Interface

```
/**
 * Defines user interface for working with the JenaBeanExtension library.
 */
public interface JenaBeanExtension {
 /**
  * Writes a serialization of the ontology model as a RDF/XML document.
  * This method doesn't run the transformation process itself, it only
  * creates the XML description of the ontology model.
  *
  * @return RDF/XML ontology document
  * @throws IOException if there occurred problems creating the stream
  */
 public InputStream getOntologyDocument() throws IOException;
 /**
  * Writes a serialization of the ontology model as an ontology document
  * in a specified language.
  * This method doesn't run the transformation process itself, it only
  * creates the XML description of the ontology model.
  *
  * @param lang Required language of the ontology document.
  * @return RDF/XML ontology document
  * @throws IOException if there occurred problems creating the stream
  */
 public InputStream getOntologyDocument(String lang) throws IOException;
}
```

Listing 10.4: Semantic Framework OWL API Interface

```
/**
 * Defines user interface for working with the OwlApi library.
 * This library can be used to convert the semantic model into
 * one of the standard semantic formats.
 */
public interface OwlApi {
   /**
    * Converts the semantic model into chosen semantic standard.
    * @param type semantic standard
    * @return XML description of the semantic in the chosen format
    * @throws IOException if there occurred problems creating
    *         the input stream
    * @throws OWLOntologyStorageException if there occurred problems
    *         with the format
    */
  public abstract InputStream convertToSemanticStandard(String type)
               throws IOException, OWLOntologyStorageException;
}
```

Figure 10.4.2 shows the most important classes of the framework. The interfaces that the user needs to know (shown by arrows from the Actor) are:

- `JenaBeanExtension` that generates an ontology document.
- `OwlApi` that enables transformation of the ontology document into various Semantic Web formats.

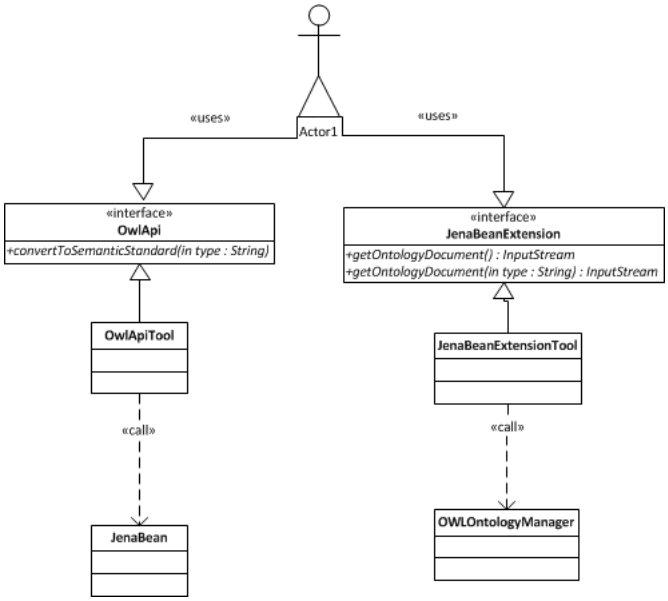

Figure 10.4.2: Class Diagram of Semantic Framework

10.4.3 Conclusion

The Semantic Framework is a complex tool for serializing common JavaBeans with additional semantics into the output OWL structure. The integration within a user system is simple because of the unified interface. The system could be used by software engineers. When such engineer does not know details about the ontology development it does not prevent him/her to work on systems that work with ontologies. Because only common programming technologies are used, the system with our framework can be immediately deployed. The input JavaBeans are semantically enriched so they still use the common Java syntax.

10.5 Annotation Tool

10.5.1 Design

Section 10.4 describes the developed Framework that transformations input annotated JavaBeans to the Semantic Web structures. If the user wants to use all capabilities the Semantic Framework provides he/she has to add an additional semantics in the form of Java Annotations.

Even though the work with the Semantic Framework is user-friendly we do the work even easier. We designed and developed the Annotation Tool that enables to annotate input JavaBeans using a well-arranged graphical tool. Such a tool minimizes the manual work needed for preparing input JavaBeans.

10.5.2 Implementation

We implemented the Annotation Tool using *Annotation File Utilities*[3]. This is an open-source tool that provides possibility to read annotations from an annotation mapping file and insert them into the class or read them from the class and write them back to the annotations mapping file. This tool is controlled by the set of Linux or Windows scripts. We extended the possibility to control this tool using a custom GUI so the annotation of the JavaBeans is more comfortable. We also provided the possibility to work with the system of projects. When the user loads the set of JavaBeans he/she can annotate them and save the work as a project within the project file. This project can be reloaded an updated whenever in future. Figure 10.5.1 shows an example of the project. The set of loaded JavaBeans can be seen in the project window. The left column shows the list of attributes of the selected JavaBean. When the user clicks on "change selected" button, a popup window with the list of available annotations appears. The user can check desired annotations and fill the parameter values within the parameterizable annotations. The tool enables a bulk change of annotations when more than one class attribute is selected (In Figure 10.5.1 the selected attributes are red colored).

[3]http://types.cs.washington.edu/annotation-file-utilities/

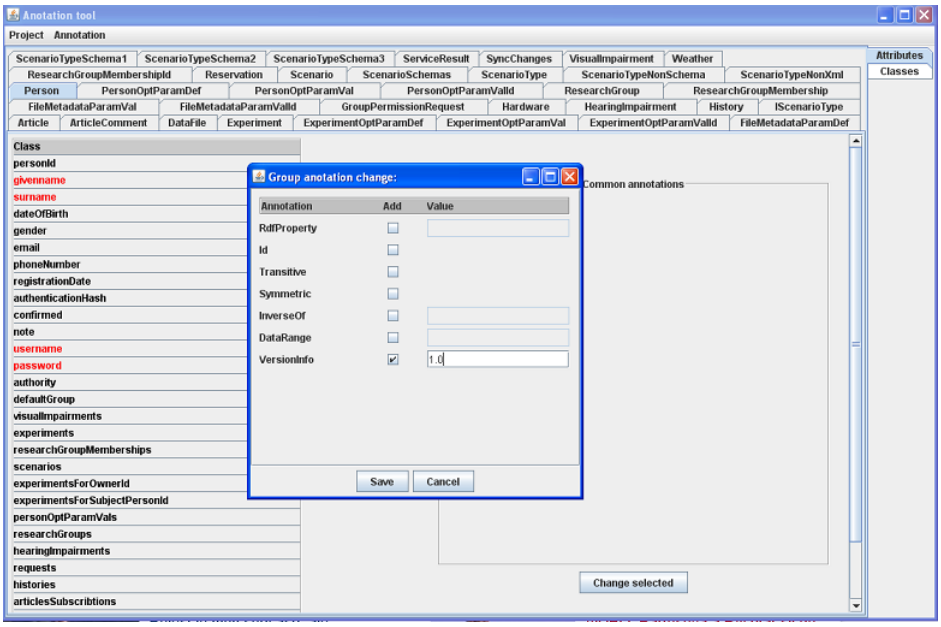

Figure 10.5.1: Annotation Tool Preview

Part V

Ontology Development in EEG/ERP Experiments

Chapter 11

Domain Ontology

11.1 Introduction

We proposed a custom ontology that precisely describes the domain of EEG/ERP experiments because of inconsistence in EEG/ERP data formats described in Chapter 3. The ontology should serve as a description of the EEG/ERP experimental data/metadata structure for the entire neuroscience community. When the community accepts this ontology as an initial standard ontology for EEG/ERP domain the interpretation of data from experiments will be unified across laboratories.

When data from experiments are precisely described by suitable metadata in a well-formed structure these experiments may be reproduced and processed across various laboratories.

The specification of the ontology originated from experience of our laboratory, co-workers from cooperating institutions, books describing principles of EEG/ERP design and data recording (e.g. [4]) and numerous scientific papers describing specific EEG/ERP experiments. It also corresponds to the effort of the INCF described in Chapter 6 and [15] in the field of development and standardization of databases in neuroinformatics.

11.2 Ontology Structure

The experimental data are supplemented by its description according to a defined protocol when we carry out an EEG/ERP experiment. The collected metadata can be divided into several semantic groups according to their

semantic meaning. We defined the following semantic groups:

- Activity
- Environment
- Tested subject
- Hardware equipment
- Software equipment
- Used electrodes
- Data digitalization
- Signal analysis
- Data presentation
- Signal artifact

The following subsections describe these semantic groups more precisely.

11.2.1 Activity

Because an experiment is carried out according to predefined scenarios (e.g. the scenario defined in Section 2.2) we need to describe the activity that take place according to the specific scenario. This description includes information about video, pictures on the computer screen, or sound sloughed off into the headphone, or information about the instructions that received the tested subject when the experiment starts. Since experiments contains a set of stimuli we propose to record which stimuli is a target and which is a non target and it's timing.

11.2.2 Environment

When the experiment may be affected by surrounding conditions we need to describe an environment and conditions where the experiment takes place. It includes: weather, time of the day and temperature.

11.2.3 Tested Subject

We also store information about the tested subject. We propose to store the following items: *Laterality* - left or right handed, *Education*, *Age*, *Gender*, *Diseases, Disability*, *Drugs*, *Optional note*.

11.2.4 Hardware Equipment

During the experiments various hardware equipment may be used. The description of the used hardware should be stored. We propose to store the type, producer and the serial number of the used hardware.

11.2.5 Software Equipment

The experiment is usually performed using supporting software equipment. This software includes software for running an experimental scenario or software for digitalizing data from electrodes. We propose to store information on each software that is used during the experiment. This description contains the name of the software, the version and the producer. When software for running an experimental scenario uses a specific configuration we propose to store this configuration (e.g. configuration files) as well.

11.2.6 Used Electrodes

Since the brain activity is measured by the set of electrodes putted on the tested subject scalp the proposed ontology describes their type, impedance, location, the used system and their fixation.

11.2.7 Data Digitalization

Before the data from electrodes are stored they are digitalized using an analogue-digital converter. This conversion is influenced by the set of parameters as filtration, sampling frequency and band-pass. The conditions of such digitalization process are described using the ontology as well.

11.2.8 Signal Analysis

The signal is analyzed using various techniques. The most often technique is averaging described in Subsection 2.1.4. The ontology describes the technique of analyzing the EEG/ERP signal. It includes the length of pre- and post-stimuli part of the signal, the number of epochs or the verbal description of signal processing procedure.

11.2.9 Data Presentation

The ontology also provides a way to describe experimental results or assumptions needed to reproduce the experiment. We propose to store: *Averaged ERP waves* - Image of averaged waves, *Grand averages* - Image of grand averages (an average over more epochs), *Evolution of ERP in time and space* - Images how the ERP is being spread over scalp, *Waves description* – description of the well-known or new waves which were formed during study, *Link to raw experimental data* – Link to all recorded raw data.

11.2.10 Signal Artifact

Since data could be affected by artifacts we included the description of this artifacts within the ontology. Artifacts may originate from muscle activities, eyes movement, etc. The ontology contains information describing a compensation method that prevents the formation of the artifact. When a method for removing of the formed artifact is used the description is also placed there. When some artifact totally degrades the signal the user can define conditions when it is possible to assume the signal as totally useless.

11.3 Ontology Visualization

Section 11.2 describes metadata that are supposed to be necessary for describing EEG/ERP experiments. We developed an ontology based on mentioned metadata. This ontology is modeled using Protége[1] a tool for modeling and visualizing ontologies. The ontology contains more than ten classes with many subclasses. Figure 11.3.1 shows the visualization of developed ontology. This ontology very precisely describes EEG/ERP experiment. This

[1] http://protege.stanford.edu/

ontology we presented as a proposal of standardized EEG/ERP experiments description. When researchers who is performing EEG/ERP experiments provide metadata according the presented ontology the experimental results will be easily reusable in future research.

In order to do the proposed ontology practically usable the following chapter describes the developed portal for management of EEG/ERP experiments. The storage of EEG/ERP experiments based on defined ontology is implemented.

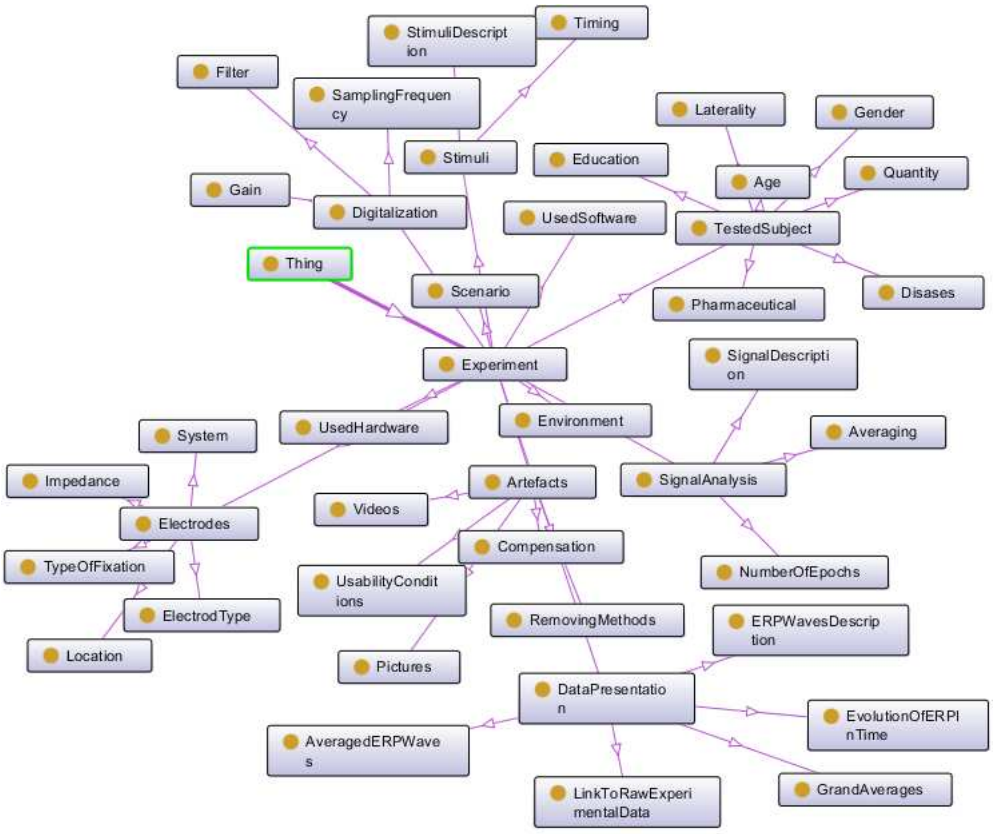

Figure 11.3.1: Ontology of EEG/ERP Experiment

Chapter 12

EEG/ERP Portal

12.1 Introduction

Because of hard manual work with large amounts of EEG/ERP data and metadata and difficulties mentioned in Chapter 3, we decided to design and implement a custom software tool suitable for EEG/ERP data and metadata storage and management.

As the result we developed the system called EEG/ERP Portal intended not only for our local research but in general it contributes to advancements in human brain understanding. In addition, we believe that such advanced software tools increase both the efficiency and the effectiveness of neuroscientific research.

The interpretation of EEG/ERP experimental data/metadata is solved by developing a custom ontology described in Chapter 11. The practical implementation of the designed ontology is ensured by the database structure and the internal logic of the developed EEG/ERP Portal.

The EEG/ERP Portal provides a possibility to store, get, manage or interchange EEG/ERP experiments among interested researchers. The internal system data storage is supposed to be implemented according to the developed ontology.

When we want the ontology to be accepted by a wide community of researchers we need to register the EEG/ERP Portal as a recognizable data source of neuroscience data. This registration is described in Section 14.4.

This Chapter describes the EEG/ERP Portal. It includes system features, system users and system user roles. Since we plan to provide the system as

an open-source we provide a brief description of the system architecture and technologies. The mapping of data from common data structures into the Semantic Web technologies according to developed ontology is also presented.

12.2 Project Scope and System Features

EEG/ERP Portal enables clinicians and various community researchers to store, update and download data and metadata from EEG/ERP experiments. The EEG/ERP Portal is developed as a stand alone product. The database access is available through a web interface. We use a web server supporting open source (Java and XML) technologies and a database system, which is able to process huge EEG/ERP data. The system is easily extensible and can serve as an open source.

The crucial user requirement is the possibility to add an additional set of metadata required by a specific EEG/ERP experiment. The complete overview of the system features and user roles (use case diagram) is available in [50].

The EEG/ERP Portal is dedicated for department users and collaborative partners as well as for a limited group of researchers interested in EEG/ERP research. The EEG/ERP Portal is supposed to be widely tested to guarantee the safety of personal information, availability of EEG/ERP resources and their usability for people interested in this research field.

12.2.1 User Roles

Since the EEG/ERP Portal is open to the whole EEG/ERP community it is necessary to protect EEG/ERP data and metadata, and especially personal data of tested subjects stored in the database from an unauthorized access. Then a restricted user policy is applied and user roles are introduced. On the basis of activities that a user can perform within the EEG/ERP Portal the following roles are proposed:

- Anonymous user has the basic access to the system (it includes essential information available in the system homepage and the possibility to create his/her account by filling the registration form).
- Reader has already his/her account in the system and can list through and download experimental data, metadata and scenarios from the sys-

tem, if they are made public by their owner. Reader cannot download any personal data or store his/her experiments into database.

- Experimenter has the same rights as the Reader; in addition he/she can insert his/her own experiments (data and metadata including experimental scenarios) and he/she has the full access to them. This user role cannot be assigned automatically, a user with the role Reader has to apply for it and the new role must be accepted by the group administrator.
- Group administrator has privileges to add a new user into the group. Each group is the autonomous unit within the system.
- Supervisor has an extra privilege to administer user accounts and change their user roles according to the policy across all groups.

12.2.2 Security

The database contains personal data, which are necessary for interpretation of experiment or for contact with tested subjects. Only the experimenter has access to personal data of tested persons who took part in his/her experiment. The collection of personal data and their storage are managed according to law.

12.2.3 Performance

The database has to work with long EEG/ERP records (usually tens of megabytes) in reasonable time. The main limiting factor is a user internet connection, not the database performance.

12.3 Design and Implementation

12.3.1 Architecture

The system uses three-layer architecture. This architectonic style is supported by selection of programming tools and technologies. We used Java and XML technologies to ensure a high level of abstraction (system extensibility) as well as a long term existence of the system as open source.

12.3.2 Persistence Layer

The persistence layer uses the Hibernate framework. It means that a relational database and an object - relational mapping are supported. Oracle 11g database server is used to ensure the processing of large data files.

12.3.3 Application and Presentation Layer

The application and presentation layers are designed and implemented using the Spring technology. This framework supports MVC architecture, Dependency injection and Aspect Oriented Programming. Integration of both frameworks, Hibernate and Spring MVC, was without difficulties. The Spring Security framework is used to ensure management of authentication and user roles. User access to the relational database is realized through the web interface. Majority of users are familiarized with web applications and they do not need any additional software except a web browser. The user interface is divided into several parts (the main menu, the second level menu, the header, the footer and a content part). Input data are validated. Error messages are presented using special marks in JSP views and by definition of CSS styles for corresponding input fields.

Storage/download of raw EEG/ERP files is universal; it is possible to store/-download any allowed file type.

Figure 12.3.1 shows the login page that is visible to not logged user. The user can see basic information about the system. He/she can see instructions for working with the EEG/ERP portal in the form of youtube videos or he/she can log-in using a custom account or using a Facebook account. When user does not have any account he/she can create one.

Figure 12.3.2 shows the home page visible for logged user. The home page provides a simple preview of various system parts. The user can access the additional system pages using the top menu.

12.3.4 Additional Modules

Since the EEG/ERP Portal is supposed to be a complex tool for working with EEG/ERP experiments including signal processing or interaction within various external systems we designed and implemented an Analytical tools module that enables running of the implemented methods. Currently we have implemented the following methods: The *matching pursuit algorithm*,

wavelet transform and *Hilbert-Huang transform* [57]. The new implemented methods are gradually added. The methods are designed and implemented as single libraries integrated within the EEG/ERP Portal.

When the user executes an individual method the method runs in a separate thread on background. The results are represented in charts when execution ends. This approach ensures that a parallel work may be done although the execution of methods may be time-consuming.

Since many research groups with different aims use various signal processing methods integration of these methods within the EEG/ERP Portal was done.

Currently, when a request for adding a method occurs it is solved individually. In the near future we plan to develop a plug-in based framework that will solve adding methods automatically.

We also designed and implemented a module for sharing our experiments with other applications using the Apache CXF Web Service Framework. Web Services in the module are secured using secured HTTP protocol when user account is used as credentials of the Web Service. This approach ensures that only the registered user is able to download experiments through the Web Service.

Figure 12.3.1: EEG/ERP Portal Login Page Preview

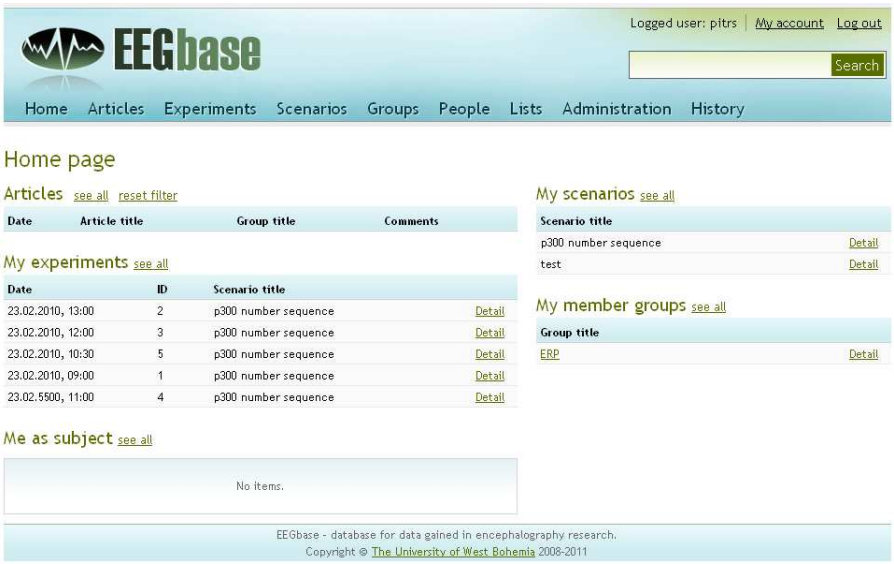

Figure 12.3.2: EEG/ERP Portal Home Page Preview

12.4 Conclusion

We presented the EEG/ERP Portal for management of EEG/ERP experiments. The set of metadata items relating to the inserted experiment, based on ontology defined in Chapter 11, is defined and implemented within the Portal. When the EEG/ERP experiment is putted in its metadata description satisfies the developed ontology. Such experiment could be easily repeated or processed by interested researchers. Due to the web interface the usage of the Portal is intuitive. The user does not need any additional software equipment except of a web browser. Today, the EEG/ERP Portal is released and available in the web site: `http://eegdatabase.kiv.zcu.cz/home.html`.

12.5 Semantic Web Extension

The EEG/ERP Portal uses a common system for preservation of experiments (the relational database, persistent JavaBeans). The EEG/ERP Portal is suitable for human readers but when we want to register the EEG/ERP Portal as a recognizable data source according to INCF recommendation (Subsection 6.2.1) we need to extract ontology from the EEG/ERP Portal into the

Semantic Web structures. We used the Semantic Framework (described in Section 10.4) to derive Semantic Web structures from the EEG/ERP Portal persistence model.

Because of the data layer consists of the relational database that is mapped to the JavaBeans using the Hibernate Framework we can straightforward map such JavaBeans into the OWL structure in the developed framework.

12.5.1 Semantic Framework Integration

The module responsible for generating the Semantic Web structure from the experiments stored within the EEG/ERP Portal uses the Semantic Framework API. The input point of the EEG/ERP Portal internal logic is created by the set of controllers according to Spring MVC design[1]. The controllers are responsible for processing HTTP requests and generating a related response. Since the software agent (OWL reasoner) reads ontologies available in the Internet we need to put our ontology on the Internet as well. We defined the set of URLs processed by `SemanticMultiController`. We defined that all portal URLs containing `*/semantic/*` are processed by the `SemanticMultiController`. The controller contains two methods `getOntology` and `getOntologyOwlApi`. The first one is responsible for generating an OWL/RDF document. The second one provides, moreover, the possibility to transform the output into various semantic formats. Since the generation of the ontology is parameterizable we transmit the input parameters using `GET` response parameters.

When the `SemanticMultiController` processes input parameters it calls the factory `SimpleSemanticFactory` through the `SemanticFactory` interface. This factory is responsible for getting *JavaBeans* from the database and handover such data to the *Semantic Framework*. The *Semantic Framework* generates the ontology according to input parameters and returns *Data Stream*. This stream is sent back to the controller that returns it to the web browser using *HTTP response*. Figure 12.5.1 shows the UML structure.

[1] http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html

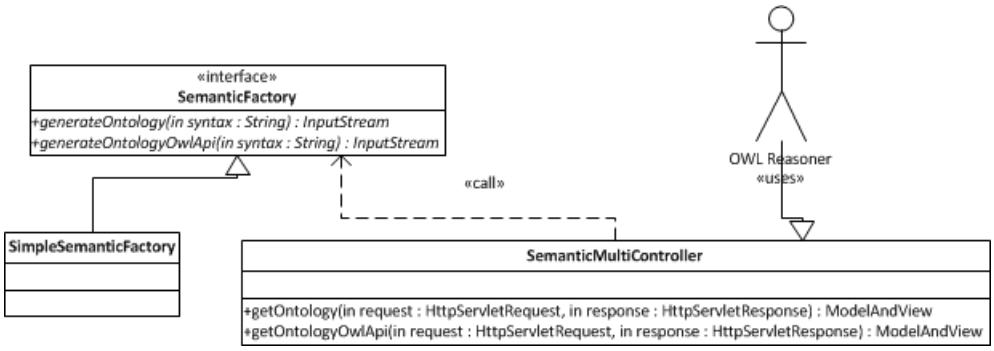

Figure 12.5.1: Semantic Framework Integration UML

The input `GET` parameter is called *type* with values `RDF/XML`, `OWL/XML` for the ontology formats and `RDF/XML-ABBREV`, `N-TRIPLE`, `TURTLE`, `N3`, `N3-PP`, `N3-PLAIN`, `N3-TRIPLE` for the ontology syntaxes.

For instance, when a reasoner visits the URL `http://eegdatabase.kiv.zcu.cz/semantic/getOntologyOwlApi.html?type=turtle` it obtains the OWL document in the *turtle* syntax. This approach ensures that the ontology is regenerated after each request so it is always up to date. Figure 12.5.2 shows the integration of *the Semantic Framework* within the EEG/ERP Portal.

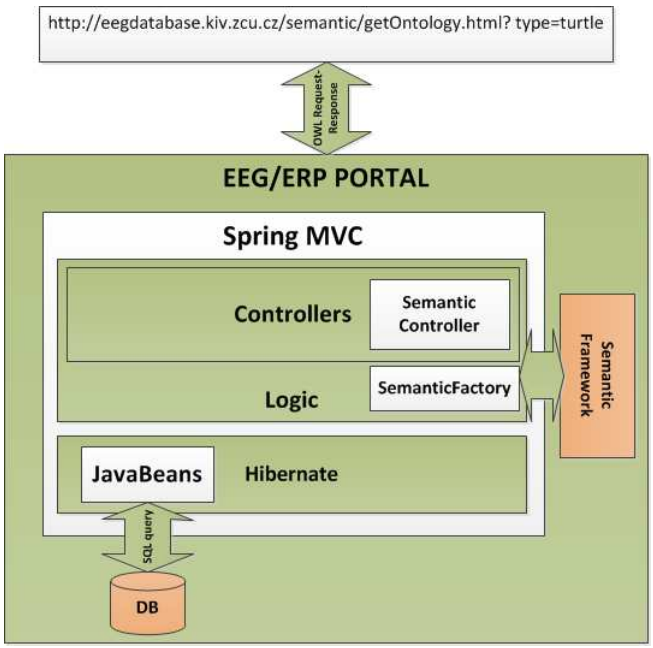

Figure 12.5.2: Semantic Framework Integration

Part VI

Results

Chapter 13

Performance Evaluation

13.1 Computational Complexity

When we want to make the developed framework usable for transforming ontological documents from systems working with real data, we need to discuss the complexity of implemented algorithm.

We analyze the time complexity of the transformational algorithm in this section. Here we consider the *transformation* operations and ignore the *preprocessing* operations (Obtaining of JavaBeans from the database).

Firstly, we derive time complexity according to the complexity of the implemented algorithm, then we do the set of experiments that should prove practical usability of the framework.

The algorithm that translates the set of input JavaBeans into the output Ontology document is expressed in Algorithm 4.1. The algorithm is based on Definition 15 that describes individual steps for transforming ontology from a JavaBean. Algorithm 13.1 shows implementation details of the transformation process.

The algorithm consists of three cycles: extracting classes $\vee$ properties of partial classes $\vee$ annotations of classes and its properties. The extraction of classes (lines 1 to 3 of the algorithm) is executed in time N, we denote it N_{C}. The extraction of properties (lines 4 to 10 of the algorithm) is executed in time N, we denote it N_{P}. The extraction of class or properties annotations (lines 11 to 18 of the algorithm) is in the worst case N+M (when *else* branch is executed). We can simplify it and substitute it $2N_{\mathrm{A}}$.

Summing up the above basic operations, the amount of work done by the

algorithm is:

$$F(N) = N_{\mathrm{C}} + N_{\mathrm{P}} + 2N_{\mathrm{A}} \approx 4N \tag{13.1.1}$$

We can infer (from (13.1.1)) the worst-case complexity of the translation algorithm is O(N). This complexity expresses that the transformation algorithm is directly proportional to the complexity of the input data.

Algorithm 13.1 Translation JavaBeans to Ontology

Input:
Set of JavaBeans
$JB = (C_{\mathrm{JB}}, P_{\mathrm{JB}}, A_{\mathrm{JB}})$ **where**
C_{JB}: classes,
P_{JB}: properties,
A_{JB}: annotations of given JavaBean

Output:
Ontology
$O = (C_{\mathrm{O}}, O_{\mathrm{O}}, D_{\mathrm{O}}, R_{\mathrm{O}}, S_{\mathrm{O}})$ **where**
C_{O}: classes,
O_{O}: object properties,
D_{O}: datatype properties,
R_{O}: restrictions,
S_{O}: subclases of given ontology
The translation from JavaBeans JB according to translation ϕ results in ontology O following:

1: **for all** classes C $\in C_{\mathrm{JB}}$ **do**
2: create a class $\phi(C) \in$ O
3: **end for**
4: **for all** properties P $\in P_{\mathrm{JB}}$ **do**
5: **if** P $\in$ T where T $\in \{real, integer, boolean, enum, set, string\} \rightarrow$ primitive type **then**
6: create a DataType property $\phi(D) \in$ O
7: **else**
8: create an Object property $\phi(O) \in$ O
9: **end if**
10: **end for**
11: **for all** annotations A $\in A_{\mathrm{JB}}$ **do**
12: **if** A $\in$ N where N $\in \{@differenFrom, @equvivalentProperty\}$ **then**
13: create a subproperty $\phi(P) \in \{C_{\mathrm{O}}, P_{\mathrm{O}}\} \in$ O
14: **else**
15: create a subclass $\phi(S) \in \{C_{\mathrm{O}}, P_{\mathrm{O}}\} \in$ O
16: create a restriction $\phi(R) \in \{C_{\mathrm{O}}, P_{\mathrm{O}}\} \in$ O
17: **end if**
18: **end for**

13.2 Experimental Verification

The computational complexity was demonstrated in Algorithm 13.1 and deduced in Formula (13.1.1). The theoretical calculation was practically verified in Table 13.1.

We designed a test scenario where we tested the time needed for transforming the set of input JavaBeans of various size to an ontology document. Firstly, we prepared the set of *Experiment* instances. Later, we assigned instances of *Person*, *Scenario*, *Hardware* and *Data* to each experiment. The class *Person* was extended by the set of supported annotations (see Table 10.1).

The tests were executed on the computer with processor Intel®Core™i5-2500, 3,3GHz with 6Gb of RAM assigned to Java Virtual Machine. The tests were run ten times and the result was created as average of partial results.

The first column of the table shows the number of created *Experiments* in individual tests. We increased the number of instances in each test. The other columns represent the time (in milliseconds) needed to generate the output ontology document for individual supported syntaxes. We tested *RDF/OWL/XML*, *RDF/XML ABBREVIATED*, *Turtle*, *N3 triple*, *N3 plain*, *N3-PP*, *N-triple* syntaxes. Syntaxes [53] are functionally equivalent they differ only in the format of the serialized output document. Figure 13.2.1 shows the chart representation of the results from Table 13.1. The linearity of chart shows the legitimacy of our claim from Formula (13.1.1) for all syntaxes.

Num.	**RDF /XML**	**RDF /XML -ABBR.**	**TTL**	**N3 -TRIP.**	**N3**	**N3 -PL.**	**N3 -PP**	**N -TRIP.**
1000	2140	1241	742	442	559	538	509	517
10000	5486	7999	4838	3732	4906	4702	4748	4618
10500	5596	8434	5037	3823	5149	4966	4944	4922
10800	5881	8480	5210	3852	5232	5311	5128	4930
19000	10591	18272	10297	6878	9142	9132	9371	8849
20000	11833	17374	9774	7358	9750	10262	11178	12663
100000	57439	82667	56997	37269	50045	49137	57613	46777
150000	83467	127402	73511	60790	76899	76344	79515	73764
200000	116840	179608	100304	83623	98697	98621	98499	110474

Table 13.1: Time Dependence on the Set of Input Objects

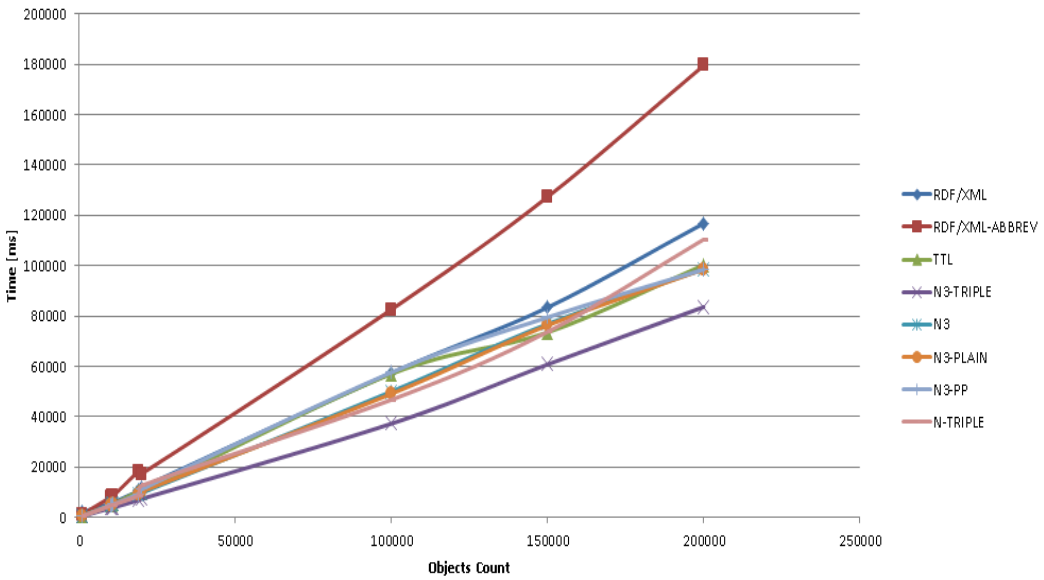

Figure 13.2.1: Time Dependence on the Set of Input Objects

Chapter 14

Evaluation of Ontology

14.1 Prerequisites

Developing ontologies is a significant task across domains. When ontologies are developed it is very difficult to discuss their correctness. We suppose that the designed ontology has to follow several requirements: The output ontology document has to be valid according to the specification defined by W3C. When the ontology is valid it is readable by the common tools (e.g. Protége) used for ontologies modeling. The ontology matching the specification is processable by a common languages for querying of ontologies (e.g. SparQL). The last, but probably the most important requirement is to ensure that the ontology is accepted by community interested in the domain for which the ontology was developed.

We defined three different ways to validate the developed ontology. We presented NIF (see Subsection 6.3.2) as a portal that serves as an online inventory of registered neuroscience data sources. Since the NIF is important framework within the INCF community developed according to INCF recommendations (see Subsection 6.2.1) we choose it as the best choice for validating the ontology we presented. Therefore, NIF was defined, in the goal of this thesis, as the authority that validates our approach.

The validation within NIF ensures that the developed ontology is supposed to express EEG/ERP experiments. Such ontology is ready to be accepted within the neurophysiology community.

Before starting with the registration of the ontology within NIF we needed to check the validity of the ontology document. We compared the document with the W3C specification and used the Protége tool.

14.2 Comparison with Specification

Ontologies expressed by the Semantic Web languages are intended to be read by reasoners. The ontology has to conform the defined syntax in order to should be read by the reasoner. We described the main important requirements that the ontology must satisfy to be valid. This requirements were presented in Definitions 10, 11 and 12. The full specification of all available OWL features is defined by W3C [55].

We took the simple JavaBean from Listing 10.1 that has the serialization expressed in Listing 10.2. Let we discuss partial OWL constructs.

Since we serialize the ontology document into XML-based syntax we have to firstly define the XML definition header;

```
<?xml version="1.0"?>
```

Before we start with description of partial ontology constructs we need to define linked namespaces:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.0="http://thewebsemantic.com/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:j.1="http://cz.zcu.kiv#"
  xml:base="http://data.pojo">
```

An OWL class is syntactically represented as a named instance of `owl:Class` [52]. When we look at our class definition from the following listing we can see `owl:Class` as well. Therefore we can say that we satisfy the syntax of class definition.

```
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment"/>
```

The listing above is correct but does not provide the full description of the class *Eperiment*. OWL provides possibility to add additional components of OWL class [52]. It contains three language constructs for combining class descriptions into class axioms: `rdfs:subClassOf`, `owl:equivalentClass` and `owl:disjointWith`. Syntactically, these three language constructs are properties that the class expresses as a subelement of `owl:Class`. If we compare this definition with listing bellow, we can say that the output is syntactically correct.

```
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment">
  <owl:equivalentClass rdf:resource="http://cz.zcu.kiv#Measurement"/>
</owl:Class>
```

Constructs `owl:equivalentProperty` and `owl:inverseOf` represent relations to other properties [52]. These properties are syntactically defined as build-in properties of `owl:ObjectProperty` or `owl:DataTypeProperty`. The following listing shows the `owl:equivalentProperty` from our ontology.

```
<owl:ObjectProperty rdf:about="http://cz.zcu.kiv#testedSubject">
  <owl:equivalentProperty rdf:resource="http://cz.zcu.kiv#TestedSubject"/>
  <rdfs:domain rdf:resource="http://cz.zcu.kiv#Experiment"/>
</owl:ObjectProperty>
```

Finally, the following listing closes the ontology document.

```
</rdf:RDF>
```

The last OWL construct is the property restriction. Property restrictions are defined in [52] as a special kind of class description. They describe an anonymous class, namely a class of all individuals that satisfy the restriction.

Property restrictions have the following general form:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="(some property)" />
  (precisely one value or cardinality constraint)
</owl:Restriction>
```

The class `owl:Restriction` is defined as a subclass of `owl:Class`. A restriction class should have exactly one triple linking the restriction to a particular property, using the `owl:onProperty` property. The restriction class should also have exactly one triple that represents the value constraint c.q. cardinality constraint on the property under consideration, e.g., that the cardinality of the property is exactly 1.

Property restrictions can be applied both to *Datatype* properties and *Object* properties.

For demonstration we extend listing 10.1 about a cardinality annotation `@MaxCardinality(1)` on `testedSubject` property. See Listing 14.1.

Listing 14.1: Tested Subject With Cardinality

```
@MaxCardinality(1)
private Person testedSubject;
```

The output OWL document is extended as follows:

```
<owl:Class rdf:about="http://cz.zcu.kiv#Experiment">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="http://cz.zcu.kiv#testedSubject"/>
```

```
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <j.0:javaclass>cz.zcu.kiv.Experiment</j.0:javaclass>
  </owl:Class>
```

We compared a partial construct of our ontology generated from JavaBeans build-in within the EEG/ERP Portal. This section verified that the ontology is syntactically correct according to the specification defined by W3C [52]. There are on-line validators enabling to verify a custom ontology according to W3C specification. We selected the validator provided by the Manchester University[1] and used it for verifying the developed ontology. Figure 14.2.1 shows the successful validation result of the developed ontology. It shows correctness of the constructs we discussed above.

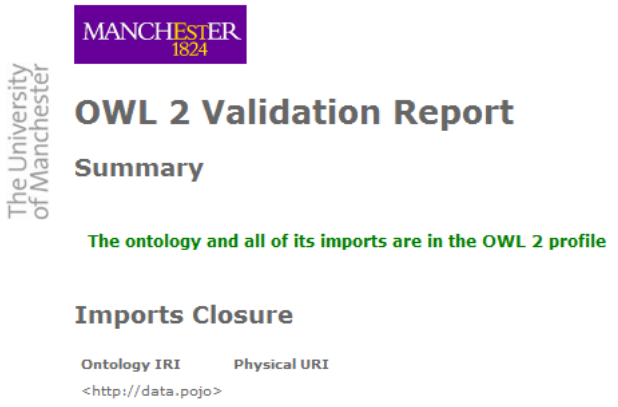

Figure 14.2.1: EEG/ERP Portal Ontology Validation Result

14.3 Validation Using Protége

Since the Protége is probably the most extended tool used for modeling of ontologies we decided to use Protége as a secondary validation tool of the generated ontology. When the ontology is readable by the Protége it can be easy managed, visualized or modified.

Figure 14.3.1 shows the loaded EEG/ERP Portal ontology using Protége. The ontology was successfully loaded so it can be considered as a valid for the Protége tool. Classes derived from the ontology are listed in the left window.

[1] http://owl.cs.manchester.ac.uk/validator/

The right two windows shows the selected class. The class *Experiment* is selected. The bottom right window shows superclasses derived from the *Experiment* class. The visualization of the ontology is shown in Figure 14.3.2. We can see the classes: *Experiment*, *Person*, *Scenario*, etc.; all are inherited from the class *Thing*.

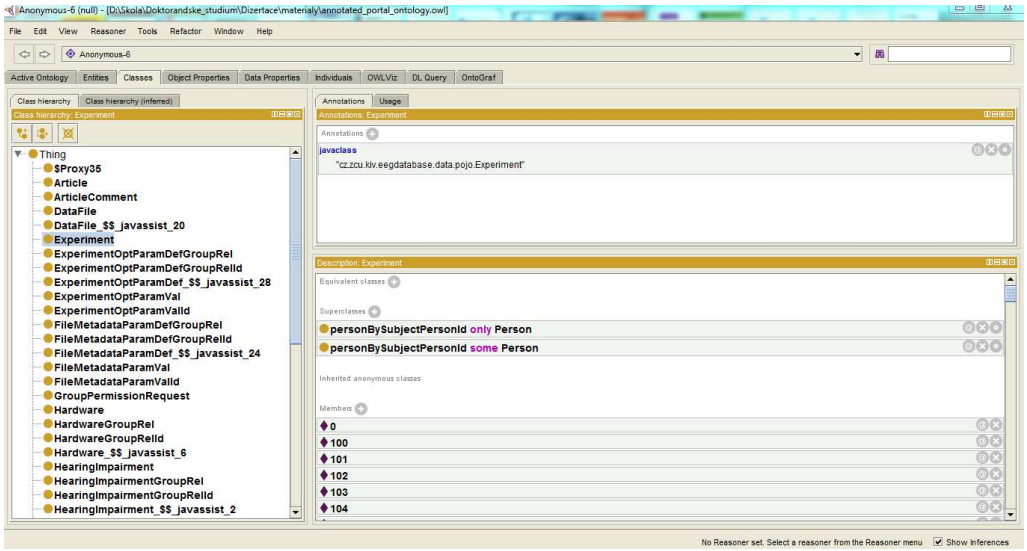

Figure 14.3.1: EEG/ERP Portal Ontology Loaded in Protége

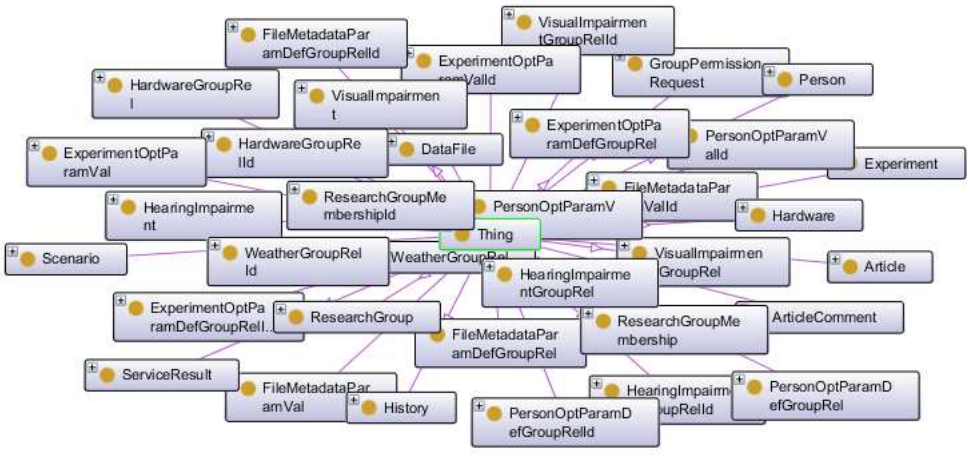

Figure 14.3.2: EEG/ERP Portal Ontology Visualized in Protége

This section proved that the ontology we presented can be processed using Protége. An advanced user can use this tool when he/she wants to adapt this ontology to custom needs, or he/she wants to only visualize the ontology.

The last two sections proved that the ontology we automatically generated from the EEG/ERP Portal using the Semantic Framework is syntactically valid according to the specification defined by W3C. Such ontology is prepared to be read by software agents (ontology reasoners).

14.4 Evaluation Using NIF Portal

14.4.1 Registration Process overview

The registration within NIF takes places at 3 levels described in 6.3.2. Each level makes a neuroscience resource available in different way. The registration takes place gradually from the lowest to the highest level.

When we were registering our resource we had to follow several steps: The first step is registering the EEG/ERP Portal on the first level. Such registration required to provide basic information about the EEG/ERP Portal. When the resource was *curated* (approved by the NIF interoperability team) the WIKI page was assigned to the resource. The *curated* resource is registered within the *NIF registry*. Once our resource was *curated* we generated a sitemap using the NIF build-in tool. Because our data source is a database we created an *Interop File*. This file allows us sharing data using the dynamic search. The last step required creation *Linkout File* that enables to link the resource with papers published in the *PubMed*[2].

14.4.2 Resource Description

NIF uses a proprietary framework *Disco* [51]. This framework consists of several tools and description files for describing resources and for accessing a dynamic context of the registered resource as well. The registered resource is described using two `XML` files: `disco.xml` and `disco.rd.xml`. Both have *RDF* syntax that are computer-processable on the NIF side. The first file provides a basic description of the registered resource as its URL, contact, etc. The second one provides a more detailed de-

[2]PubMed comprises more than 21 million citations for biomedical literature from MEDLINE, life science journals, and online books. See http://www.ncbi.nlm.nih.gov/pubmed/

scription of the source as a description of partial sections of the system, keywords, publication links, etc. These files are located in the root of our EEG/ERP Portal so they can be read by *NIF mediators*. The URLs of this files are: `http://eegdatabase.kiv.zcu.cz/disco.xml` and `http://eegdatabase.kiv.zcu.cz/disco.rd.xml`. When we change the description in these files the NIF registry description will be changed automatically. This approach ensures that we manage the description of our resource locally. The EEG/ERP portal registered as the NIF resource is available at `http://neurolex.org/wiki/Nif-0000-08190`.

14.4.3 Dynamic Content Registration

We prepared two *Disco* files that describe our EEG/ERP Portal in the NIF registry. Such description provides static information about the registered source, but the main contribution, in relation to our EEG/ERP Portal, of the Disco protocol is the possibility to access a dynamic content of the stored experiments. When experimental data and metadata are searchable we can assume that the ontology is valid.

The stored experiments we provided in two different forms. The first form is the ontology in the Semantic Web form that is generated by integrated Semantic Framework (see: Section 10.4). The second form uses *Interoperability XML* file [21].

The *Interoperability XML* is used to describe the structure of metadata instances stored in a plain text file. The plain text file is a dynamic file that consists experimental data/metadata from the EEG/ERP Portal. The XML file is in the project root as well as Disco XML files and is accessed by NIF. The XML description file is available at: `http://eegdatabase.kiv.zcu.cz/eegdatabase.xml`. The NIF reloads it in regular intervals. When the *Interoperability XML* file is reloaded the dynamic plain text file is also refreshed. This approach enables a dynamic access to the dynamic content of the EEG/ERP Portal using the NIF registry. Figure 14.4.1 shows the overview of the NIF registry with the EEG/ERP Portal listed.

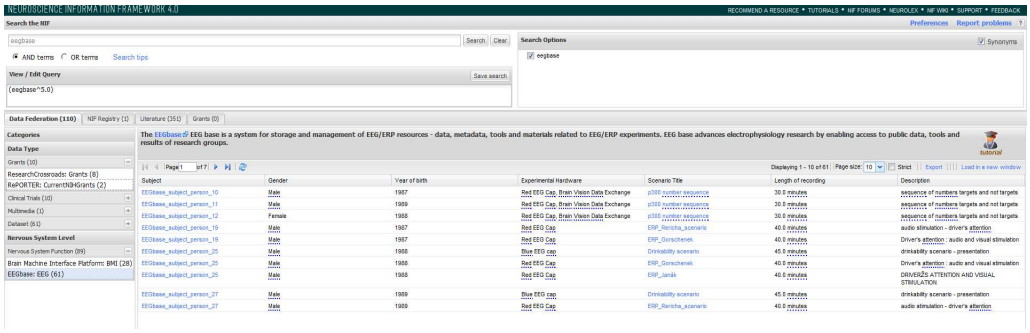

Figure 14.4.1: EEG/ERP Portal within NIF Registry

When experiments stored in the EEG/ERP Portal are searchable through NIF we can infer that the ontology is searchable and valid according to NIF requirements. Since NIF supposes to be an authority for neuroscience resources we can consider a custom developed ontology as an appropriate expression of EEG/ERP experiments. The last step, the registration of the ontology in the Semantic Web form, despite intensive cooperation with the NIF support, is still in progress. We have the OWL output fully prepared but due to unpreparedness of NIF we are not able to finish this step yet. We suppose that the process of the registration of ontologies is still under development. As soon as the NIF side will be prepared we are able to register our OWL immediately. However, we can verify the ontology according to the W3C specification or by using Protége tool.

Chapter 15

Conclusion

The goal of this thesis was to identify difficulties relating to storage of data/metadata from EEG/ERP experiments.

The scientific papers focused on EEG/ERP experiments usually describe designing and creating experimental scenarios or performing experiments. They usually do not solve storage, interchange or description of experimental data and medatata.

The interchange of experimental data relates to the need to provide unified data/metadata formats. The Internet seems to be appropriate to share experimental data. However, due to limits that the current web gradually reaches a parallel web called the Semantic Web is being developed. The current web consists of large collections of data without any classification. The Semantic Web that expresses meaning of data by domain ontologies is supposed to solve the problem of missing semantics of the current web.

Development of specific ontologies is crucial task in the creation of Semantic Web. Ontologies usually serve as recognizable data sources accessible by automatic software readers. However, current software systems are usually based on object-oriented programming languages and they operate over large data collections. The data layer of such systems is usually represented by the set of data objects. These objects are stored within the relational database.

Since fundamental differences between semantics of the object-oriented code and Semantic Web languages exist, it is necessary to ensure a suitable mapping.

Because expressive capabilities of Semantic Web languages are richer than in the case of the object-oriented code we investigated the way to fill these semantic gaps.

In this work we presented an approach based on extending a common JavaBean using JavaAnnotations that adds missing semantics into the plain JavaBean. We presented the defined mapping of Java annotations to the corresponding OWL constructs. We also presented the developed framework where the mapping is implemented. This framework is supposed to be a powerful tool for preparing domain ontologies extracted from object-oriented Java based systems independently on the specific domain needs.

Difficulties relating to description of EEG/ERP experiments were solved by designing the ontology that describes metadata. This ontology was presented to large neuroscience community to accept it as the standard ontology of EEG/ERP experiments.

The EEG/ERP Portal serves the community as the practical tool for storing, managing and interchanging custom experiments. The internal structure of the system is designed to satisfy restrictions given by the ontology. The web based user interface is easy to use.

The developed Semantic Framework was integrated with the EEG/ERP Portal, therefore we are able to transform the portal ontology into the Semantic Web languages (OWL, RDF). This ontology was registered in the NIF Portal. This approach proved the correctness of our solution.

15.1 Current State of Work

The work solves several difficulties we were facing. Firstly, we pointed out the problems related to description of data by suitable metadata and the needs of unified formats to preserve large data sets obtained during EEG/ERP experiments. We solved it by defining a custom ontology based on practical experiences gained during experiments performed in our laboratory, reading scientific papers and books and intensive communication with our collaborators.

The second important task included development of the EEG/ERP Portal. The system is implemented with the emphasis to provide data/metadata in the form prescribed by the defined ontology.

The most important contribution was to infer the transformational mechanism that transforms common JavaBeans into the Semantic Web languages. In addition, the semantic diversity that exists due to different semantic expressivity of the object oriented code and the Semantic Web languages we partly solved. The proposed transformational mechanisms was practically

implemented in the Semantic Framework. This framework is usable as a single library integrable within various Java-based systems. Such libraries can serve in various domains, not only in the domain of EEG/ERP experiments.

The ontology registered in NIF is searchable through the unified interface that enables simple sharing of stored experiments.

15.2 Evaluation of Thesis Goals

Now we summarize the thesis goals appointed in Section 1.3:

1. The first point of this work was the definition an ontology that describes EEG/ERP experiments. The ontology was designed and implemented. This ontology is fully prepared to be discussed within the community. The interested community can use it as the template for the description of custom experiments. The defined ontology is represented using common Semantic Web languages and it can be manipulated using various tools for ontologies modeling.

2. The second point of this work was the representation of the developed ontology using Semantic Web technologies. The developed ontology is represented by Ontology Web Language (OWL). It is fully prepared to be read by ontology reasoners.

3. The third point was investigation of the transformation of an ontology from the systems that are not primarily based on the Semantic Web languages. As the result we proposed and developed a way to transform common data structures into the Semantic web languages. The main contribution of this part of the thesis was extension of current data modeling to solve the semantic gap caused by different expressivity of various data modeling techniques. In addition, the presented system improved mapping in general, it is used across various systems, not only in neuroscience.

4. The developed transformational mechanism and the developed ontology were verified to discuss their validity in the fourth point. We selected NIF. NIF enables registration of domain ontologies in three levels. We successfully registered the ontology almost at the last level. This registration is done by XML-based files. The registration of the ontology in the form of OWL document was not finished yet due to unfinished

implementation of the NIF registry. However, the ontology registered in the present state is searchable (experiments are listed within NIF).

15.3 Future Work

The presented work introduced the complete way from theoretical design of the ontology of EEG/ERP experiments over its practical implementation until its automatic extraction from common object-oriented system and its verification using NIF.

Since a lot of work was done, the range of issues related to expression of knowledge by the Semantic Web and mainly its transformation from common data structures is not possible to cover by this thesis. The presented approach produces the ontology document that is generated in OWL. When we compare the ontology document with specifications defined in Subsection 5.2.2 we can say that it fully satisfies OWL DL but it does not use all constructs of OWL Full. Since OWL Full e.g. does not enforce a strict separation of classes, properties, individuals and data values it is problematic to map object-oriented constructs (where such strictly separation is enforced) to equivalent OWL constructs.

In the future we plan to investigate a strict separation between OWL specifications and clearly define which OWL constructs are possible to express by object-oriented languages and where it is fundamentally impossible.

Since many limitations of OWL exist the extension (called OWL2)[1] has been started developed [58] when this work was written. OWL2 aims is to remove issues of different syntaxes, improve datatype expressivity, provide better organization of imports or remove difficulties with different versions of OWL syntaxes [59].

As the significant next step we plan to start with transformation from the object-oriented code into the OWL2 syntax.

There is also outstanding task related to the technical implementation of the presented solution. From Chapter 13 we suspect possible future performance problems. Currently the EEG/ERP Portal contains tens of experiments but in the future we expect a sharp increase in the number of stored experiments. Since we inferred linear dependence of transformation process in Chapter 13 the direct online serialization could be time consuming. Therefore we plan to store partial models formed from the transformation process in the

[1] the current one was renamed to OWL1.

database. The stored model will be gradually supplemented by experiments being added. The serialized output documents will be stored also in the database. When a document request occurs it will be retrieved from the database and returned to the user.

Together with the continuing development of NIF we will work on the registration of the presented OWL document into the NIF registry. It should replace the current XML-based approach.

Bibliography

[1] Rémond, A., Handbook of elektroencephalography and clinical neurophysiology: Graphic and magnetic-tape recording of bioelectrical phenomena, Amsterdam, 1976, ISBN: 978-0444801258

[2] Handy, T. C., Event-related potentials, A Bradford Book; 1 edition, 2004, ISBN: 978-0262083331

[3] Johnson, G., Understanding how to brain works. Traumatic Brain Injury survival guide, http://www.tbiguide.com/howbrainworks.html, Online, 2010

[4] Luck, S. J., An Introduction to the Event-Related Potential Technique (Cognitive Neuroscience), The MIT Press, August 2005, ISBN: 978-0262621960

[5] Dean, A., Voss, D., Design and Analysis of Experiments, Springer Verlag, New York, USA, 1999, ISBN: 978-0-387-98561-9

[6] Polich, J., Kok, A., Cognitive and biological determinants of P300: an integrative review, Biological Psychology, Elsevier Science 41, pp. 103 - 146, 1995

[7] Kemp, B., European Data Format, http://www.edfplus.info/index.html, Online, 2010

[8] Kemp, B., Värri, A., Rosa, A. C., Nielsen, K. D., Gade, J., A simple format for exchange of digitized polygraphic recordings. Clinical Neurophysiology 82, pp. 391 - 393, 1992

[9] Walter Graphtek, http://www.walter-graphtek.com/, Online, 2010

[10] Natus, http://www.natus.com/index.cfm?page=company_1&crid=139, Online, 2010

[11] Brainlab, http://www.brainlab.be/, Online, 2010

[12] OpenXDF Consortium, OpenXDF, http://www.openxdf.org/, Online, 2010

[13] Brain Product, http://www.brainproducts.com/, Online, 2010

[14] Holmes, G., Donkin, A., Witten, I.H., WEKA: a machine learning workbench, Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on, IEEE, pp. 357-361, 1994, ISBN: 0-7803-2404-8

[15] Pelt, J. van, Horn, J. van, Workshop report, 1st INCF Workshop on Sustainability of Neuroscience Databases, Stockholm, 2007

[16] Kötter, R., Neuroscience Databases: A Practical Guide, Kluwer Academic Publishers, USA, 2003, ISBN: 1-4020-7165-5

[17] CARMEN Portal, http://www.carmen.org.uk/, Online, 2010

[18] INCF Japan node, http://www.neuroinf.jp/, Online, 2010

[19] Gupta, A., Bug, W., Marenco, L., Qian, X., Condit, Ch., Rangarajan, A., Müller, H. M., Miller, P. L., Sanders, B., Grethe, J. S., Astakhov, V., Shepherd, G., Sternberg, P. W., Martone, M. E., Federated Access to Heterogeneous Information Resources in the Neuroscience Information Framework (NIF), Neuroinformatics 6, Springer, pp. 205–217, 2008

[20] The NIF DISCO Framework: Facilitating Automated Integration of Neuroscience Content on the Web, Neuroinformatics 8, Springer, pp. 101–112, 2010

[21] NCBO BioPortal, http://bioportal.bioontology.org/ontologies/40510, Online, 2010

[22] Mouček, R., Mautner, P., Driver attention while double stress - EEG/ERP experiment (Pozornost řidiče při dvojí zátěži – EEG/ERP experiment in Czech), Kognice a umelý život IX, Opava 2009, ISBN: 978-80-7248-516-1

[23] Antoniou, G., Harmelen, F. van, A Semantic Web Primer, The MIT Press, April 2004, ISBN: 0-262-01210-3

[24] Berners-Lee, T., Hendler, J., Lassila, O., The Semantic Web, Scientific American Magazine, May 2001

[25] Obitko, M., Translations between Ontologies in Multi-Agent Systems, Ph.D. dissertation thesis, CTU, Prague, 2007

[26] Yanhui Lv Ma, Z. M., Transformation of relational model to RDF model. Systems, Man and Cybernetics, 2008, ISBN: 978-1-4244-2383-5

[27] Guimaraes, J. de O., The object oriented model and its advantages, ACM SIGPLAN OOPS Messenger, January 1995, pp. 40 - 49, ISSN: 1055-6400

[28] Kalyanpur, A., Pastor, D. J., Padget, J. A., Automatic Mapping of OWL Ontologies into Java, Software Engeering and Knowledge Engeering, pp. 98 - 103, June 2004

[29] LePendu, P., Ontology based Relational Databases, University of Oregon, 2007

[30] Dou, D., LePendu, P., Ontology-based integration for relational databases, In ACM Symposium on Apllied Computing (SAC) pp. 461-466, 2006

[31] Dou, D., LePendu, P., Kim, S. and Qi, P., Integrating databases into the semantic web through an ontology-based framework, Proceedings of the 22nd International Conference on Data Engineering Workshops, pp. 54-63, 2006

[32] Koide, S., Aasman, J., Haflich, S., OWL vs. Object Oriented Programming. In International Workshop on Semantic Web Enabled Software Engineering (SWESE), 2005

[33] Jena Framework, http://jena.sourceforge.net/, Online, 2010

[34] Prud'hommeaux, E., Seaborne, A., SPARQL Query Language for RDF. W3C Recommendation, http://www.w3.org/TR/2005/WD-rdf-sparql-query- 20050217/, February 2005.

[35] Steer, D., SquirrelRDF, http://jena.sourceforge.net/SquirrelRDF/, Online, 2010

[36] The D2RQ Plattform - Treating Non-RDF Databases as Virtual RDF Graphs, http://www4.wiwiss.fu-berlin.de/bizer/d2rq/, Online, 2010

[37] Sesame sematic web toolkit, http://semanticweb.org/wiki/Sesame, Online, 2010

[38] Švihla, M., Transforming Relational Data into Ontology Based RDF data, Thesis, CTU, Prague, 2007

[39] Sommer, https://sommer.dev.java.net/sommer/index.html, Online, 2010

[40] Java2OWL-S, http://www.daml.org/2003/10/java2owl/, Online, 2010

[41] JenaBean, http://code.google.com/p/jenabean/, Online, 2010

[42] The OWL Api, http://owlapi.sourceforge.net/, Online, 2010

[43] Bell, D., UML basics: An introduction to the Unified Modeling Language, http://www.ibm.com/developerworks/rational/library/769.html, Online, 2011

[44] OMG Consorcium, Ontology Definition Metamodel, http://www.omg.org/spec/ODM/1.0/PDF/, Online, 2011

[45] OMG Consorcium, OMG Object Constraint Language (OCL), http://www.omg.org/spec/OCL/2.3.1/, Online, 2011

[46] Oren, E., Delbru, R., Gerke, S., Haller, A. and Decker S., ActiveRDF: object-oriented semantic web programming, In Proceedings of the 16th international conference on World Wide Web, pp. 817-824, 2007

[47] Po-Huan, Ch., Chi-Chuan, L., Kuo-Ming, Ch., Integrationg Semanic Web and Object-Oriented Programming for Cooperative Desig, In Journal of University Computer Science,vol. 15, no. 9, 2009

[48] Liu, F., Wang, J., Dillon, S. T., Web Information Representation, Extraction and Reasoning based on Existing Programming Technology, In Computational Inteligence 37, pp. 147-168, 2007

[49] Hitchcock, S., The Open Journal Project, What is a Bean, http://journals.ecs.soton.ac.uk/java/tutorial/beans/whatis/simple-definition.html, Online, 1998

[50] Pergler, J., Database of ERP experiments - business and presentation layer (Databáze ERP experimentů - aplikační a prezentační vrstva), Thesis (in Czech), University of West Bohemia, Pilsen, 2009

[51] Marenco, L., Wang, R., Shepherd, M. G., Miller, P. L., The NIF DISCO Framework: Facilitating Automated Integration of Neuroscience Content on the Web, In Journal Neuroinformatics no. 8, 2010

[52] W3C OWL Working Group, OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/, W3C Recommendation, Online, 2011

[53] W3C OWL Working Group, RDF/XML Syntax Specification (Revised), http://www.w3.org/TR/rdf-syntax-grammar/, W3C Recommendation, Online 2012

[54] Roebuck, K., Object-relational mapping (Orm): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors, Tebbo, June 2011, ISBN: 1743044755

[55] W3C OWL Working Group, OWL Web Ontology Language, W3C Recommendation 10 February 2004, Overview http://www.w3.org/TR/owl-features/, Online, 2012

[56] Teorey, T. J., Lightstone, S. S., Nadeau, T., Jagadish, H.V., Database Modeling and Design, Fifth Edition, Morgan Kaufmann, February 2004, ISBN: 0123820200

[57] Ciniburk, J., Mouček, R., Mautner, P., Řondík, T., ERP components detection using wavelet transform and matching pursuit algorithm, DCII, Prague 2010

[58] W3C OWL Working Group, OWL 2 Web Ontology Language Document Overview, http://www.w3.org/TR/owl-overview/, W3C Recommendation, October 2009

[59] Graua, B. C., Horrocksa, I., Motika, B., Parsiab, B., Patel-Schneiderc, P., Sattlerb, U., OWL 2: The next step for OWL, Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier, Volume 6, Issue 4, pp. 309-322, November 2008

Appendix A

Author's Publications

The following papers were published in conference proceedings:

1. Ježek, P. **Hromadné úložište EEG/ERP záznamů**. Informatika v škole a v praxi. Ružomberok, 2008, pp. 158 - 161, ISBN 978-80-8084-362-5
2. Mouček, R., Ježek, P. **EEG/ERP experiments - Data and Metadata Structures**. Frontiers in Neuroinformatics. Stockholm, 2008, pp. 123
3. Ježek, P., Mouček, R. **Database for EEG/ERP Experiments**. 2^{nd} INCF Congress of Neuroinformatics. Frontiers in Neuroinformatics, Pilsen, 2009, pp. 141
4. Ježek, P. **Úložište dat a metadat EEG/ERP experimentů**. Kognice a umelý život IX. Opava, 2009, pp. 125 - 129, ISBN: 978-80-7248-516-1
5. Ježek, P., Mouček, R. **Database of EEG/ERP Experiments**. Third International Conference on Health Informatics. Valencia, Spain, 2010, pp. 222 - 227, ISBN: 978-989-674-016-0
6. Mouček, R., Ježek, P. **System for Storage and Management of EEG/ERP Experiments - Generation of Ontology**. International Conference on Enterprise Information Systems. Funchal, Madeira - Portugal, 2010, pp. 415 - 420, ISBN: 978-989-8425-04-1
7. Ježek, P., Papež, V. **Systém pro správu ERP experimentů**. Kognice a umělý život X. Slezská univerzita v Opavě, 2010, pp. 177 - 180, ISBN: 978-80-7248-589-5

8. Mouček, R., Ježek, P., Papež, V. **Prostředky sémantického webu v oblasti EEG a evokovaných potenciálů**. Kognice a umělý život X. Slezská univerzita v Opavě, 2010, pp. 259 - 262, ISBN: 978-80-7248-589-5

9. Papež, V., Ježek, P. **Neuroinformatická databáze a sémantický web**. Kognice a umělý život X. Slezská univerzita v Opavě, 2010, pp. 269 - 273, ISBN: 978-80-7248-589-5

10. Ježek, P., Mouček, R. **EEG/ERP Portal - Semantic Web Extension,Generating Ontology from Object Oriented Model**. Second Global Congress on Intelligent Systems. Wuhan, China, 2010, pp. 392 - 395, ISBN: 978-1-4244-9247-3

11. Ježek, P., Mouček, R. **System for Storage EEG/ERP Data and Metadata**. Frontiers in Neuroinformatics. 3^{rd} INCF Congress of Neuroinformatics. Kobe, Japan, 2010, pp. 150

12. Mouček, R., Ježek, P., Papež, V. **Deriving Semantic web Structures from EEG/ERP Data Source**. Frontiers in Neuroinformatics, 3^{rd} INCF Congress of Neuroinformatics. Kobe, Japan, 2010, pp. 45

13. Ježek, P., Mouček, R. **Integration of Signal Processing Methods into EEG/ERP System**. Healthinf - International conference on health informatics. Rome, Italy. 2011. pp. 563 - 566, ISBN: 978-989-8425-34-8

14. Mouček, R., Ježek, P., Václav, P. **Semantic Web Technologies in EEG/ERP Domain - Software Solution**. Healthinf - International conference on health informatics. Rome, Italy, 2011, pp. 618 - 621, ISBN: 978-989-8425-34-8

15. Ježek, P., Mouček, R. **Transformation of Object-Oriented Code Into Semantic Web Using Java Annotations**. ICEIS - 13^{th} International Conference on Enterprise Information Systems. Beijing, China, 2011, pp. 1 - 4, ISBN: 978-989-8425-65-2

16. Mouček, R., Ježek, P. **Overview of Neuroinformatics Infrastructure in Pilsen, CZ**. Frontiers in Neuroinformatics. 4^{th} INCF Congress of Neuroinformatics. Boston, USA, 2010, pp. 310

17. Ježek, P., Mouček, R. **EEG/ERP portal - Extending Object Oriented Code for the Missing Semantics Using Java Annotations**. Frontiers in Neuroinformatics. 4^{th} INCF Congress of Neuroinformatics. Boston, USA, 2010, pp. 320

18. Mouček, R., Jaroš, P., Ježek, P., Papež, V. **Software infrastructure for EEG/ERP research**. International Conference on knowledge Enginering and Ontology Development. Setúbal: SciTePress, 2011, pp. 478 - 481, ISBN: 978-989-8425-80-5

19. Ježek, P., Mouček, R. **Semantic Web in EEG/ERP Portal**. 4^{th} International Conference on Biomedical Engineering and Informatics. New York: IEEE. 2011, pp. 2076 - 2080. ISBN: 978-1-4244-9350-0

20. Ježek, P., Mouček, R. **Semantic Web in EEG/ERP Portal - Extending of Data Layer using Java Annotations**. HEALTHINF - International Conference on Health Informatics. Vilamoura, Algarve, Portugal, 2012 pp. 350 - 353, ISBN: 978-989-8425-88-1