Exploring Object oriented language and migration with Zend

ALEN ŠIMEC
Polytechnic of Zagreb, Vrbik 8, 10 000 Zagreb, CROATIA
alen@tvz.hr, www.tvz.hr

JOSIP PERIĆ
Polytechnic of Zagreb, Vrbik 8, 10 000 Zagreb, CROATIA
jperic@tvz.hr, www.tvz.hr

LIDIJA TEPEŠ GOLUBIĆ
Polytechnic of Zagreb, Vrbik 8, 10 000 Zagreb, CROATIA
ltepes2@tvz.hr, www.tvz.hr

Abstract: - This paper refers to the research of novelties and differences in the newest and currently active versions of PHP programming language. It gives some answers to the questions of usefulness and feasibility of changing the version used in a certain web project. The attention has been mostly focused on describing functionality with occasional evidence in the shape of a code.

Key - Words: PHP, engine, memory, exception, class, object, generator

1 Introduction

Rapid expansion of web technologies has required a development of new approaches to data management. The consequence of the demand of interactive and dynamic systems is the emergence of a script language also known by the acronym PHP (Hypertext Preprocessor). Before PHP started being used, PHP web sites had been mostly static with the capacity limited to roughly show information in the form of a text of images. Integration of PHP into web spaces has enabled interaction with the users as well as processing and application of data from user sources.

Even though it is being performed on the server side, it has been imagined as a communication connection between the client and the server in client-server architecture. It has the characteristics of the code openness and free availability. It has been written in C programming language and it acts as a database manager.

PHP has so far been published in six versions with various editions inside of versions. The last available version is PHP7, and this paper is going to be based on the analysis of the newest and currently existent versions, that is basic features that determine them and separate them from the previous editions and mutually. The goal of this paper is to highlight positive and useful sides of the latest versions of this programming language by applying a mostly descriptive method with occasional use of induction, experiments, measurements and analysis.

2 Zend Engine System Overview

Metadata is the most important part of the information management infrastructure in data repositories. The metadata is defined according to the standards, specifications or their application profiles.

The fifth version of PHP programming language is based on the exchange of mechanisms for script implementation. The new version of Zend Engine with a secondary suffix has, among other things, enabled the code writing in a more automated way. It represents the language core in a way that it sets predefined functions for the development of PHP code. In addition, it provides the language with standard services such as resource and memory management. A novelty in this field is the addition of expandable objective models together with improvement of basic performances.

Expandable structures also pertain to writing function arguments such as defining information about the minimal and maximal number of parameters and structures of transferred parameters. Object classes can be implicitly demanded when calling which ensures the reception of requested data. Otherwise, fatal error occurs.

```
<?php
function transfer(Player $player, Club $club){
echo "transferring " . $player->name . " to " . $club-
>name ;
}
```

Code 1 –Defining function arguments

According to the stated function declaration, the first entry parameter has to be a class Player object, and the second one class Club. If a null default value is set for an argument, then it is possible for it to be forwarded. This way of defining arguments is being called type hinting.

Additional code implementations, such as parameter information, are being demanded from the author when creating extensions. The header of the wanted structure can be defined by two forms:

- ZEND_BEGIN_ARG_INFO()
- ZEND_BEGIN_ARG_INFO_EX()

The difference lies in the fact that the second performance enables restrictions in the number of mandatory parameters. There is a similar situation when defining the parameters themselves where the following functions are being differentiated:

- ZEND_ARG_INFO()
- ZEND_ARG_OBJ_INFO()

Zend_arg_obj_info() Function specifies the parameter by marking the parameter class and the possibility of setting NULL values.

PHP5 in this way frees the author of checking the number of forwarded parameters by placing the load on the side of Zend Engine.

Zend Engine changes have resulted in enormous increase in speed, performances and memory consumption. For example, a website launched on PHP7 can reach a doubled number of performed requests per second during a half as long testing time in comparison to the same one launched on PHP5. The stated statistical values of handling requests are the best way of showing progress in the newest PHP version. Despite the obviously positive characteristics, it is necessary to carefully approach the transition to PHP7 due to the changes in functions and extensions, and especially in specific cases of removing those. Moreover, many services still do not support the changes made.

3 PHP scripting language and data objects

The next influential extension that is being introduced into the language are PDO objects that enable consistency of application programming interface between data base extensions. Such objects simplify the creation of applications that are mostly dependent on data base access and managing the saved data. By using the advantages built into the fifth language version, the script performance has been overridden by the original C code in a way that the parts specific for the base have been separated into a special extension.

The most remarkable benefit of using PDO objects is the possibility of reusing. Beside reducing the number of code lines, such implementation simplifies the change of technologies provided that the substitute access also supports the object-oriented paradigm.

For performing tasks of the interface through which the communication with the bases is being performed, has been structured with two key classes: PDO that is being used for establishing communication and direct communication and PDOStatement that refers to the request and the result in the opposite direction. [4]

From the existing methods related to PDOStatement it is necessary to emphasize:

- *lastInsertId()* –for the return value it gets the last line inserted into the base
- *columnCount()* –it restores the number of columns in the set of results
- *rowCount()* –it restores the number of lines in the set of results
- *fetch()* –it restores the next line from the set of results
- *fetchAll()* –it restores the field that contains all the lines of the set of results
- *fetchObject()* –it fetches the next line in the form of an object
- *exec()* –it is being used for adding, changing and deleting lines

Beside the above stated functions, functions such as prepare() and bindParam() that particularly emphasize the multiple code usability are often being used. For example, the prepare() method is being oriented at the use of reserved places in a way that the request is being defined one time, and then later used multiple times.

```
$query= $db->prepare("INSERT INTO players(name,
  position) VALUES (:name,
:position) ");
$query->execute(array(':name'=>'John',
  ':position'=>'Striker'));
```

Fig. 2 – Use of prepare() method

In this way SQL attacks are being automatically disabled. In the case of unnamed reserved places a

question-mark symbol is being placed where the value parameter should be.

Often in combination with prepare() function come bindValue() and bindParam(). Simplified request creation represents a visible advantage of the stated structure. The difference is being noted when giving data to the reserved places. In the case of merging of values it is about setting variable values to a placeholder, while the merging of parameters refers to the variable that is being transferred through a reference. A common feature of both functions is the usage prior to performed requests.

```
$query= $db->prepare("INSERT INTO players(name, position) VALUES (?, ?) ");
$player = array("John","Striker");
$query->bindValue(1, $player[0]);
$query-bindValue(2, $player[1]);
$query->execute();
```

Fig. 3 – Use of bindValue() method

```
$query= $db->prepare("INSERT INTO players(name, position) VALUES (?, ?) ");
$query->bindParam(1, $name);
$query->bindParam(2, $position);
$name = "John";
$position = "Striker";
$query->execute();
```

Fig. 3 – Use of bindParam() method

In the documentation of PHP programming language it has been specifically stated that the actions on the database cannot be performed by using only PDO extensions and that it si necessary to use a specific PDO driver in order to access the database server.

With the advent of Semantic Web, ontologies are gaining importance mainly due to availability of formal ontology languages. These standardization efforts promote several notable uses of ontologies like assisting in communication between people, achieving interoperability (communication) among heterogeneous software systems and improving the design and quality of software systems. One of the most prominent applications is in the domain of semantic interoperability. While pure semantics concerns the study of mean-ings, semantic elevation means to achieve semantic interoperability and can be considered as a subset of information integration (including data access, aggregation, correlation and transformation). Semantic elevation of proposed matching and merging framework represents one major step towards this end.

3.1 Exceptions handling

Changes in exceptions handling in different versions of PHP5.x, one of the modules that has undergone a complete reconstruction is error handling. To be precise, exceptions handling has relieved the developer from the need of doing an additional check of return value of each function. Similarly to the previously mentioned separated database performance, exceptions and error handling has also been separated from the software logic.

The structure and the task of exceptions handling is almost identical to the ones in programming languages such as Java and C++. If the catching of exception is not being defined, an error of uncaught error occurs. [5]

```
<?php
function provjeraSume($br1, $br2) {
if(($br1+$br2)<10){
throw new Exception("Suma mora biti veća ili jednaka 10");
}
return true;
}
try{
provjeraSume(10,2);
echo 'Suma je ispravna';
}
catch(Exception $ex){
echo $ex->getMessage();
}
?>
```

Fig. 4 – Exceptions handling

Taking into consideration the provjeraSume() function (code 5), the exception will occur if numbers whose sum is smaller than 10 are being sent as parameters. Such situation development shall result in a message “Suma mora biti veća ili jednaka 10”. In the final finishing of the fifth version of PHP, finally block has been added in case that the unexpected exception occurs.

According to research, exceptions handling has still not found wide application in PHP, but the peak of such approach is yet to come.

Changes in exceptions handling in versions PHP7.x, soon after the exceptions have been introduced into the programming language their handling has been improved. When developing a new handling system, the focus was on fatal errors that have been stopping the execution of script in the previous version. PHP7 enables the catching of such exceptions, excluding uncatchable ones, such as lack of memory. Moreover, any kind of uncaught exception shall continue to cause the interruption in script performance. In order to prevent that the existing PHP5.x code catches

exceptions that in any case crash its performance, fatal errors in the new version do not inherit class Exception. As a consequence, there is a new class Error from which instances of manageable fatal exceptions are being created. In this way changes in the structure and hierarchy of exceptions themselves are being caused. In order for the hierarchy not to be separated, a new interface, Throwable, has been written that is being implemented by both branches.

Just like any other exception, the exception of Error nature can be caught and controlled land it allows for the finally block to be executed. With this new hierarchy the root interface can retrieve any exception instance. Class Throwable defined within the catch block can catch objects of both Exception and Error types. Even though such structure enables reaction to various unpredicted situations, the practice recommends use of specific classes for individual exceptions handling.

```
try {
…
} catch (Throwable $ex) {
…
}
```

Fig. 5 – Class Throwable

Beside the above mentioned classes, there are also classes that do not have the possibility of implementing the Throwable interface. We are talking about user created classes. An important thing about exceptions handling is that exceptions contain information about the position on which the object has been created. Contradiction occurs in the fact that user objects do not include automatic parameters for saving such information.

However, there is an alternative way for the implemented class to act according to some rules of Throwable interface. The implemented class primarily inherits classes Exception or Error and then it extends through the interface inherited by Throwable.

```
interface PracticeThrowable extends Throwable{}
class PracticeException extends Error implements
PracticeThrowable{}
throw new PracticeException();
```

Fig. 6 –Creating a class for fatal error handling

The base class Error is divided into specific subclasses. The first one in alphabetical order, ArithmeticError refers to arithmetic errors such as moving bits by a negative number or incorrect use of intdiv() function that divides two numbers. The specific arithmetic error that needs to be mentioned in this context is division by zero for which there is a special implementation of DivisionByZeroError class.

```
$value = 1 << -1; // it returns the arithmetic error due to
the negative moving of bits
intdiv(PHP_INT_MIN, -1); // it returns the arithmetic
error
$value = 1 % 0; // it returns DivisionByZeroError
because it is being divided by zero
```

Fig. 7 – Fatal errors

It is necessary to mention that division by zero with the operator (/) results only with a warning, and the result is being evaluated with a value NaN (not a number).

AssertionError occurs in cases when the conditions set by assert() method have not been met. Preconditions for the use of function itself also need to be set. The above mentioned preconditions refer to the initial program settings. It is necessary to define the settings in the following way:

```
ini_set('zend.assertions',1);
ini_set('assert.exception',1);
```

Fig. 8 –Initial values setup

It is also possible to change identical actions manually, by making changes in the configuration of php.ini data file. Zend.assertion variable can be found with value -1 which implies that assertion code shall not be generated. In the case of value 0, the code has been generated but it shall not be executed in the executive surroundings. In accordance with the code above(kodx.x), it is necessary to adjoin value 1 to the variable in order to fully use the offered functionalities. The same change needs to be made to Assert.exception variable which, with the default value 0, shows only simple warnings when error occurs.

Assert structure is generally used to test a certain code segment and to compare the results with the expectations.

```
try{
   $var = 1;
   assert($var > 2, "The value is less than two");
}
catch(AssertionError $ex)
{
   echo $ex->getMessage();
}
```

Fig. 9 – Assert structure

Provided that the settings have been properly set, the following code segment shall return an error with a message: “The value is less than two.”

In the previous hierarchy errors related to variable types, code TypeError, have also been highlighted. This subclass throws the exception only in situations when the given function parameters or the return value do not correspond to their references in the method definition.

```
function operate(int $n1, int $n2)
{
   return $n1 + $n2;
}
try{
   $result = operate( 'first', 'second' );
}
catch (TypeError $ex){
    echo $ex->getMessage();
}
```

Code 10 –TypeErrora overview

Considering that the function clearly asks for int value, sending strings shall cause an error with a clear message that parameters need to be real numbers. A question can be sensed about what shall happen if numeral strings occur instead of textual string. Add operation shall then be performed, if the command „declare(strict_types=1);“ is run above the existing code.

ParseError also belongs to fatal errors and they refer to errors in syntax in data files included in the code. In the second scenario, it can be an error in syntax within the call eval() function. In the previous version of PHP, such errors would cause the program crash. Advantages of introducing this subclass can be seen when sending relevant information in order to check their accuracy.

```
//Prva datoteka ("Prva.php")
<?php
$var='provjera'
//Druga datoteka
<?php
try {
   require "Prva.php";
}
catch(ParseError $ex){
   echo $ex->getMessage();
}
```

Fig. 11 –ParseErrora overview

The error lies in the included data file (Prva.php) because the order of allocating a value to the variable has not been concluded with the „ ; “ symbol.

```
try{
   eval("vr_dump(2);");
}
```

Fig. 12 – Example of ParseError error

Considering there is an error in syntax within the evaluation function, ParseError occurs.

In exceptions handling it is not sufficient to know the structure of code writing, but it is also necessary to have strong theoretical background as well as good organization. The programmer needs to use the instance of a correct class that inherits Error or Exception class in certain surroundings. Error is basically used for code problems that require the programmer’s attention. Errors caused in the engine of PHP belong to this group. Objects of Exception class are being mostly used in more harmless conditions where in parallel with resolving the error another action can be performed and the program performance can be continued.

Accordingly, the practice suggests that catching the Error object should be minimized, that is, realized only in critical conditions such as logging onto the system, showing the critical message or performing the crucial clearing. If the need for the program to support the previous and the new version of PHP exists, the code adjusts in a way that it especially catches Throwable and Exception exceptions. The former one would ensure the correct program reaction in PHP 7.x versions, while the latter refers to PHP 5.x versions. Root changes in exceptions handling require the programmer’s constant attention due to frequent neglect of fatal errors. It is being suggested to the beginners not to pair the performances of catching various performances, but to directly use the Throwable class. Another innovation in this area of PHP7 is the multiple catching of exceptions in one order via the operator „ | “ by which the object types that the catch block catches are being separated.

3.2 Generators in handling with scripting language

Generators refer to the syntax similar to the functions with iterators, the difference being that the generators return the optimal number of values instead of return value. Beside the simplicity of writing and reducing the number of code lines, the main advantage of the generators is in the decline of memory utilization. The excess of free memory enables the hardware to serve even more requests through free resources. The reason of such effect is to avoid creation of arrays in memory so that the object of Generator class is being forwarded. Moreover, that object implements the interface iterator that enables the generator state manipulations functions.

```
<?php
$pocetak=microtime(true);
$polje = array();
$rezultat = '';
for($broj=1;$broj<2000000;$broj++)
{
  $array[]=$broj*0.456;
}
foreach($polje as $obj)
```

```
{
  $obj+=56.189;
  $rezultat .= $obj;
}
$kraj=microtime(true);
$vrijeme1= bcsub($kraj, $pocetak, 4);
$mem= memory_get_peak_usage(true);
$pocetak=microtime(true);
$rezultat = '';
function gen(){
        for($broj=1;$broj<2000000;$broj++){
                yield $broj*0.456;
        }}
foreach(gen() as $obj)
{ $obj+=56.189;
  $rezultat .= $obj;
}
$kraj=microtime(true);
$vrijeme2=bcsub($kraj, $pocetak, 4);
$mem1=memory_get_peak_usage(true);
$mem2=$mem1-$mem;
?>
```

Fig. 13 –Utilization of generators and testing

The key word in generator function is yield. Or better to say, by writing the aforementioned command, regular function turns into the generator one.

By starting the previous code (code 6) the effect of the generator on execution time and used memory can be compared. By adding the appropriate HTML code, the requested values are being shown in a table.

	With Generator	Without Generator
Time	1,3264[s]	0,0979[s]
Memory	20971520[B]	54525952[B]

Table 1 - Details of generator testing

The saving of memory space with the use of a generator with accompanying negative time consequences can be seen from the above table. Accordingly, the usefulness of generator functions can be discussed in appropriate conditions. The main of the favorable conditions is the need to reduce the memory load. The most common application can be found in the functions that implement the pass through a large list of numbers.

The key rule of "collecting rubbish" in PHP programming language is that the container cannot be released if the number of references on it is larger than zero. Accordingly, it is possible to determine which parts are surplus by reducing the number of references by one and by that check which containers do not have the reference number in the set of integer numbers. Considering that the goal is to reduce the unnecessary starting of cycles, a temporary container (storage) for saving all containers is being organized. Only when the container is filled, the cycle begins for all the containers inside the container (storage). The storage is being filled even in the situation when the mechanism for rubbish collection is shut down. However, if the storage fills up then, the next records shall not be registered. The unsaved records directly cause memory leakage.

The user himself can initiate the process of collecting rubbish by using the gc_collect_cycles() function with a return value of the number of collected cycles. The forced call of the stated function is being suggested immediately before the "rubbish collection" mechanism shuts down because by this the risk of memory leakage is reduced.

Starting the mechanism causes two obvious consequences:

- reduction of memory consumption
- slowing performance

Even though contradictory consequences are being shown, the work results are positive, especially for scripts that are being started for a longer period. The reason for the successful impact lies in the fact that the reduction of memory use is many times bigger than reduction of performance.

The idea of the makers of new version is to improve the elements referring to flexibility, elegance and code quantity. The necessity to discard limitations in the listed features made them do a complete revision and alteration of codes. Some changes are entirely new elements in the field of this language, while the others are just mostly positive upgrades.

Iterators, structures that enable the for-each loop to pass through various kinds of data, such as lists, set of results and even entire documents belong to innovations. Related iteration classes are inherited for the purpose mess reduction and code shortening.

Regarding data exchange, a step forward has been accomplished by compiling JSON extension within PHP programming language.

Besides the iterators, the namespace support has been established to be able to manage the code understanding in an easier way. By marking all the methods of some library with the name of the namespace, functions are being determined and reckless errors occur less frequently. Certain functions are being unambiguously determined with a key command namespace.

Goto command that transfers focus from one to some distant code line has been introduced. It represents a contrast to the classic return of function value. A mark that identifies the place of further code execution is being entered after the command.

The library distribution has been improved by activating PHAR, package for archiving and unpacking a large number or data files. In this way the goal of faster performing of applications through FTP protocol has been achieved.

Beside the improvement in the field of MySQL and object-oriented programming, this version removes some errors from earlier versions related to XML (eXtensible Markup Language).

Standardization has been described in only one XML library and the cooperation is finally getting characteristics of a united whole. Proper and necessary XML tools that are entirely in compliance with W3 specifications have been provided to the user, that is, to the programmer.

Changes and reading of XML documents lose the complexity, while the data processing saves money and spent resources.

The big news in this field is operator modifications. So called "space number" operators are being introduced. This name was given out of a practical reason of similarities between those two terms. It is about a triple comparison of variables in the sense that it returns the value "-1" in case that the right variable is bigger, "1" as a consequence of the bigger value of the left variable and "0" if the given values are equal.

Operator	Equality
$v1 < $v2	($v1 <=> $v2) === -1
$v1 <= $v2	($v1 <=> $v2) === -1 \|\| ($v1 <=> $v2) === 0
$v1 == $v2	($v1 <=> $v2) === 0
$v1 != $v2	($v1 <=> $v2) !== 0
$v1 >= $v2	($v1 <=> $v2) === 1 \|\| ($v1 <=> $v2) === 0
$v1 > $v2	($v1 <=> $v2) === 1

Table 2 - Overview of new operators

In order to save time and code lines, a new structure has been created for checking existence of non-NULL values of some variable or return value.

```
$country = $_GET['country'] ?? 'unknown';
```

Code 14 – „??“ operator

Previous excerpt with "??" operator checks if the value from country variable is being fetched and it examines its stability and, in case of a negative results, it responds with a default value.

Regarding functions, the programmers are able to perform declaring in a more precise way by setting a fixed type for a return value. The positive effect of the stated functionality is the prevention of unwanted return values. For example, for the following function return value of integer type is obligatory and return of string or some other type causes the fatal error "TypeError".

```
function broj(): int {
   return 1;
}
broj();
```

Code 15 –Setting a fixed return value

According to the listed mechanisms we can already spot the PHP developers' effort to keep up with the most advanced object-oriented languages. A step forward represents the introduction of anonymous classes that contribute to the shortening of performance and accelerating the coding. It is about nameless classes that are preferable in conditions when the class does not need to be documented or it is only being used on a one-time basis during the program execution. Anonymous classes are being used in cases of developing a simple and one-time object.

```
$obj = new class{
   public $msg = 'Greeting';
   public $num = 7;
};
```

Code 16 – Application of anonymous class

Except for direct allocation of the anonymous class to the variable as in the previous code, it is possible to get the identical result by using the function that creates the anonymous class. However, such classes are not entirely anonymous because they are automatically being named and parsed into the global scope and they can be reached again later.

Namespaces that the users were introduced to only in PHP5.x versions are also being improved. Group insertion of classes, constants and functions from the same namespace in one command has been enabled.

It is necessary to also have in mind the removed characteristics and pay attention to them. It is mostly about tiny syntax parts of the programming language. For a start, ASP-style tags have been removed and the programmer is limited to the standard „<?php“ tag. Hexadecimal numbers have been written as a string and they are no longer recognized as a numerical value. The use of multiple default blocks in switch structure from the new version shall cause the fatal error. One of the most prominent syntax changes is the removal of "mysql" extension due to, among other things, lack of support for object-oriented programming, transactions, procedures. Thus, "mysqli" and "PDO" extensions have survived for connection with the database.

4 Conclusion

The number of Internet users grows every day and so does the filling of web sites with accesses and requests.

Allocation of active roles to the user has left a strong influence on the server's development, and by that on programming languages that access the servers. New ideas for improvement of performances and mostly for speeding up the processing of received requests have been constantly occurring.

Different business processes and important transactions have been frequently performed through the Internet in today's rapid modern world, whereby the security and satisfaction of the user becomes a primary reference for further development and web-technologies changes. Accordingly, even the shortest unavailability or the simplest malfunction inside a financially directed system, can cause enormous losses to some bigger or a smaller organization.

The research has shown that by 2020 the Internet traffic shall be increased by 100 times compared to 2005. Judging by this information, PHP shall necessarily be developed among other programming languages.

Overall, the new versions shall strive to additionally simplify the use and learning how to write a PHP programming code and they shall achieve desirable results of saving memory and speed of command processing with new ideas.

5 Reference

[1] Ananthakrishna, R., Chaudhuri, S., Ganti, V. (2002). *Eliminating fuzzy duplicates in data warehouses.* In*:* Proceedings of the International Conference on Very Large Data Bases, pp. 586–597.

[2] Allamaraju S. (2010) *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity.* O'Reilly. United States of America.

[3] Castano, S., Ferrara, A., Montanelli, S. (2010). *Dealing with matching variability of semantic web data using contexts*. In: Proceedings of the International Conference on Advanced Information Systems Engineering, pp. 194–208.

[4] Dagiene V., Jevsikova T., Kubilinskiene S. (2013) *An Integration of Methodological Resources into Learning Object Metadata Repository*. Vol. 24, No. 1,. Vilnius University Institute of Mathematics and Informatics, pp. 13–34.

[5] Hernandez, M., Stolfo, S. (1995). *The merge/purge problem for large databases*. In:Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 127–138.

[6] Lockhart J., (2015) *Modern PHP: New Features and Good Practices.* O'Reilly. United States of America.

[7] Mitchell L.J. (2016). *PHP Web Services: APIs for the Modern Web*. O'Reilly. United States of America.

[8] Porebski B., Przystalski K., Nowak L. (2011) *Building PHP Applications with Symfony, CakePHP, and Zend Framework.* Wiley Publishing, Inc. United States of America.

[9] Prettyman S. (2016). *Object Oriented Modular Programming using HTML5, CSS3, JavaScript, XML, JSON, and MySQL*. Apress. United States of America

[10] Šubelj, L., Jelenc, D., Zupančić, E., Lavbič, D., Trček, D., Krisper, M., Bajec, M. (2011). *Merging data sources based on semantics, contexts and trust*. IPSI BgD Transactions on Internet Research, pp. 18–30

[11] Žitnik S., Šubelj L., Lavbič D., Vasilecas O., Bajec M. (2013) *General Context-Aware Data Matching and Merging Framework*. Vol. 24, No. 1, Vilnius University Institute of Mathematics and Informatics, pp. 119–152.