College of Polytechnics Jihlava
College of information Management, Business Administration and Law
Czech Technical University in Prague, Faculty of Electrical Engineering

SDOT 2013

Proceedings of federated conference on 39th Software Development and 17th Object Technologies

Martin Molhanec, Michal Bejček, Jan Voráček (Editors)

Jihlava, Czech Republic, November 8, 2013

Steering committee:

Chairman:

Martin Molhanec	Czech Technical University in Prague, CR

Members:

Petr Koubský	iCollege, Prague, CR
Branislav Lacko	Brno University of Technology, CR
Vojtěch Merunka	Czech University of Life Sciences, Prague, CR
Rudolf Pecinovský	University of Economics, Prague, CR
Bogdan Pilawski	Bank Zachodni WBK, Wroclaw, PL
Jiří Polák	SPIS, CR
Ľubomír Sadloň	University of Žilina, SK
Milena Tvrdíková	VŠB-Technical University of Ostrava, CR
Miroslav Virius	Czech Technical University in Prague, CR
Jan Voráček	College of Polytechnics, Jihlava, CR
Štefan Zajac	Czech Technical University in Prague, CR

Programme committee:

Chairman:

Vojtěch Merunka	Czech University of Life Sciences, Prague, CR

Members:

Michal Bejček	College of Information Management, Business Administration and Law, Prague, CR
Dagmar Brechlerová	Czech University of Life Sciences, Prague, CR
Alena Buchalcevová	University of Economics, Prague, CR
František Huňka	University of Ostrava, CR
Ján Janech	University of Žilina, SK
Tomáš Kozel	University of Hradec Králové, CR
Emil Kršák	University of Žilina, SK
Martin Molhanec	Czech Technical University in Prague, CR
Robert Pergl	Czech Technical University in Prague, CR
Tomáš Pitner	Masaryk University, Brno, CR
Jaroslav Ráček	IBA CZ, s.r.o., CR

Karel Richta	Charles University, Prague, CR
Tomáš Richta	College of Polytechnics, Jihlava, CR
Antonín Slabý	University of Hradec Králové, CR
Václav Snášel	VŠB-Technical University of Ostrava, CR
Petr Šaloun	VŠB-Technical University of Ostrava, CR
Michal Valenta	Czech Technical University in Prague, CR
Michal Vopálenský	College of Polytechnics Jihlava, CR

Organising committee (College of Polytechnics, Jihlava, CR)

Chairman:

Jan Voráček

Members:

Zbyněk Bureš

Michaela Machovcová

Antonín Přibyl

Tomáš Richta

František Smrčka

Hana Vojáčková

Michal Vopálenský

Úvodní slovo

Table of contents

LINQ preprocessor for the C++

Jakub Judas, Miroslav Virius

miroslav.virius@fjfi.cvut.cz

Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Praha 2, Trojanova 13

Abstract. An implementation of the LINQ preprocessor for the C++ programming language is presented. Syntactic rules for the queries are derived from the LINQ specification in the C# version 3 programming language. Our implementation allows the queries to the `vector< >` objects, but the extension to other data sources is rather straightforward.

Keywords: LINQ, preprocessor, C#, vector class.

1 Introduction

Query mechanisms similar to the LINQ in the C# are common extensions in many contemporary programming languages, except the C++. In this article we present an implementation suitable for the C++ programming language conforming to the [1] international standard. Our implementation consists of

- the preprocessor that translates the C++ program containing the queries to pure C++ code,
- the header files containing the necessary declarations of auxiliary classes and functions.

The header files contain even the method bodies, because the classes and functions are mostly implemented as templates. The only data source supported in this version of our LINQ preprocessor is the `vector< >` template class from the C++ standard library; nevertheless, it is easy to extend it for other data STL containers or for user's own classes.

1.1 LINQ

The acronym LINQ stands for the Language Internal Query; it was originally the part of the C# programming language and its specification can be found e.g. in [2], [3]. The LINQ query is a kind of expression composed of special keywords, variable declarations and conditions specifying the desired data. These expressions are similar to the SQL queries. An example of a simple query in the C# language is

```
from point in SetOfPoints where point.x < 0
select Math.Abs(point.y)
```

This expression selects all points with negative `x` coordinate from the `SetOfPoints` container that serves as the data source and constructs an object that will contain absolute values of their `y` coordinates.

In this example, `from` is the keyword that starts the query and `point` is an auxiliary variable that will be used to access individual data pieces selected from the data source `SetOfPoints`. The `where` operator introduces the condition describing the desired data items and the `select` operator starts the so called projection – an expression describing, what to do with the selected data.

The result of the query behaves like a container, even though the lazy evaluation is usually applied – next data item is read from the source and processed when a new data item is asked from the result of the query. The queries are translated to a series of invocations of extension methods in C#.

Note that this example does not show all the possibilities of the LINQ in C#. Here is the short review of the remaining LINQ keywords:

- The `let` operator may be used to introduce another auxiliary variable,
- the `orderby` operator may be used to order ("sort") the result,
- the `group by` operator may be used to organize the result into groups,
- the `join` operator may be used to join the results of several queries,
- the `descending` operator in the order by clause forces the result to be arranged in the descending order.

1.2 Our Implementation of LINQ in C++

Our implementation of the LINQ queries in C++ consists of several header files and of the preprocessor linq that converts the queries in into a series of invocations of methods of auxiliary template classes that are defined in the header files. The linq preprocessor has to be called before the standard C++ preprocessor (that is usually part of the C++ compiler). It requires the C++ implementation conforming at least partially to the [1] standard – viz. the lambda expressions and the `auto` keyword in the new meaning.

2 Existing Implementations

There are several implementations of LINQ for C++ available, but they are based on different principles than ours. We describe two of them shortly here.

2.1 cpplinq

The cpplinq implementation (see [4]) may be used in Windows with the Visual C++ 2010, Visual C++ 2012 and g++ v4.7.0 compilers; in Linux it may be used with g++ v4.7.0 and clang++ v3.1 compilers. It is based on the use of

the lambda expressions and some other advanced features of the C++11. The regular C++ arrays and the STL containers may be used as data sources.

This implementation does not support the queries in the form shown in the previous section. Instead, the queries use predefined classes and functions and the overloaded `>>` operator. The following example has been adopted from [4]. Assume we have an array of integers, `ar_ints`, and we need to extract all even numbers, arrange them in ascending order and store them in a `vector<>`. This is done by the following function:

```
#include "cpplinq.hpp"

int compute_a_sum_of_evens (int* arr_ints) {
    using namespace cpplinq;
    return
        from_array (arr_ints)
        >>  where ([](int i) {return i%2 ==0;})
        // Keep only even numbers
        >>  orderby_ascending ([](int i) {return i;})
        // Sort them
        >>  to_vector ();    // Sum remaining numbers
}
```

In this example, `from_array()`, `where()`, `orderby_ascending()` etc. are functions returning instances of predefined classes.

Note that this implementation is contained in one header file.

2.2 boolinq

The boolinq implementation (see [5]) may be used with the latest versions of the C++ compilers; more detailed information could not be found. As well as cpplinq, it is based on advanced features of the C++11. The regular C++ arrays, the STL containers and some Qt containers may be used as data sources.

The queries in this implementation have the form of a series of method calls. The following example is adopted from [5].

```
int arr_int[] = {1, 2, 3, 4, 5, 6, 7, 8};            // data source
auto dst = from(arr_int).where( [](int a){return a%2 == 1;})
                .select([](int a){return a*2;})
                .where( [](int a){return a>2 && a<12;})
                .toVector();
```

This query selects all odd numbers from the data source, computes their squares, selects those lying between 2 and 12 and stores them into a `vector<>` instance.

3 linq preprocessor

Our implementation enables using queries similar to the form usual in C# in C++ source code. As stated earlier, it consists of several header files and the

linq preprocessor. The user has to include the `linq.h` header file. The source file has to be processed by the linq program before the compilation. This can be arranged in the makefile.

This preprocessor may be used with any C++ compiler implementing the lambda expressions and the `auto` keyword in the new sense. Only the STL `vector<>` instance can be used as a data source in the current version.

The syntax of the query differs from the C# syntax only slightly. The main difference is that the type of the auxiliary variable in the from clause has to be declared always. The linq preprocessor translates the LINQ expressions into a series of method calls. Consider the query

```
from int x in array where x%10>5 select x*2
```

that selects from the array instance of the `vector<int>` class the squares of all integers that give the remainder greater than 5 when divided by 10. The result is the pointer to the `LinqVectorExpression<int>` instance. The instance of this auxiliary class contains a `vector<>` instance with the query result as a data member. This query is translated to the following sequence of function and method calls:

```
From(array)->Where([]{int x}{return x%10>5})->
Select<int>([int x]{return x*2})->getData();
```

The condition expression in the where clause, as well as in the projection, is translated to the lambda expression. Note that you may use our LINQ implementation without the preprocessor, if you write the queries this way.

4 Implementation

The linq preprocessor is based on our formal specification of the LINQ expressions (see [6]), which has been derived from the specification in [3]. The compilation uses the so called *island grammar* – only the LINQ expressions are compiled, the remaining code is left unchanged.

The compilation of any source code usually consists of the following phases:

1. Lexical analysis,
2. syntactic analysis,
3. semantic analysis,
4. generation of intermediate code,
5. generation of the final code.

We can omit the semantic analysis and the intermediate code generation, because these phases will be done during the compilation of the resulting code by the standard C++ compiler. The remaining phases are discussed in the following subsections.

4.1 Lexical analysis

During the lexical analysis, the program is decomposed into tokens that are characterized by the pairs composed of the token name and the token position. This is done by a state machine generated by the lexer generator lex [7]. (We have used the GNU implementation of lex known as Flex.) The input of the Flex is composed of the rules for individual token types. Any rule consists of the regular expression describing the token followed by the C statement that should be performed when the token of this type is found. E.g., the rule for the from keyword has the form

```
[^a-zA-Z_]"from"[^a-zA-Z]   {LINQ_FROM_TOKEN(yytext)}
```

where `LINQ_FROM_TOKEN` is a macro defined in the program. Special token `OTHERSYMBOL` is used for the C++ text that is not part of the LINQ query.

4.2 Syntactic analysis (parsing)

During the second phase, an internal representation of the program – the abstract syntax tree (AST) – is built from the tokens. The parser used in the linq preprocessor has been generated by the YACC parser generator [7] (we have used the GNU implementation known as Bison). The input of the Bison is the description of the syntactic rules of the LINQ expressions in a form similar to – but not equal to – the Backus–Naur form. As usual, these rules describe the non-terminal symbols using the terminal symbols or other non-terminal symbols. E.g., the rule

```
from_generators: from_generator
              | from_generator COMMA from_generators
              ;
```

defines, that the *form_generators* non-terminal symbol may be either the *from_generator* alone or the *from_generator* followed by the *COMMA* symbol and the *form_generators* symbol. The *from_generator* and *COMMA* non-terminal symbols must be defined elsewhere. Note that special symbols *other* and *words* were added; these symbols describe the C++ source code that is not part of any LINQ expression.

4.3 Abstract Syntax Tree Nodes

It is necessary to locate the LINQ expressions and to split the original source code in order to replace these expressions by the new code. After that, the AST is constructed. Any node of the AST is represented by an instance of a specialized class. All these classes have the following two methods:

- The `print()` method is invoked when the `<<` operator is used. This method invokes recursively the `print()` method of all its child nodes and outputs the compilation result.

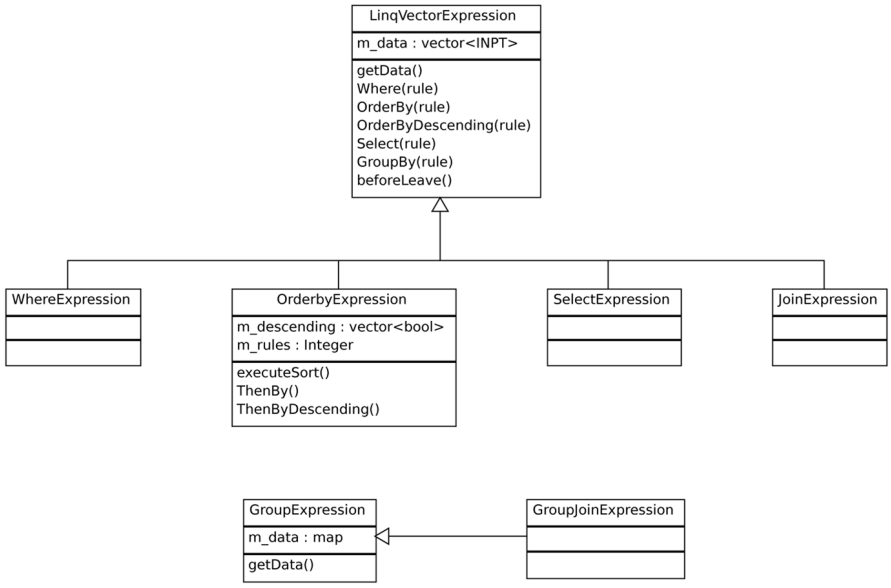


Fig. 1. Classes used for the representation of the query in AST (from [9])

– The `compile()` method translates the LINQ keywords to the C++ code. In some cases – e.g. in nodes of the other type, that need not be translated – this method does nothing.

Fig. 1 shows the class diagram of the main classes in the abstract syntax tree for the LINQ expression with the `vector<>` data source.

5 Conclusion and Outlook

This implementation of the linq preprocessor has been tested with newest Windows and Linux C++ compilers and has been published on the SourceForge web pages [9]. The project documentation has been created by the Doxygen tool [8]. As stated earlier, the only data source supported in the current version is the STL `vector<>`.

5.1 Extension to other data sources

The extension to other data sources is straightforward. To add the support of the data source of type `T`, it is necessary to implement the set of classes with appropriate methods. The `From()` function should have exactly one parameter of type `T`; it should return an instance of the class that contains the `Where()`,

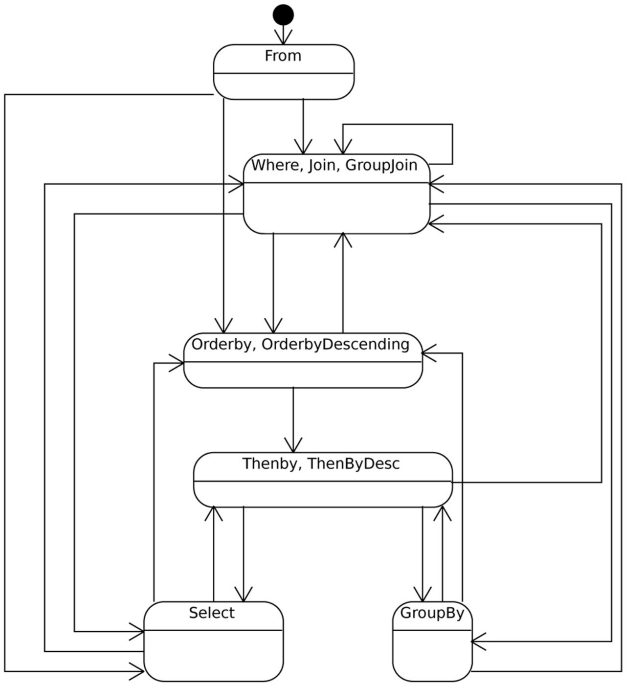


Fig. 2. The diagram of the possible sequences of function calls in the queries (from [9])

`Select()`, `SelectMany()`, `OrderBy()`, `OrderByDescending()` and `GroupBy()` methods. The possible sequences of method calls in the query are described by the state diagram on Fig. 2.

These methods should accept the lambda expression as input parameter and should return an instance of a class that supports the methods allowed in the actual state according to Fig. 2.

Acknowledgement. This work was supported by grant no. SGS 11/167 of the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. International Standard ISO/IEC 14882:2011. Programming Languages — C++. ISO, Genève 2011.
2. Magennis T.: LINQ to Objects Using C# 4.0. Addison-Wesley 2010. ISBN 978-0-321-637000-0.
3. C# Language Specification. Version 5. Microsoft Corporation 2012. Available at http://www.microsoft.com/en-us/download/details.aspx?id=7029 [Accessed July 13, 2013].

4. LINQ for C++. Available at http://cpplinq.codeplex.com/ [Accessed July 13, 2013]
5. boolinq. C++ header-only Ranges and LINQ template library. Available at https://code.google.com/p/boolinq/. [Accessed July 13, 2013]
6. Judas J.: The implementation of the LINQ preprocessor for the C++ programming language. Czech Technical University in Prague 2013. [Diploma project, in Czech]
7. Levine J. R., Mason, T., Brown, D. lex & yacc (2 ed.). O'Reilly 1992. ISBN 1-56592-000-7.
8. Doxygen. Available at http://www.stack.nl/ dimitri/doxygen/ [Accessed July 13, 2013].
9. Linq to C++. Available at http://sourceforge.net/projects/linqtoc/?source =directory [Accessed July 13, 2013]

A Brief Comparison of Groovy and Smalltalk

Josef Smolka

Department of Software Engineering, Faculty of Nuclear Sciences and Physical Engineering, CTU in Prague
smolkjos@fjfi.cvut.cz

Abstract. The paper gives a brief comparison of the Smalltalk and Groovy languages from a viewpoint of a nowadays developer, who is not writing his own sorting procedure, but must design and implement complex information systems and advanced web and graphical applications.

Keywords: Groovy, Smalltalk, XML, history

1 Introduction

An era in which a programmers task was to write a simple database of telephone numbers and sort it according to owners names is history. Nowadays developer has to deal with much more complex and more abstract assignments, which are complicated by integrations and various targeting platforms ranging from numerous mobile platforms to monstrous enterprise class middleware. This paper offers a brief comparison of both languages considering the needs of such a developer. Goal of the paper is not to decide which language is better, but to put the best from both worlds to one place. Let first compare history and evolution of the languages, to get a better understanding of an environment in which the languages were born.

It is early sixties when Alan C. Kay is discovering Sketchpad and Simula and thus getting the feeling of OOP in the form of master and instance drawings (case of Sketchpad) and activities and processes (case of Simula). It is an era, in which the idea of a personal computer is emerging. In these times, a programming language and a programming environment are integral part of the computer and greatly influence overall user experience. Between years 1967 and 1969, Kay is cooperating with Dave Evans on the design of FLEX machine. While Evans is concerned with the hardware, Kay is contemplating on software to support the hardware, influenced by his encounters with Sketchpad, Simula, Algol, Euler, Joss, Logo and Grail. Conference on Extensible Languages, which is held in 1969, gives Kay the idea of interpreted language as opposed to compiled ones, which is a standard in these times. In 1969, Kay is also getting the idea of access control to objects internals from CAL-TSS system. In this system pointers are enhanced by a bitmask specifying the access restrictions. In the context of the Smalltalk history, the beginning of seventies belongs to Xerox research center in Palo Alto, California, particularly to Kays Learning Research Group (LRC) contemplating on the ideas of small personal computer - notebook. In 71, Kay is refining his

idea of KiddiKomp machine into a new concept miniCOM, computer with new programming language called Smalltalk, indicating that programming should not be cryptic mystery but easy understandable even by children. Smalltalk-71 is born. Next year is a year of bets. One of bets between Ted Kaehler, Dan Ingalls and Kay is about how large a language must be to have great power. Kay is declaring that the most powerful language in the world can be defined in a page of code. Kay is getting done the concept; Ingalls is later implementing it in BASIC. First version of Smalltalk-72 is born [8]. By the end of year 1972, six basic principles are drawn [10]:

- Everything is an object.
- Objects communicate by sending and receiving messages.
- Objects have their own memory.
- Every object is an instance of a class.
- The class holds the shared behavior for its instances.
- To evaluate a program list, control is passed to the first object and the remainder is treated as its message.

This list can be considered as mantra for the new Smalltalk programming language, which is succeeded by Smalltalk-74 and Smalltalk-76. These versions bring new virtual memory system with garbage collection procedures. During further years Smalltalk is given much more attention even by Xerox executive board. Many demos of Smalltalk features are done. During one presentation for Apple in 1979, Steve Jobs says that he does not like the way the content is scrolled on the screen in the presented application and asks if it could not be scrolled in a smooth continuous style. Dan Ingalls does the required change in less than minute and the change is immediately reflected in the application. This story nicely demonstrates the incremental power of Smalltalk which is missing to majority languages even nowadays. In 1980, major release of Smalltalk, Smalltalk-80, is published [7]. It is based on Smalltalk-78 which was developed parallel to NoteTaker application. Two main branches are derived from Smalltalk-80: Squeak derived from the first version of Smalltalk-80 and VisualWorks derived from the second version [10, 9]. Business success of Smalltalk in nineties is complicated by several factors including high price, intensive memory requirements and lacking support of raising SQL databases. The position of Smalltalk is even more complicated by the birth of Java (1996, JDK 1.0), as Sun is choosing for Java much more liberal distribution model and Java is available for download from java.sun.com for free.

Back in eighties, two years after the release of Smalltalk-80 another event important to this story occurs and also in Palo Alto. Andy Bechtolsheim, graduate student of Standford University is founding in cooperation with Vinod Khosla and Scott McNealy new company Sun Microsystems. Eight years after that, Sun is a prospering company and can also afford to carry out project contemplating on feature of computing and computers as was common in seventies. Project Green is expected to bring answers to questions what will the computer market looks like in the near future. The answer of involved prophets is that the future lays in fusion of computers with consumer electronics. The result of this vision

is small PDA Star 7 targeted to cable companies as a smart remote controller for televisions. The side product of this PDA is a new language intended for programming of devices like Star 7. The language is designed by James Gosling and named after a tree, which grows in front of the Goslings office - Oak. Thanks to this language and the WebRunner browser, the first dynamic content is brought to internet. In 1994 Oak is renamed to Java, as the name Oak is used already by another company. Since the first public release in 1996 to present days, Java has undergone gradual development; rather an evolution then a revolution. The Java platform, governed by Java Community Process, is quite conservative concerning adoption of new ideas. This has positive effect on adoption of the Java platform by enterprise domain as fundamental core of enterprise middleware. On the other hand, the Java language is still lacking demanded features like closures, list literals, multiple return values and others [5]. This is creating a space for new languages to fill in the gaps created by slow evolution of Java. In 2003, James Strachan, the author of Groovy, is playing with dynamic languages, Python and Ruby, and with their respective JVM versions, Jython and JRuby, and feel a necessity to complement the overall complex (but lacking core functionalities at the same time) and sometimes problematic original language of the Java platform with something new and fresh [11]. Dynamic languages like Python, Ruby and even Javascript enjoy great popularity by present developers, but transition to the Java platform is not so straightforward and JVM implementations of these languages suffer from it. This is the main motivation for designing new language from the ground, which would resemble Java in core syntax a compile directly to the Java byte code. The language gets the name Groovy because it is built on top of all the groovy parts of a Java code. At the beginning of 2007, the first stable version 1.0 is released [11, 1]. Groovy, despite the fact that it is directly compiled to byte-code, is criticized for its lack of performance. In 2012, Groovy 2.0 is released, which brings huge improvements in speed.

2 Method of comparison

The comparison of Smalltalk and Groovy is not focused on the language part only, but it is also a comparison of the platforms and the support they give to the hosting languages. Thus, the first part is dedicated to distribution model comparison, mainstream IDEs and other supporting tools like versioning control, frameworks for web, XML and services. The second part is focused on the languages themselves, comparing code efficiency and simplicity (expression power), code portability and execution efficiency.

3 Language support

Every sentence “I want to code in Smalltalk” is sooner or later followed by “Wait, there are so many implementations, which to choose?”. It can be confusing for beginner to orient himself in various implementations that have their roots in Smalltalk-80:

– VisualWorks - Derived from the second version of Smalltalk-80. It is a cross-platform implementation and runs on variety of systems including Windows, Mac OS X, Linux and several UNIX-type systems. It is freely available for non-commercial use. Commercial license demands the company to share its profit with the owner of VisualWorks.
– Squeak - Derived from the first version of Smalltalk-80. Squeak was originally developed by Apple and later developed by Walt Disney Imagineering. Squeak is freely available under combination of MIT and Apache licenses [2].
– Pharo - Fork of Squeak created in 2008. The main focus of Pharo is to bring users a clean and innovative open-source Smalltalk-like environment [3].
– GNU Smalltalk - Implementation under GNU project. It differs from mainstream implementation in a way that it works with standard text files and not with an image of the system. Advantage of GNU Smalltalk lays in native bindings for libraries like SDL and SQLite [4].

As can be seen from the list, situation from nineties, when Smalltalk was pure commercial product is history and variety of free implementations exists. Besides the traditional implementations, there is also a variety of implementations for uncommon platforms like Amber Smalltalk, which runs atop JavaScript or several implementations that run on Java Virtual Machine (Bistro, Athena). In most cases, implementation contains platform specific virtual machine and platform independent image containing development environment and standard library.

A legacy of times in which Smalltalk was born is that in many cases the Smalltalk has its own windows system (Squeak, Pharo) bypassing the main window manager. That can be inconvenient for people used to different schemas of windows managing. On the other hand, platform independent GUI library is a considerable advantage for multiplatform development. When it comes to versioning, standard tools like Subversion, Git and others do not work for Smalltalk as these tools are file-based. Monticello is distributed optimistic semantic versioning system for Smalltalk. It can have several distributed remote or local repositories and allows versions to move between the repositories. It is optimistic in a way that it does allow to create new version from code available to developer without version locking in repository or another similar mechanism. Monticello does versioning on a semantic level and recognizes semantic structures as classes or methods. This helps the merging algorithm to behave in more consistent way [2, 3].

With spread of personal computers, new style of systems architecture quickly emerged the client/server architecture pattern. The more traditional client, programmed using native GUI library (thick client), is nowadays replaced in a great portion of information systems by thin client running in web browser. Seaside is Smalltalk web application programming framework consisting of HTML generators, action callbacks and advanced state and control flow management. Web applications in Seaside are constructed using components. Every component is responsible for a definition of user interface and the control flow. Rendering of component HTML interface is done by provided generators; there is no mixing

of HTML and Smalltalk code. But there is also no support for templates, business logic is combined with presentation logic. Every component has its own control flow directed by actions. When an user clicks on a link, one component goes further in its control flow and may display different content, while other components remain the same. Seaside draws its power from the incremental development model and debugging capabilities of the Smalltalk language itself [6]. While definitely more elegant during the development, the deployment phase prevents Seaside from greater penetration in an enterprise environment. The reason is lack of application middleware which could shield the application from data source definitions and a connection pooling, from identity management systems, thread pooling and provide a communication channels using messaging queues. Seaside is not the only web framework available, others are AIDAweb and Iliad. Lacking support of traditional SQL databases is another drawback, Squeak has native binding for MySQL and PostgreSQL but that is all. Connection to another legacy databases (from the viewpoint of Smalltalk) is provided by ODBC, but it is single-threaded and blocks the VM during the query. On the other hand, Smalltalk ecosystem offers several object databases: GOODS, Omnibase, SandstoneDB and Gemstone, where the last one seems to be the most mature and enterprise ready. Another drawback is lacking strong support for nowadays standards like SOAP, WS-*, WSDL, UDDI and other technologies used for implementing service-oriented systems. This all complicates the position of Smalltalk in a decision process of a system architect.

Groovy is built atop the Java platform, which is widely spread and adopted by enterprise sphere. It is alternative language for Java platform compiled directly to byte-code interpretable by Java Virtual Machine. The extended language syntax and features are inspired by Ruby, Python and Smalltalk, but the core is quite similar to original Java, thus the language is easily adopted by Java programmers. Groovy is distributed as a package containing compiler and standard library. Groovy can be used in two modes, as a primary language compiled to byte-code or as a scripting language within the Java code. In both cases, arbitrary Java library can be used in Groovy which gives Groovy all the strength of the Java platform. Groovy is fully supported in mainstream Java development environments like IntelliJ IDEA, Eclipse and NetBeans and several coding editors like Emacs, JEdit and TextMate. Making of GUI applications in Groovy is backed by Javas Swing framework and SwingBuilder, which allows creating of full-fledged Swing GUI in a declarative manner. When there is a need for complex application framework for building desktop applications, Groovy platform can offer the Griffon framework following the convention over configuration paradigm and providing intuitive MVC pattern implementation. Versioning on Java platform is ensured by standard developments tools like CVS, Subversion, Mercurial, Git and others, there is no need for a special tool just for Groovy.

Nowadays, Groovy is perhaps mostly known through its web framework called Grails. It takes over ideas like convention over configuration from Ruby framework Ruby on Rails, and it is based on one of the most acclaimed J2EE frameworks Spring. The Spring framework is providing its lightweight inver-

sion of control container enabling Grails to use automatic dependency injection. Object relation mapping is ensured by proven Hibernate library.

4 Language efficiency

Java is often justly criticized for its chatty syntax. In contrast, the syntax of Smalltalk is much more elegant, using minimalistic form for message sending, code blocks and lambda expressions. As Groovy is inspired by Smalltalk, the syntax of Groovy, while still faithful to its Java roots, contains new constructs for list literals, lambda expressions, closures and others. Let take a look at very simple example which consists of XML parsing.

The Pharo way

```
|xml a|
xml := 'testxml.xml' asFileReference readStream contents .
a := (1 to: 25) collect: [ :x |
  Time millisecondsToRun: [ XMLDOMParser parseDocumentFrom: xml ]
].
Transcript show: a sum / a size asFloat .
```

The Groovy way

```
def xml = new File("testxml.xml").getText()
def a = (1 .. 25).collect{x ->
  millisecondsToRun{new XmlParser().parseText(xml)}
}
println a.sum() / a.size()
```

It can be seen from examples that Groovy resembles the syntax of Smalltalk in many ways. This example was also used for performance measurement. Let assume, that application like service bus, which routes and transforms XML messages, is going to be programmed. Thus, performance in XML parsing is critical to such application. Measurements were taken on following configuration: Intel Core i7, 2.67 GHz, 8 GB RAM, Windows 7 64b using Groovy 2.1.7 with Java 1.7 64b, Pharo 2.0 (update #20621) and Squeak 4.3 (update #11860). XML parsers were used in default configurations.

The table 1 shows that Groovy outperforms free distributions of Smalltalk in XML processing. The worst candidate for service bus implementation is definitely Pharo, while Squeak is comparable to Groovy.

5 Conclusion

The paper gave a comparison of Smalltalk and Groovy languages based on needs of nowadays developer. The comparison started with historical background of

Table 1. Execution times of XML parsing for Pharo, Squeak and Groovy.

Language	XML Size	Execution time
Groovy	2.95 MB	82.84 ms
Pharo	2.95 MB	1337.0 ms
Squeak	2.95 MB	133.52 ms
Groovy	29.5 MB	653.8 ms
Pharo	29.5 MB	20580.36 ms
Squeak	29.5 MB	1918.92 ms

both languages and continued with comparison of platforms and languages performance in XML parsing. The result shows that there is a substantial difference in various Smalltalk implementations. The results also indicate that Groovy could bring sufficient performance for services implementation while still offering the comfort of the Smalltalk syntax. The substantial drawback of Groovy is hidden in Java platform itself - lacking debugger functionality that can be hardly compared to incremental development capabilities of Smalltalk.

Acknowledgment. Creation of this paper was supported by grants SGS11/ 167/OHK4/3T/14 and LA08015.

References

1. Batsov, B.: Java.next()–the Groovy Programming Language. `http://batsov.com/articles/2011/05/06/jvm-langs-groovy/` [2013-09-29]
2. Black, Andrew, et al.: Squeak by example. (2007). `http://squeakbyexample.org` [2013-09-29]
3. Black, Andrew, et al.: Pharo by example. (2009). `http://pharobyexample.org` [2013-09-29]
4. Byrne, S., Bonzini P., Valencia, A.: GNU Smalltalk User's Guide. (1990)
5. Devijver, S.: Groovy Will Replace the Java Language as Dominant Language. `http://groovy.dzone.com/news /groovy-will-replace-java-langu` [2013-09-29].
6. Ducasse, S., Lienhard, A.,Renggli, L.: Seasidea multiple control flow web application framework. Proceedings ESUG 2004 International ConferenceResearch Track, volume Technical Report IAM-04-008. (2004)
7. Goldberg, A.: SMALLTALK-80: the interactive programming environment. Addison-Wesley Longman Publishing Co., Inc. (1984)
8. Ingalls, D.: The Smalltalk-76 programming system design and implementation. Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM. (1978)
9. Ingalls, Dan, et al.: Back to the future: the story of Squeak, a practical Smalltalk written in itself. ACM SIGPLAN Notices 32.10, 318–326 (1997)
10. Kay, Alan C.: The early history of Smalltalk. History of programming languages–II, 511-598 (1996)

11. Strachan, J.: Groovy–The Birth of a New Dynamic Language for the Java Platform. http://radio-weblogs.com /0112098/2003/08/29.html#a399 [2013-09-29].

Java Design Patterns Automation Survey

Marketa Horakova

Department of Informatics and Quantitative Methods, Faculty of Informatics and Management,
University of Hradec Kralove, Czech Republic
Marketa.Horakova.4@uhk.cz

Abstract: This paper examines the current possibilities and research activities in the area of design patterns automation and how it is supported in CASE tools dedicated for Java developers. The submission provides classification of the CASE tools according to the level of automation that facilitates design patterns usage and application in the software development life cycle and describes examples of these tools suitable for Java developers.

Keywords: Object-oriented Programming, Java, Design Patterns, CASE Tools

1 Introduction

Design patterns solve the common design and programming problems, make the software design more flexible and reusable and thus represent the best practices in object-oriented software design and development.

As the design pattern automation can be marked an approach that, in any degree, automates the application of design patterns at the implementation stage of the software development process.

This survey examines CASE tools suitable for Java developers that support design pattern automation. The scope of the article is thus focused on Java programming language, although the implementation of design patterns can be done in any object-oriented language; initially the patterns have been implemented in C++ or Smalltalk code.

First of all the paper provides an overview of design patterns and their principles. Then it is concerned with the design pattern automation definition and classification of related development tools and environments. After that the tools suitable for Java developers are listed for each of defined categories and their pros and cons are discussed. Finally, the two examples of expert systems focused on advising of design patterns application for novice software designers and developers are described.

2 Design Patterns

The authors of the key book about design patterns, [1], defined the term design pattern as “a description of communicating objects and classes that are customized to solve a general design problem in a particular context”. Now the design patterns are considered to be the best practices used in many recurring design problems.

As Freeman [2] explains, a pattern is a solution to a problem in a context and the context is the situation in which the pattern applies. Design patterns solve the common software design problems during object-oriented application development.

The solutions described by the pattern are usually general and can be applied in several common situations. On the other hand, the general solution can be also implemented in several ways, so in the next parts of the article we will face difficulties in some tasks related to automation of design patterns usage and recognition.

2.1 Design Patterns Description

Design pattern can be identified by name and by participating classes and instances and relationships between them. The next attribute of design pattern is usually description of the situation, specification of the object-oriented design problem, when it can be applied.

Gamma et al. [1] pointed out four essential elements of the design pattern:

1. *The pattern name* is an important part of the pattern as it makes a handle that is used to be shared between the developers; usually the name summarizes the problem and its solution in one or two words.
2. *The problem* describes the problematic situation within its context when it is advantageous to apply the pattern. Usually it describes specific object-oriented design problems.
3. *The solution* describes the classes and their collaborations and responsibilities that form the design in general way. The solution and the structure of design pattern can be typified by one of the graphical notations. The essential patterns defined in [1] have been represented in the notation based on the Object Modeling Technique (OMT), which is a predecessor of Unified Modeling Language (UML) used heavily nowadays.
4. *The consequences* are the results or effects that applying the pattern could have and they can be both positive and negative. According to [1], this element can help to understand the costs and benefits of applying the pattern. The consequences of a pattern include also impact on flexibility and extensibility of a software system, so they should be red carefully.

2.2 Design Patterns Categories

Design patterns are most frequently classified into three groups according to their main intent.

Creational design patterns deal with the process of object instantiation; the three main representatives of this are the Factory Method Pattern, the Abstract Factory Pattern and the Singleton Pattern. Another patterns belonging to this group are the Builder Pattern or the Prototype Pattern.

Structural patterns involve the composition of classes or objects into larger structures. Following patterns deal with the objects composition: the Decorator Pattern, the Adapter Pattern, the Façade Pattern, etc.

Behavioral patterns characterize how classes or objects interact and distribute responsibility; the following two patterns are great examples of it: the Strategy Pattern

and the Observer Pattern. The another representatives of this patterns group are the Command Pattern, Visitor Pattern and the State or Template Patterns

Another distribution of the design patterns can be done according to their scope. Pattern can be either applied primarily to classes, or objects. The class patterns are more static and relationships between classes are implemented via class inheritance. Object patterns are more dynamic and deal with object relationships that can be changed at program run-time.

2.3 Design Patterns Purpose

The main advantages of design pattern applying recognized by [1] include following areas:

- Facilitation of reusing the proven and successful designs and architectures. Software development companies often maintain a design patterns catalog to make these solutions more accessible to developers.
- Design patterns descriptions help simplify the choice of design alternatives that make a system better flexible or reusable.
- Design patterns specification can improve the documentation, understanding and maintenance of existing systems.

3 Design Patterns Automation

According to [3], design patterns automation may be defined as an approach that applies design patterns at the implementation stage of the software development life cycle.

The first publication that mention this term is [4] and author defines it as an approach of applying design patterns to software construction.

In this article, the design pattern automation means an approach that in any degree automates the application of design patterns during the software development or maintenance process.

In the next parts of this submission we shall see that this automated design patterns application can be beneficial in the design stage, for example during system's UML diagram creation, in the implementation phase, directly in the integrated development environments or during the reengineering or maintenance process.

3.1 Design Patterns Automation Purpose

The original purpose of design patterns automation is to accelerate and simplify the design pattern usage and applying during the design and implementation stage of software development.

The further levels of design pattern automation, specifically automatic pattern discovery, can be beneficial also in the following areas, as noted in [5]:

- Code optimization.
- Software documentation management.
- Program model verification.
- Program design recovery.

- Reverse engineering.
- Bug finding.
- Security vulnerabilities discovery.

3.2 Design Patterns Automation: CASE Tools Classification

The four main categories of design patterns support in development environments and CASE tools can be recognized according to the level of automation:

- UML modeling tools and their support of pattern application tasks within the model editor.
- Integrated development environments (IDEs) and their support of pattern application in the code editor.
- Code analysis tools focused on automated design patterns recognizing.
- Expert systems that combine design patterns discovery techniques with the knowledge base related to patterns usage and which intent is to advise the best design patterns application to software developers.

UML Modeling Software Environments

The UML modeling tools and their features to simplify design patterns application into the model can be beneficial in the forward engineering process.

Forward engineering deals with the transition from the high-level or abstract design model of a system into its physical implementation. The UML models of classes and relationships between them can be used in this context for automatic platform-specific code generation; this topic is examines e.g. in [6].

According to the level of design pattern automation, modeling tools can be divided into following groups [4]:

- Static – tools that enable inserting a group of design pattern classes into the modeling workspace
- Dynamic – tools that integrate selected design patterns classes with existing classes
- Wizards – tools that provide a wizard for each pattern, in which developer can customize the selected pattern

Chapter 4 of this article examines the possibilities of design pattern automation in UML modeling tools.

Development Tools and Environments

Similarly to modeling CASE tools, the following levels of design pattern automation can be observed in software development environments:

- Static – tools that enable inserting a group of classes into the project
- Dynamic – tools that automatically refactor existing classes and method according to the inserted pattern
- Wizards – tools that provide a wizard for pattern application

Design patterns automation functionality is often available via plug-ins for integrated development environments (Eclipse IDE), or it is directly integrated in the environment (NetBeans IDE). Part 5 of the article provides examples of these tools.

Design Patterns Detection Tools

Another group of development tools with high level design pattern automation is created by tools for automatic design patterns discovery in the current code. They are mostly used for reverse engineering.

Reverse engineering, as opposed to the forward engineering, require code analysis in order to detect high-level model of a system and its components. Lande [7] defines the term as follows: "The concept of reverse engineering refers to a variety of practices undertaken to understand how a software program is built and how it achieves its functionality."

Chapter 6 contains an overview of design patterns detection approaches suitable for Java code analysis.

Advanced Design Patterns Automation Tools

Advanced design pattern automation tools include expert systems that can help the novice software developers with the choosing and application of design patterns. Two such systems are described in the part 7.

4 UML Modeling Software Environments

The first paper that explores the state of pattern automation in the UML modeling software was produced by [4]. In that time the UML environments began to introduce support for design patterns and the survey presents some of them and provides also their comparison according to the level of automation.

Author [4] compares UML modeling tools according to the level of design patterns automation and defines three main categories of these tools:

Static group of these tools enables basic insertion of design pattern classes into the modeling workspace. Advantage of these tools is modeling time saving, but the inserted elements must be then manually customized.

Dynamic support of design patterns enables automatic integration of design patterns classes selected for inserting into the workspace with existing classes. Classes and relationships are renamed and updated automatically, so the developer does not have full control when applying selected pattern.

The best approach seems to be the *Wizard* feature for each pattern. Wizards offer specific customization of the design pattern before its elements are inserted into the workspace.

All these types of design pattern automation tools so far described do not replace the designer's deliberations if to apply the pattern in current design or what pattern to choose, tools those advise pattern selection will be mentioned in the chapter 7.

Bulka [4] compares following three UML tools: ModelMaker [8], UML Studio [9] and Together [10]. Dascalu [3] pointed out, that all the UML-based design patterns automation tools use graphical user interface to customize design pattern before inserting it into the workspace. Together uses dialog boxes, ModelMaker and UML Studio use wizards. ModelMaker is a refactoring and UML modeling tool for Delphi and C#, so it is out of scope of this article. Together and UML Studio can be beneficial also for Java software system designers.

UML Studio

UML Studio [9] is a modeling tool for object-oriented model. It allows documentation and code generation in many programming languages including Java, so it can be used in the forward engineering process. The code generation is based on scripts that users can customize. As described by [3], UML Studio use wizards when inserting design pattern into the UML model.

Together

Together [10] is a software modeling tool that enables code generation from UML diagram into Java and contains support for design patterns. Dascalu [3] provides description of design pattern automation in Together via dialog boxes. First of all the user choose a pattern to use, then the dialog appears with all the necessary parameters for pattern and user can customize the pattern properties before the UML diagram is created.

Rational Rose Modeler

Rational Rose [11] is a set of products for software system modeling and development provided by IBM. The tool Rational Rose Modeler intended for UML modeling enables code generation from visual models and supports also patterns-based modeling.

Description of principles, how Rational Rose products support patterns automation in previous versions, provides e.g. [12]. The design pattern class diagram can be inserted into modeling workspace as a static element, similarly to inserting classes or interfaces. Design pattern static elements contain description according to pattern catalog defined in [1]. Also the dynamic aspects of each pattern are taken into account. So the collaboration, sequence, and activity diagrams for each pattern and its components are also stored in the tool.

The comparison between older versions of Rational Rose UML tools and Together tool based on design pattern support provides [13]. The comparison used criterions like patterns supported, implementation languages supported, source code generation, support of pattern creation, etc. and both tools has been assessed as being analogue in the area of design patterns support.

Enterprise Architect

Enterprise Architect [14] is another example of visual modeling platform for UML diagram creation and code generation into more than ten programming languages. Elements of design patterns can be also inserted into the modeling workspace easily. The extension for Enterprise Architect with design pattern catalog and definition can be downloaded from the tool web site.

5 Development Tools and Environments

Similarly to UML CASE tools we can observe basic design pattern automation support in integrated development environments focused on programming code editing.

5.1 Eclipse IDE and Platform

Eclipse is a software development platform that among others supports Java language. Eclipse platform supports also user-developed plugins to add new functionality to the IDE.

According to [15], "The Eclipse Platform subproject provides the core frameworks and services upon which all plug-in extensions are created. It also provides the runtime in which plug-ins are loaded, integrated, and executed." The Eclipse implementation itself includes the usage of many design patterns, the architecture with respect to design patterns is described e.g. in [16].

So it is not any surprise that there are several Eclipse plug-ins that support design pattern automation; brief overview of some such tools follows.

PatternBox

PatternBox [17] is an editor for Eclipse focused on the work with design patterns. The tool creates Java classes and interfaces according to the selected pattern and they can be customized depending on the application needs.

PatternBox plug-in provides extension to Eclipse's Java development tooling and the Plug-in Development Environment and allows template based code generating.

PatternBox tool generates a special file that represents the pattern and can be customized, e.g. location of the pattern can be chosen. Then the tool generates sample code of that element of the pattern.

Pattern Wizard

Vandyke [18] proposes in his article solution called Pattern Wizard. This tool differs from the PatternBox. While PatternBox enables only generation of the new code, proposed Pattern Wizard contains functionality that enables current code modifying, which requires ability to code reading and parsing.

Design Pattern Toolkit

Design Pattern Toolkit is an Eclipse plugin similar to PatternBox that enables generation of the Java source code for design patterns. For this purposes it uses XML templates. The intent of this tool summarizes [19]: "The idea is that experienced developers create templates for standard code structures that inexperienced developers then use to generate those code structures for the specific needs of their custom applications."

Java Pattern Wizard

The Java Pattern Wizard [20] is another tool based on Eclipse that allows design pattern classes generation. It was included in the CodePro products in the past. The solution provides a pattern wizard that enables to modify properties of pattern selection, pattern instantiation and code generation.

5.2 Other Tools and IDEs

NetBeans IDE

NetBeans [21] is, similarly to Eclipse, a popular open-source development environment supporting several programming languages. NetBeans IDE includes also UML facilities and design patterns support.

As [22] describes, IDE in the version 6.5 contains an 'UML diagrammer' that enables code generation by using Freemarker templates. It supports all design patterns defined in [1] that can be inserted into the workspace. Before insertion, the patterns can be customized by wizard, e.g. existing classes can be selected to fulfill pattern roles.

DPAToolkit

According to [23], the Design Pattern Automation Toolkit is "a tool to help in software development via design patterns. The design can be visualized via class diagrams and design patterns can be incorporated into the design easily." The design patterns are stored in XML format.

The tool supports forward engineering process as the code generation is enabled into the several languages, and also reverse engineering as it allows to some extent creation of class diagram from the code.

6 Design Patterns Detection Tools

Another extensive group of tools with high level design pattern automation is focused on automatic design patterns recognition in current code.

Discovering of design patterns from the source code can help to understand the system and support the process of re-engineering. It can be useful also for gaining the original architecture of the system and simplifying the system maintenance. Design patterns detection tools are also heavily used for reverse engineering.

Jakubik [24] remarks two main approaches in code analysis:

- *Static (structural) analysis* - static structure of a software system is compared with static structure of a design pattern.
- *Dynamic (behavioral) analysis* - compares behavioral representation of design pattern and behavioral representation of running system. However, this kind of analysis is dependent on the executed part of system.

In general, the design pattern discovery in the code can be quite successful for the patterns with unique class structure, the most problematic patterns for the detection are those that have similar structure and differ only in their intent. Sindelar [25] provides an example with the Bridge and the Adapter pattern: "The Bridge is used during the design phase, but the Adapter is used to wire up already existing classes".

The first tools and approaches in the area of design patterns discovery were mostly focused on inspection of C++ code, for example Columbus or Maisa [26]. After the Java became popular, approaches focused on Java code analysis started to be researched as well. The overview of the automatic design pattern recognizing tools for Java follows.

HEDGEHOG

One of the first tools focused on design patterns analysis in Java code is called HEDGEHOG. Blewitt [27] pointed out the main principles of the tool: "Patterns are defined in terms of variants, mini-patterns, and constraints in a pattern description language called SPINE. These specifications are then processed by HEDGEHOG, an automated proof tool that attempts to prove that Java source code meets these specifications".

Its main purpose is to verify patterns in Java code, but it could be used also as a basic pattern detection tool by brute-force searching of existing code bases.

As [28] remarks to the usage of the HEDGEHOG as a pattern recognizer, the user has to specify a target class and a target pattern to verify against.

PTIDEJ

PTIDEJ shortcut means Pattern Trace Identification, Detection, and Enhancement in Java and according to [29] "the PTIDEJ project aims at developing a tool suite to evaluate and to enhance the quality of object-oriented programs, promoting the use of patterns, at language-, design-, or architectural-level".

One of the first PTIDEJ modules focused on design pattern detection called EPI (Efficient Pattern Identification). EPI used the algorithm that finds occurrences of design patterns using bit-wise operations on a finite set of bit vectors representing a program.

FUJABA

The Fujaba Tool Suite [30] is an open source CASE tool providing developers with support for both forward and reverse engineering in Java and thus the name is an acronym for "From UML to Java and back again". It serves also as a implementation tool for few design patterns detection algorithms, as it is described e.g. in [31].

DP-Miner

Dong et al. [32] present in the paper another approach and tool to discovering design patterns by defining the structural characteristics of each design pattern in terms of weight and matrix. System description is based on the XML Metadata Interchange format for metadata that specifies how UML models are mapped into a XML file. The XMI file then can be searched for patterns.

Crocopat

Crocopat is a tool that allows analyzing graph models of software projects in order to find patterns written by Dirk Beyer.

As [33] explains, Crocopat uses binary decision diagrams to represent relations. The design patterns deal heavily with the high-level design of classes and their relationships which can be easy to express with relations.

Crocopat cannot parse Java code, so another tool has to be used in order to transform Java code into an abstract syntax tree.

PINOT

Shi [28] presents a fully automated pattern detection tool for Java, called PINOT (Pattern INference recOvery Tool). This tool is suitable for reverse engineering and

for this purpose the design pattern description has been updated and reclassified into the new categories. Shi et al. observed promising results with this approach: "Based on this reclassification, we automated the entire pattern recognition process using only static program analysis. This relatively simple approach has proven effective."

Pattern Detection Engine

Birkner [34] came up with an approach of design pattern detection that is using static and dynamic analysis in Java application. Each design pattern has defined static and dynamic definitions. The static definitions are used to find pattern instances during the static analysis. Then the dynamic analysis starts that examines running application.

Ontology-Based DPD Tool

Paper [5] presents an approach with ontology-based architecture for pattern recognition. The proposed system searches for patterns by static source code analysis.

Kirasic [5] explains the main principles of the tool as follows: "The parser subsystem translates the input code to AST (abstract syntax tree) that is constructed as an XML tree. The OWL ontologies define code patterns and general programming concepts. The analyzer subsystem constructs instances of the input code as ontology individuals and asks the reasoner to classify them."

MARPLE

Tosi et al. [35] present design pattern detection plug-in for Eclipse called MARPLE (Metrics and Architecture Reconstruction Plug-in for Eclipse) that also uses static source code analysis.

MARPLE approach to design pattern detection is based on the detection of design pattern subcomponents which indicate the presence of patterns. Abstract Syntax Trees representation of the analyzed system is parsed in order to obtain the structures for subcomponents detection.

ePAD

Another Eclipse plug-in for the design pattern discovering is suggested by [36]. Contrary to the previous tools, this solution uses both static and dynamic analysis. As author noted, "The tool is able to recover design pattern instances through a structural analysis performed on a data model extracted from source code, and a behavioral analysis performed through the instrumentation and the monitoring of the software system".

DeMIMA

Authors of DeMIMA, [37], present a multilayered approach for design pattern detection. The tool consists of three layers: two layers to recover an abstract model of the source code, and a third layer to identify design patterns in the abstract model. So it is an approach covering static code analysis.

The tool is implemented on the top of PTIDEJ framework.

MoDeC

Another study from the members of PTIDEJ research team presents also the dynamic analysis approach to detect the behavioral and creational patterns [38]. Firstly, the static analysis tool is used to detect design pattern, then the dynamic analysis is executed to find behavioral and creational motifs in the code.

7 Advanced Design Patterns Automation Tools

An intelligent CASE tool that takes benefit of design patterns to provide a direct help to the engineer in the design of a complex software system, described in [39], can be placed among the advanced design patterns automation. This chapter points out the main principles of this tool and provides also description of the second expert system that helps to the novice developers in design pattern applying during the implementation process.

Bergenti [39] also specified the requirements to the expert system which purpose is to criticize the design patterns application:

1. The system should find automatically pattern realizations to assign a role to the design elements.
2. The system should propose pattern-specific critiques directed to improve the design.
3. The system should suggest alternative realizations of the patterns emphasizing the achievable improvements.
4. The system should propose a design pattern to solve a particular design problem.
5. The system should find recurrent design solutions that can be the base for new design patterns.
6. The system can enhance these processes with learning capabilities.

7.1 IDEA: Interactive Design Assistant

Bergenti [39] proposed the tool IDEA that is a design assistant intended to support the engineer in the design process. IDEA is based on automatic design pattern detection and fulfills the first two cases from the design pattern expert system requirements.

IDEA tool supports design phase of the development process. It analyses model created and provides critiques related to the design patterns application during the user's work with the UML model of a system.

IDEA Expert System Structure

IDEA tool is composed from the following three modules:

- *The input module* purpose is to retrieve the UML model from CASE tool, extract class and collaboration diagram from it and transform these diagram into the standard representation that can be processed by the reasoning engine.
- *The reasoning engine* analyses the system's model retrieved by the input module and for this purpose contains a knowledge base. The loaded system's model is represented as a set of Prolog facts and creates model-dependent part of the knowledge base. The second part of the knowledge base is called model-independent and contains set of patterns that can be detected and design rules

with associated critiques. This part of the knowledge base is represented by Prolog clauses.

- *User interface* is a part of the IDEA that is directly exposed to the user; it is a CASE tool for UML modeling. User interface is not implemented by IDEA, two common CASE tools, Rational Rose or ArgoUML, are integrated with IDEA, so the user can choose the CASE tool according to his preferences.

The input module and the reasoning engine are implemented in Java to support its integration with different CASE tools running on different platforms.

IDEA Expert System Principles

Logic of the patterns detection, reasoning and critiques providing is based on the Prolog, which is one of the first declarative and logic programming languages often used in the expert system implementations.

Knowledge base contains list of detectable pattern described by structure template and collaboration template. As Bergenti [39] explains, the structure template identifies pattern-specific constraints and the collaboration template is used to refine the detection performed by means of the structure template. Then the base contains also set of design rules with the corresponding critiques.

After the UML model is transformed into the Prolog representation, the rules are evaluated and if the related clause is satisfied, IDEA assistant triggers relevant critiques to be displayed to user.

IDEA tool displays to the user two lists, the pattern list with all design patterns detected and the to-do list with the design relevant critiques ordered by their importance. The critiques provide a user with following suggestions:

- Improve the names of classes, attributes and operations.
- Consider changing the access modifier for classes, attributes and operations.
- Operations are likely to be missing in a class.
- Operations are likely to be dangerous for reusability.
- Another design pattern can be used to solve design problems.

7.2 Pattern Advisor

Berdún [40] introduces an expert system enhanced with machine learning techniques that can assist novice software designers during the software implementation process.

The expert system is called Pattern Advisor and its user interface looks like common UML modeling CASE tool, similarly to IDEA. The CASE tool is enriched by active assistance agent for design pattern usage.

The system expects two types of users, novice developers and expert developers. The knowledge base does not include only design patterns catalog, but it stores also knowledge gained by observing an expert developer during his work with the CASE tool. The tool is then able to assist novice developer to apply design patterns like a real specialist. The biggest advantage of Pattern Advisor beside IDEA is that this tool implements machine learning techniques and is able to learn from an expert designer.

Pattern Advisor Knowledge Base Training Principles

More specifically, the knowledge base of Pattern Advisor is formed from the following sources:

1. Pattern catalog as it is defined by [1]. Structure patterns representation is determined by class diagram, and behavioral and creational patterns representation is based on the sequence diagram.
2. Learning about pattern applying by observing expert developer interactions with CASE tool, this kind of knowledge can be called background knowledge, or heuristically gained knowledge.

To capture and update expert knowledge of design patterns applying, Pattern Advisor uses Bayesian Network approach which combines principles from several research areas, e.g. statistics, probabilistic theory, graph theory and computer science.

Berdún [40] explains that Bayesian Network is a probabilistic graphical model of uncertain knowledge in which knowledge is modeled by a directed, acyclic graph where nodes denote random variables and arrows denote dependencies among these variables. This graph then enables to compute the probabilities of specific output variables, in our case design pattern representations, based on the occurrence of other variables assumed as input.

Initially, the Pattern Advisor knowledge base contains only data derived from pattern catalog and just minimum of background knowledge. Bayesian Network is then formed mainly from the knowledge gained from pattern catalog. As the agent monitors the expert user's action in the CASE tools, the probabilities stored in the Bayesian Network get updated with the newly gained information.

Pattern Advisor gains the expert knowledge in following ways:

1. Monitoring of expert developer's behavior during application of a new pattern into the solution while he is not aware of it.
2. Feedback provided by the developer. Advisor allows the developer express his satisfaction with patterns advised and agent updates the Bayesian Network according to it.

Pattern Advisor Expert System Principles

When the Bayesian Network gets well trained, the novice developer can start using Pattern Advisor as an expert system. Agent monitors the novice user's actions within UML diagram and uses its Bayesian representation and computation in order to infer a list with recommended design patterns.

Dialog with advised design patterns and their description can be shown to the user in following cases:

1. Agent initializes conversation. Advice window is triggered by agent when it detects the possible pattern application.
2. User can initialize the conversation with Pattern Advisor by pressing an intended button.

The technological principles used for the Pattern Advisor implementation are similar to Bergenti's system; the core implementation technologies are Java and Prolog. The framework Javalog [41], which enables integration of Java and Prolog for agent-oriented programming, has been used. For the Bayesian Network implementation, the framework JavaBayes [42] has been selected which enables creation and manipulation of Bayesian networks.

8 Conclusions

The article presented the current possibilities and research activities in the area of design patterns automation and how it is supported in CASE tools dedicated for Java developers.

Although the research area examines the development of expert systems which purpose is to advise to inexperienced developers design patterns applying into the software solution, in the praxis the developer still need to have basic knowledge about each design pattern intent and can derive benefits rather from several IDE's feature that save the implementation time.

In the reverse engineering area and for the automatic system documentation, the various code analysis tools focused on automatic design pattern detection can be applied to inspect the system's architecture.

9 Acknowledgement

This work was supported by the project No. CZ.1.07/2.2.00/28.0327 Innovation and support of doctoral study program (INDOP), financed from EU and Czech Republic funds.

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Pattern - Elements of Reusable Object-Oriented Software. 1994
2. Freeman, E., Freeman, E., Bates, B., Sierra, K., Robson, E.: Head First Design Patterns. 2004
3. Dascalu, S., Hao, N., Debnath, N.: Design Patterns Automation with Template Library. 2005
4. Bulka, A.: Design Pattern Automation. 2002
5. Kirasic, D., Basch, D. Ontology-Based Design Pattern Recognition. 2008
6. Martinez, L., Favre, M., Pereira, C.: Integrating Design Patterns into Forward Engineering Processes. 2004
7. Lande, R., Sobin, S.: Reverse Engineering of Computer Software and U.S Antritrust Law. 1996
8. ModelMaker, http://www.modelmakertools.com
9. UMLStudio, http://www.pragsoft.com/prod_umls.html
10. Together, http://www.borland.com/_images/DS-Together_tcm32-206857.pdf
11. Rational Rose Products, http://www-03.ibm.com/software/products/us/en/ratirosefami
12. Rational Software Copyright: Pattern-Oriented Development with Rational Rose. 2001
13. Jakubik, J.: Comparison of CASE tools based on design patterns source code support. 2006. http://www2.fiit.stuba.sk/iit-src/2006/03jakubik.pps [Accessed 02/04/2013]
14. Enterprise Architect, http://www.sparxsystems.com/products/ea/index.html
15. Eclipse, http://www.eclipse.org/platform/overview.php
16. Shinkarenko, I.: Design Patterns Used in Eclipse. Bangalore, Eclipse Summit India 2009
17. PatternBox, http://www.patternbox.com/eclipse-plugin.html
18. Vandyke, C.: A Design Pattern Generation Tool. 2009

19. Design Pattern Toolkit, https://www.ibm.com/developerworks/community/blogs/woolf/entry/the_design_patterns_toolkit
20. Java Pattern Wizard, https://developers.google.com/java-dev-tools/codepro/doc/features/patterns/pattern_wizard
21. NetBeans, https://netbeans.org/
22. AndyPatterns Copyright: UML and Design Pattern Support in Netbeans 6.5. 2013. http://www.andypatterns.com/index.php/blog/uml_and_design_pattern_support_in_netbeans_6_5/ [Accessed 25/04/2013]
23. DPAToolkit, http://dpatoolkit.sourceforge.net/
24. Jakubik, J.: Extension for Design Pattern Identification Using Similarity Scoring Algorithm. 2009
25. Sindelar, S., Zavoral, F.: Design Patterns Support in Development Environments. 2011
26. Ferenc, R., Gustafssony, J., Muller, L., Paakki, J.: Recognizing Design Patterns in C++ Programs with the Integration of Columbus and Maisa. 2002
27. Blewitt, A., Bundy A, Stark, I.: Automatic Verification of Java Design Patterns. 2005
28. Shi, N., Olsson, R.: Reverse Engineering of Design Patterns from Java Source Code. 2006
29. Guéhéneuc, Y. et al.: Ptidej: A Tool Suite. 2005
30. Fujaba Tool Suite, http://www.fujaba.de
31. Niere, J. et al.: Towards pattern-based design recovery. 2002
32. Dong, J., Lad, D. S., Zhao, Y.: DP-Miner: Design Pattern Discovery Using Matrix. 2007
33. Reinersdorff, A.: On Recognizing Design Patterns with Crocopat. 2009
34. Birkner, M.: Objected-oriented design pattern detection using static and dynamic analysis in java software. 2007
35. Tosi, C., Zanoni, M., Maggioni, S.: A Design Pattern Detection Plugin for Eclipse. 2009
36. De Lucia, A.: An Eclipse Plug-in for the Identification of Design Pattern Variants. 2011
37. Guéhéneuc, Y., Antoniol, G.: DeMIMA: A Multilayered Approach for Design Pattern Identification. 2008
38. Ng, J. K., Guéhéneuc, Y., Antoniol, G.: Identification of Behavioral and Creational Design Motifs through Dynamic Analysis. 2009
39. Bergenti, F., Poggi, A.: IDEA: A Design Assistant Based on Automatic Design Pattern Detection. 2000
40. Berdún L., et al.: Assisting novice software designers by an expert designer agent. 2008
41. Amandi, A., Campo, M., Zunino, A.: JavaLog: A framework based integration of Java and Prolog for agent-oriented programming. 2004
42. JavaBayes, http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html

Failures of Outsourcing of Software Development

Aziz Ahmad Rais, Rudolf Pecinovsky
University of Economics, Prague, Faculty of Informatics and Statistics,
Department of Information Technologies
W. Churchill Sq. 4, 130 67 Prague 3
arais@seznam.cz, rudolf@pecinovsky.cz

Abstract. An outsourcing of software development has higher risks of failure than an internal software development. Both the internal and the outsourcing types of development have some common failure types. However, the root cause of the failures, the risks and the solutions of the risks and preventing from the failures are not the same for both types of software development. The goal of this paper is to analyze the outsourcing of software development failure's causes and risks.

1 Introduction

The generally known problems of software developments are lack of business requirements, lack of technical requirements, lock of resources and poor testing 1. There are the software development project's level problems that cause also a project failure for example: a wrong cost estimate, a wrong project scheduling, the technologically unrealistic requirements and a wrong effort estimate 2. The probabilities of these risks are getting higher when a software development is outsourced. Some of the reasons why the outsourcing software development risks are higher than the internal software development are mentioned in article 3.

Also, in the outsourcing of software development we can identify risks on two levels (development and project). The project level risks are already analyzed and there are some recommendations and some processes defined 4.

This paper is mainly about to identify the risks and the problems of software development in outsourcing. The result of analysis of researches in different papers and articles in the following chapter doesn't differentiate between the problems of a software development outsourcing and a software service outsourcing. However software development outsourcing is a core part of a global outsourcing and the result of the researches will show to some extent the problems of software development that I have experienced too.

2 Analysis

In order to understand the need for this analysis it is necessary to understand "why the outsourcing projects fail?" Answering this question requires a very large research

and mainly means to do such a research. Besides that, most of the companies and CIO's have done such researches and some results are available on the Internet in the form of html pages or articles. The subject of this paper is not to do another research to answer the reason of failure of an outsourcing project but rather to concentrate on problem analysis; that is why this question will be answered based on publicly available results of such researches.

Analysis 5 shows that most fear from outsourcing a project was that quality and expectation will be on the nearly the same level as an internal project. The result was that the quality of the outsourced projects was not good. In the research were missing the detailed description of the quality attributes. The research was based on 500 companies and the quality was risen by a new process, where more human resources to the project, training were added.

Research does not show the number of increased resources, but if the reason for outsourcing would be financial advantage in such cases, adding more resources and training will increase the cost.

The quality of a software development is usually measured on three business levels, design and source code levels 3. Business level quality means what business processes are automated and implemented by an application; design level quality means architecture and non-functional requirements, and finally source code quality is about how code is self-descriptive and commented.

Mostly the quality of a design and a source code cannot be checked easily and agreed on the details without proper solution. The source code and the design level quality can be checked and agreed based low level design patterns 6; the source code standard to follow, properly documented source code and so on. Usually in outsourcing IT projects a business process is checked by a user acceptance test and all other steps of quality checks are skipped.

The research 7 shows a long list of risks that cause outsourcing projects failure. The most important risks for a software development process that would be covered by this paper are:

- The products are unclear, the requirement changed continually
- Personnel capacity, resources, or lack of responsibility
- Service quality, such as software maintenance and technical support
- Software test plans

All other risks mentioned in that research are related to the preliminary phases of the outsourcing projects, and cannot be solved within a software development process.

The research 8 is about the general outsourcing risks, there can also be found some technical risks that are directly related to the software development. As technical risks are mentioned integration, interoperability and supportability of systems. All these problems are part of the technical architecture of the software application and they are called non-functional requirements. Another risk identified in this research is Quality Assurance. This one of the most important disciplines in software development as general, and in outsourcing, it is getting even more important as it will be the only way how to verify the software against requirements. This research proves with statistics the correlation between some risks e.g. a quality and a schedule. It means that the quality of the software can be affected with high probability when the project time to delivery is set up shorter than in the reality required. The schedule can be decided as

part of the delivery date already in SLA part of a contract, which belongs to preliminary phase of outsourcing of the software development. This research also shows that statistically all these risks are technical, QA (quality), schedule of project or delivery deadline are higher than 50%. Even though the research had some limitation in sample size, if many authors points to similar risks, the results and risk of the research can be accepted as reliable and applicable with outsourcing of software development.

The following research is done based on software and services outsourced to India. It covers most actual statistic on failure and success of software and service outsource to India 9. This paper takes the statistics from Standish Group and points out that the overall success rate of software projects, which is 32%, partial failure rate is 44% and complete failure rate is at 24%. These statistics are based on 8,000 software projects. Main risks of failure are identified management problems (accounted for 65%) and technical problems (accounted for 35%). Based on Microsoft research spending 5% cost on risk management can increase the successful finishing of projects from 32% to 50%, or even to 70%. But still, research does not show how many projects will be at 50% success and how many projects at 70% success. So in case of 50% success, we have still a lot of risk that even risk management cannot solve it. These percentages are very important to rely on. As the results of this research can be generalized and make conclusion that outsourcing of software development has technical risks and so far there is no methodology recommended reducing or fully removing them.

The research describes that 46% of total of global service outsourcing market goes to India and 65% of the total global software outsourcing market goes to India.

The following research is based on small size projects and the number of projects analyzed is 785,325 from which 437,278 projects are from 2001 till 2008. 348,047 projects are from 2009 to 2012 10. Analysis was made based on variables whether client has skills of outsourcing. For example whether client knew the risk of sourcing or whether the client knew satisfaction score of outsourcing projects. Another variable was whether provider has skills of outsourcing; for example whether provider knew risk of sourcing or whether the provider knew satisfaction score of outsourcing projects. It is shown here that having pervious skills of outsourcing increases the probability of success because you can predict risks. Because the research was based on smaller size projects this implies that the smaller the project, the smaller the risks.

It is analyzed here that the role of intermediaries on success and failure of outsourced projects 11. Analysis is done based on 700 projects from 1989-2009. Analysis was done based on an econometric model and calculated with the likelihood of failure of outsourcing projects. The result of the research is in contrast with expectation, as the result is that the likelihood of a success of outsourced projects without the use of intermediaries 71%, and the success of outsourcing projects with the use of intermediaries is 52%. The reason is, for such result mentioned, that the role of intermediaries makes selection of a vendor more competitive. Further, this paper references other researches that confirm similar percentages of a failure as all the above mentioned papers.

3 Analysis based on experience

This chapter will describe details of possible problems that can happen in the software development outsourcing process. The problems/issues are as following:

1. Process
2. Technical
3. Quality assurance

The bellow detailed analysis is based on my experience with outsourced software development. Beneath are provided a summary of outsourced software problems, in order to understand the three types of problems.

One of my former employers outsourced their e-portal to increase their products sales via Internet. The software was outsourced to an outsourcing company and the implementation was based on proprietary framework that was based on EJB 2.1. Software development's cost around 1M €. When the product was deployed to production, its performance was so poor that it could not even be used in an intranet by a few users from a Call Center. Unfortunately, I do not have more detail about the source code, architecture, design and the documentation of this system. My employer tried another outsourcing company, and product was based on Hibernate, JSF, spring, Mule. The system was handed over to internal team for further maintenance. The first problems we had were poor documentation and the knowledge transfer. The quality of both of these did not meet any specific criteria, hence none of the developers was able to fix/extend or change anything in the system as was expected by company executives.

Technology selection was good to use with light-weight components. However, the problem was that the system had no business logic, no service layer, no common validation of inputs, and the system was not separating sales and marketing data. Marketing data were only for analytical purpose and its volume was five times more than the sales data. The data model was using an entity attribute value (EAV) model. This model was causing queering of the database with entity property as there was a missing column per attribute of entity, so you had to select the whole entity. This was causing big performance problem when there was any selection done on data. This DB model caused that the object oriented representative had multiple layer of wrapper which was decreasing flexibilities, maintainability, and extensibility of the system. Any new change in the system cost twice more than originally expected. The cost of IT was at the end more than half of what company income.

Other example of outsourcing is that there was hired whole team of 24 developers, account manager, and a project manager. Team was remote and the company invested approximately 500 Thousand US Dollars and no project was given to the team. So outsourcing was only a cost and no project. But the team did some internal projects for learning purposes; however the implementation lasted three times longer than the internal team did in any similar project. The problem might be the outsourcing model, than the software development, but the team supposed to deliver software. This is a good example of a process, there was no process agreed upon.

The last example of outsourcing software is that outsourcing company (vendor) development team was twice bigger than the internal team that took over the software.

This software had yet again similar technical problems; in the source code there were a lot of experimental codes, source code was not commented, the application had memory leak, and the performance, these problems were because by poor architecture. The documentation was describing some business functionality that was vague and it was in contrast with implementation, mostly difficult to judge whether a problem is bug or enhancement.

Other problems, like very poor security, extensibility, maintainability, were caused by a wrong design and the component model of the whole architecture. Also, the system did not have clearly separated layers.

4 Conclusion

Based on the above searches, the results can derive that it is very difficult to analyze all variables and identify all risk factors of outsourcing and make conclusion and decide which factors exactly are the most critical root cause of a failure. Different authors were interested in different risks of outsourcing and identified different risks of failures of outsourcing projects. Even though outsourcing of software development is important part of all of the outsourcing projects, none of the above made it clear whether there are any differences between software development outsourcing and software service outsourcing.

Some authors were analyzing inter risk dependencies and therefore can be made other conclusion that not only external events, management and process mistakes can lead to failure of outsourcing project, but also if some issue in the planning of outsourcing project is underestimated can create other types of risk factors for outsourcing projects.

A paper [11] from 2001 analysis issues of outsourcing software development. It means that software development outsourcing history is similarly long like all other outsourcing projects and that is why the failure factors and risk factors identified can be applied on software development outsourcing.

Most of the risk mentioned or identified so far by researches can be categorized into risks that are connected to the initial phase or pre-project phases of outsourcing of software development project. The risks important for this paper are risks of a technical type, which can be divided into risk of an architecture, design, implementation and quality assurance.

All the researches so far did not provide a solution to the technical problems of software development outsourcing. The solutions, like investment in risk management, training, adding additional resourcing to outsourcing projects or multi-sourcing and so on, are good for management of outsourcing project at a high level. However, solving technical problems is not fully possible with risk management ad adding resource. That is why there are still many software development outsourcing projects that fail.

Risk of outsourcing can be divided in many ways, but in this page they will be layered into different phases of outsourcing a project. Every project has multiple phases and phases of software development as it is known from different development methodologies: pre-project activities, analysis, design, implementation, testing and de-

ployment, and knowledge transfer. The pre-project activities can be whole outsourcing process, define outsourcing objectives, sending RFI to identify vendor, Signing contract with vendor, SLA and so on. A complete process of outsourcing is defined by many authors; the difference between each defined process is mainly in details. See example for process definition and details [2].

It is important to summarize all the problems that are important for further detailed analysis. There can be three types of problems identified in the outsourcing of software development: the technical, process and quality assurance problems that can happen in the following phase of outsourcing software development: business analysis, design, implementation, testing, and knowledge transfer.

5 References

1. G. Rajkumar, Dr.K.Alagarsamy: THE MOST COMMON FACTORS FOR THE AILURE OF SOFTWARE DEVELOPMENT PROJECT, Volume 1, No. 11, January 2013 ISSN – 2278-1080, The International Journal of Computer Science & Applications (TIJCSA)
2. Kurt R. Linberg: Software developer perceptions about software project failure: a case study, The Journal of Systems and Software 49 (1999) 177-192
3. RAIS A. A., PECINOVSKÝ R.: Agile outsourcing of Software development. Objekty 2010, Ostrava. ISBN 978 80 7368 890 8.
4. VOŘÍŠEK Jiří, BRUCKNER Tomáš: Outsourcing IS/IT z hlediska zadavatelského podniku. červen 1998
5. Atish Banerjee: Success and Failure of Firms in the IT Outsourcing Industry, June 2005 Massachusetts Institute of Technology.
6. FREEMAN Eric, ROBSON Elisabeth, BATES Bert, SIERRA Kathy: Head First Design Patterns. O'Reilly Media October 2004
7. Jiangping Wan, Dan Wan, Hui Zhang: Case Study on Business Risk Management for Software Outsourcing Service Provider with ISM: Technology and Investment, 2010, 1, 257-266 doi:10.4236/ti.2010.14033 Published Online November 2010 (http://www.SciRP.org/journal/ti)
8. Shereazad Jimmy Gandhi: AN ANALYTICAL CHARACTERIZATION OF OUTSOURCING RISKS, 2010, STEVENS INSTITUTE OF TECHNOLOGY.
9. DONG Meixia, GE Jiping: Experience and Inspiration of Risk Management of India's Software and service Outsourcing, 2012, Management Science and Engineering Vol. 6, No. 3, 2012, pp. 34-38 DOI:10.3968/j.mse.1913035X20120603.Z0660
10. Magne Jørgensen: Failure Factors of Software Projects at a Global Outsourcing Marketplace, Simula Research Laboratory and University of Oslo. P.O.Box 134, NO-1325 LYSAKER, Norway.
11. Ravi Bapna, Alok Gupta, Gautam Ray, Shweta Singh: Analyzing IT Outsourcing Contract Outcomes: The Role of Intermediaries, 2010, Carlson School of Management, University of Minnesota.

Software Process Improvement in small companies

Alena Buchalcevová[1]

[1] Department of Information Technologies, Prague University of Economics,
W. Churchill sqr. 4, 13067 Prague 3
buchalc@vse.cz

Abstract. This paper focuses on Software Process Improvement which can bring business benefits to small companies. Firstly, a current status of software processes in the Czech Republic is stated. Then, the ISO/IEC 29110 standard "Lifecycle profiles for Very Small Entities" is presented as an example of Software Process Improvement initiatives focused on small companies. Lastly, this paper presents the initiatives undertaken by the author towards a diffusion of this standard in the Czech Republic and also their results.

Keywords: Software Process Improvement, standard, small companies

1 Introduction

Today's economic climate forces companies to focus on those projects with an immediate value to the business. IT projects under this pressure have to be made right the first time, on time, and match to customer requirements. According to several surveys [15], [3] the ratio of successful software projects ranges to 60% while the rest is categorized as challenged or failed.

Software Process Improvement (SPI) is a way of improving a status of software development. International standards like ISO/IEC 12207 [2], ISO/IEC 15289 [3], ISO/IEC 15504 [4], and ISO 9001 [5] play an important role in SPI initiatives as companies are willing to show compliance with common business rules. However, it was identified that small companies find it difficult to implement international standards as they do not have enough resources, in terms of number of employees, budget and time [6], [7]. To solve these difficulties, the ISO/IEC 29110 standard "Lifecycle profiles for Very Small Entities" is being developed.

In this paper a current status of software processes in the Czech Republic is stated. Then, the ISO/IEC 29110 standard "Lifecycle profiles for Very Small Entities" is presented as an example of Software Process Improvement initiatives focused on small companies. Lastly, the initiatives towards a diffusion of this standard in the Czech Republic and their results are presented.

2 Current Status of Software Processes in the Czech Republic

There are only few surveys focused on using software development methodologies, standards and tools in the Czech Republic. Survey conducted in 2006 and described in [4] showed the use of agile methodologies and approaches in the Czech Republic was only at the starting line. To find out the current status of software processes in small software companies in the Czech Republic and the support by software tools we conducted a questionnaire survey at the conference WebExpo 2011.

2.1 Research Model and Method

The research objective was to determine to which extent company processes are supported by software systems in software companies, which problems software companies have to address and whether they perceive that a suitable software system would help to improve their software processes. The research was focused on small companies that are prevailing in the field of software development. We define small companies as those with fewer than 25 employees according to definition of the working group WG24 within the ISO / IEC JTC 1 SC7. [17] To perform the research, a questionnaire survey method was used. The questionnaire consisted of three main parts:

- Segmentation questions,
- A section focused on software development methodologies and practices,
- Questions concerning the support of company processes by software systems.

We conducted the questionnaire survey at the conference WebExpo 2011 which took place from the 22nd till the 24th of September, 2011 in Prague and was held by the University of Economics. Overall, 1100 participants signed up mainly from small and medium-sized companies that are engaged in software development primarily within the internet environment. This conference is traditionally the most important professional event of the year for many web developers not only from the Czech Republic.

Table 1 Company size

Company size	Absolute frequency	Relative frequency
freelancer	14	13%
2 – 5 employees	18	17%
6 – 10 employees	16	15%
11 – 25 employees	16	15%
26 – 50 employees	7	7%
51 – 250 employees	25	24%
more than 250 employees	9	9%
Total	**105**	**100%**

Conference participants received the questionnaire in a paper form at registration as well as electronically in e-mail newsletters. 108 participants responded to our

questionnaire survey. Thus, the response rate was 9.8%. 105 of 108 respondents are involved in software development, i.e. 97%. Regarding the company size of our respondents, all 105 respondents replied to this question. Table 1 summarizes the frequencies for each category. Based on the results, we identified the segment of small companies, which includes 64 respondents.

2.2 Selected Research Results

One part of the questionnaire survey was focused on whether and how are methodologies and practices of software development used in companies. The best known and most widely used methodologies were chosen (see Table 2)

Table 2 Answers on the question: Do you use in your company any of following software development methodologies?

Software development methodology	Small companies (60)							
	∑ We use it		We use it very often		We use it occasionally		We don't use it	
	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.	Abs.	Rel.
Scrum	26	43%	12	20%	14	23%	34	57%
Extreme Programming	19	32%	2	3%	17	28%	41	68%
Rational Unified Process	3	5%	1	2%	2	3%	57	95%
Open Unified Process	1	2%	1	2%	0	0%	59	98%
Feature Driven Development	9	15%	4	7%	5	8%	51	85%
Microsoft Solutions Framework	4	7%	2	3%	2	3%	56	93%
Lean Software Development	3	5%	0	0%	3	5%	57	95%
Kanban	4	7%	1	2%	3	5%	56	93%

In the segment of small companies, the methodologies that are used more frequently include Scrum, Extreme Programming, and Feature Driven Development. The most popular is Scrum, which is used very often in one fifth of respondents. But the rate of use of methodologies is overall lower than in the group of all companies. At least one methodology is used by 40 respondents, i.e. 66%. Remaining one third does not use any methodology even occasionally. Very often is at least one of the methodologies used by only 18 respondents, i.e. 30%.
Another question was focused on software development practices usage. Results of the survey showed that modern software development practices are more widely used in comparison with the methodologies. Similar results came from the surveys of Scott Ambler [Ambler, 2009].

8. Je Vaše firma certifikována na některou z norem či standardů?

ISO 9001 Systémy managementu jakosti
ISO/IEC 15504 Posuzování procesů
CMMI

jiné:

The vast majority of respondents think that software processes in their company can be improved by using suitable software development methodologies, practices and software systems, as seen in Figure 6. The results were quite identi-cal with the segment of small companies.

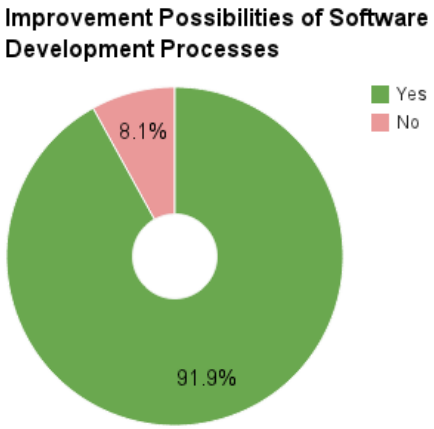


Fig. 1 Answer for the question: Do you think that it is possible to improve the processes of software development at your company by using appropriate software development methodologies, practices, tools or information systems?

3 Software life cycle processes for very small entities

Worldwide conducted surveys [1], [16] indicated that even though very small companies developing software have a significant influence on the economy, most of them do not implement any international standards and models like ISO/IEC12207 [2] or CMMI. Subsequently, these companies have no, or very limited opportunities to be recognized as entities that produce quality software and therefore are often cut off from contracts.

The ISO/IEC 29110 standard "Lifecycle profiles for Very Small Entities" aims at addressing these issues. The term "very small entity" (VSE) was defined by the ISO/IEC JTC1/SC7 Working Group 24 and consequently adopted for use in the emerging ISO/IEC 29110 software process lifecycle standard [15], as being "an entity (enterprise, organization, department or project) having up to 25 people". The 29110 standard consists of five parts as shown in Figure 3 [15]

Part 1 Overview [15] explains main concepts, terms and structure of the standard. Part 2 Framework and Taxonomy [16] presents principles and mechanism of building VSE Profiles. VSE Profile is a subset of the base standard for VSE that is necessary to accomplish a particular function.. Part 3 Assessment Guide [17] defines the process assess-ment guidelines and compliance requirements needed to meet the purpose of the defined VSE Profiles. This part of the standard is used by certified assessors for

VSE assessment. Part 4 Specifications of VSE Profiles [18] provides the mapping to the source standards and is useful for method developers and assessors. Part 5 Management and Engineering Guide [19] is intended for VSEs. The set of 29110 standards was published in 2010. Part 1, 3 and 5, are available at no cost from ISO.
ISO/IEC 29110 is based on existing standards like ISO/IEC 12207 [2], ISO/IEC 15289 [3], ISO/IEC 15504 [4], and ISO 9001 [5].

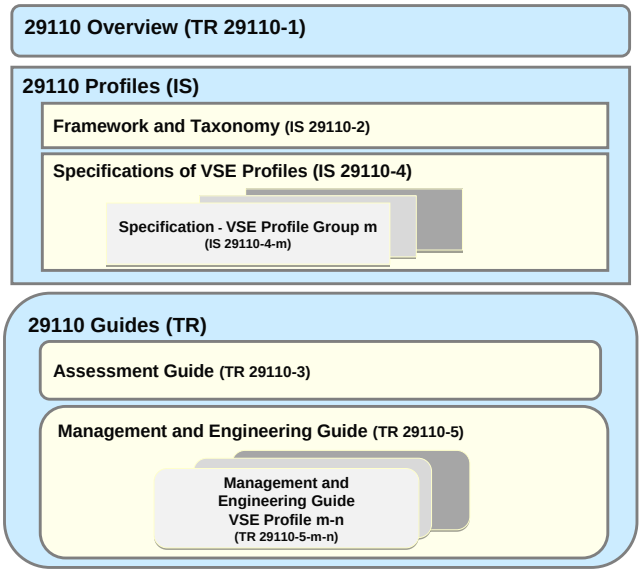


Fig. 2. ISO/IEC 29110 Set of Documents [15]

A key concept of the ISO/IEC 29110 standard lies in development of various VSE Profiles. As a starting point the "Generic" profile group was defined, which is applicable to a vast majority of VSEs that do not develop critical software. Within the Generic profile group four profiles were proposed, i.e. Entry, Basic, Intermediate, Advanced. By using these profiles very small companies have the chance to improve their processes in a clear and stepwise manner. First the Basic Profile intended for a single project with no special risks or situational factors was developed and published. Implementation of the Basic Profile enables VSE to establish good practices for project effectiveness and efficiency, project planning, monitoring, control and configuration management. Besides the project management process also the software engineering process is defined, which encompasses practices for requirements management, analysis, design, construction, verification, validation, and testing.
As some pilot projects of Basic Profile implementation in VSEs showed this profile is still difficult to implement for some companies. That is why the Entry Profile was developed. It applies to six person-months effort or start-up VSEs. Entry and Basic Profiles are published and can be used. Other two profiles are still under development. The Intermediate Profile is intended for VSE, which has more than one project at a time, and therefore is aware of assigning project resources and monitor projects for achieving business objectives and customer satisfaction. Furthermore, VSE at this level needs to define, deploy and improve the organizational standard processes to

achieve similar results in all projects. Therefore, the Intermediate Profile adds to the Basic Profile following processes: Project Portfolio Management, Resource Management and Process Management. In addition, the Advanced Profile is going to supply business management practices.
To help VSE with implementation of the Entry and Basic Profiles, series of deployment packages were developed and offered for free [21]. A Deployment package acts as a detailed methodology that guides company through the process of profile implementation. Typical Deployment package includes process descriptions, activities, tasks, roles and products, templates, checklists, examples, reference and mapping to standards and models, and a list of supporting tools. In order to accelerate the adoption of the ISO/IEC 29110 standard a set of university courses for undergraduate and graduate students was created. [22].

4 The initiatives towards a diffusion of ISO/IEC 29110 standard in the Czech Republic

As results of our survey proved small companies think that by using right methodologies, standards and tools it is possible to improve their software processes. Quality orientated process approaches and standards are maturing and gaining acceptance in many companies worldwide. Use of international standards in companies in the Czech Republic is a key factor for their competitiveness in the global market. Standards emphasize communication and shared understanding. This kind of understanding is not only important in a global development environment; even a small group working in the same office might have difficulties in communication and understanding of shared issues. Standards can help in these and other areas to make the business more profitable because less time is spent on non-productive work. The use of standards has many potential benefits for any organization:

- Improved management of software,
- Schedules and budgets are more likely to be met,
- Quality goals are likely to be reached,
- Employee training and turnover can be managed,
- Visible certification can attract new customers or be required by existing ones,
- Partnerships and co-development, particularly in a global environment, are enhanced.

Seeing the broad adoption of standards in countries in Asia and Latin America companies in the Czech Republic have to move forward in this area. Unfortunately, they are not sufficiently supported by governmental institutions to the effect that standards would be required. However, companies themselves should care about quality of their processes and products. To support the adoption of the ISO/IEC 29110 standard in the Czech Republic I conducted several activities. First, I incorporated this standard into the university courses on the undergraduate and graduate level. With the help of students of the graduate course Software process improvement we prepared local version of the standard and all deployment packages and published them on the

website http://spicenter.vse.cz/. We also translated the wikipedia page about this standard into Czech language http://cs.wikipedia.org/wiki/ISO_29110. The Faculty of Informatics and Statistics of the Prague University of Economics is in the process of building the Center for very small entities in the Czech Republic as a part of the netcenter for VSE – the global net of centers for very small entities. I have also prepared public course about this standard.

5 Conclusions

In this paper selected results of the survey focused on support for software processes in small software companies in the Czech Republic were presented.
As results of the survey showed small companies want to improve their software processes. International standards could be a tool to fulfil this goal and to help with their competitiveness in the global market. The ISO/IEC 29110 standard "Lifecycle profiles for Very Small Entities" was presented as an example of Software Process Improvement initiatives focused directly on small companies. Lastly, this paper presented the initiatives undertaken towards a diffusion of this standard in the Czech Republic.

Acknowledgment

The work reported in this paper was supported by the project IG406013 Software process improvement and software quality assurance supporting company competition.

References

1. Anacleto, A.; von Wangenheim, C.G.; Salviano, C.F.; Savi R.; Experiences gained from ap-plying ISO/IEC 15504 to small software companies in Brazil, 4th International SPICE Con-ference on Process Assessment and Improvement,Lisbon, Portugal, April 2004.
2. Ambler, S.W. Agile Practices Survey Results: July 2009. Ambysoft [online]. Available at: http://www.ambysoft.com/surveys/practices2009.html
3. Ambler, S.W. 2011 IT Project Success Rates Survey Results. Ambysoft [online]. Available at: http://www.ambysoft.com/surveys/success2011.html
4. Buchalcevova, A. Research of the Use of Agile Methodologies in the Czech Republic. In: Barry, C; Conboy, K; Lang, M; et al.(eds) Information Systems Development: Challenges In Practice, Theory And Education. 2009 (51-64) DOI: 10.1007/978-0-387-68772-8_5
5. Deployment Packages repository available from http://profs.logti.etsmtl.ca/claporte/English/VSE/index.html

6. ISO/IEC 12207: Systems and software engineering – Software life cycle processes, 2008.
7. ISO/IEC 15289: Systems and Software Engineering — Content of systems and software life cycle process information products (Documentation), 2006.
8. ISO/IEC 15504: Information technology – Process Assesment, 2004.
9. ISO 9001: Quality management systems — Requirements, 2008
10. ISO/IEC 29110-1, "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) -- Part 1: VSE profiles Overview". Geneva: International Organization for Standardization (ISO), 2010.
11. ISO/IEC 29110-2, "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) -- Part 2: Framework and Taxonomy". Geneva: International Organization for Standardization (ISO), 2010.
12. ISO/IEC 29110-3, "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) -- Part 3: Assessment Guide". Geneva: International Organization for Standardization (ISO), 2010.
13. ISO/IEC 29110-4, "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) -- Part 4: Specifications of VSE Profiles". Geneva: International Organization for Standardization (ISO), 2010.
14. ISO/IEC 29110-5, "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) -- Part 5: Management and Engineering Guide". Geneva: International Organization for Standardization (ISO), 2010.
15. Johnson, J. My Life is Failure. The Standish Group International, Inc. 2006. ISBN 1-4243-0841-0.
16. Laporte, C.Y.; April, A. and Renault, A.; Applying ISO/IEC Software Engineering Standards in Small Settings: Historical Perspectives and Initial Achievements, Proceedings of SPICE Conference, Luxembourg, 2006
17. Laporte, C. (2007) Applying International Software Engineering Standards in Very Small Enterprises. CrossTalk – The Journal of Defense Software Engineering Feb 2007, 29-30
18. O'Connor, R.V., Laporte, C.Y., Towards the Provision of Assistance for Very Small Entities in Deploying Software Lifecycle Standards, 11th International Conference on Product Fo-cused Software Development and Process Improvement (Profes2010), Hosted by LERO, Ireland, June 21-23, 2010.
19. VSE Education Special Interest Group http://profs.logti.etsmtl.ca/claporte/English/VSE/VSEEducation

Petri Nets versus UML State Machines

Hana Kubátová[1], Karel Richta[2], Tomáš Richta[3]

[1] Dept. of Digital Design, Faculty of Information Technology, Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czech Republic
hana.kubatova@fit.cvut.cz
[2] Dept. of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám.25, Praha 1, 118 00, Czech Republic
karel.richta@mff.cuni.cz
[3] Dept. of Intelligent Systems, Faculty of Information Technology, University of Technology in Brno
Božetěchova 2, 612 66 Brno, Czech Republic
irichta@fit.vutbr.cz

Abstract. Petri nets are widely used for the specification of problems, in particular for describing concurrent systems. On the other hand, new versions of the UML specification precisely define the semantics of activity diagrams, and state machines, which can also be used to describe parallel systems. An interesting question is whether we can replace any Petri net machine by the state machine describing the same behavior, and vice versa.

Keywords. Petri Net, UML, State Machine

1 Introduction

Petri nets are widely used for the specification of problems, mostly in parallel systems. New versions of the UML specification language define precise semantics of activity diagrams and state machines. An interesting question is whether we can substitute arbitrary Petri net by a state machine, with an equivalent behavior, and also opposite. We have to distinguish between a state machine and an activity diagram. Activity diagrams are essentially similar to flowcharts. The distinction between state machines and flowcharts is especially important because these two concepts represent two diametrically opposed programming paradigms: the event-driven programming (state diagrams) and the structured programming (activity diagrams or flowcharts). In this paper we will try to describe all these concepts formally, and we also try to formulate transformations between these formalisms.

In our case we are in a situation where we want to describe a network of communicating agents using Petri nets. E.g. we want to define a smart home solution, which consists of a set of agents monitoring and controlling various elements, such as heating, cooling, etc. We are able to describe behavior of these agents by Petri nets, and we have a tool that can simulate designed system. On the other hand, there exist available tools that can convert description of the system by UML state diagrams to the target platform implemented as small embedded systems, or, if it is necessary, to transfer it directly to the hardware implemented using e.g. by FPGA. Our aim, therefore, was the proposal of a universal procedure that can convert any description by Petri nets to an equivalent description using UML state machines. This would have allow us to simulate virtually the design of communicating agents described by Petri nets, and then transform it into an effective solution based on state diagrams.

2 Formal definition and basic terminology

The following formal definition is loosely based on Peterson 1. Many alternative definitions exist.

2.1 Syntax

A ***Petri net graph*** (called *Petri net* by some, but see below) is a 3-tuple (S,T,W), where:

- S is a finite set of *places*;
- T is a finite set of *transitions*;
- S and T are disjoint, i.e. no object can be both a place and a transition;
- $W : (S \times T) \cup (T \times S) \to \mathbf{N}$ is a multi-set of arcs, i.e. it defines arcs and assigns to each arc a non-negative integer *arc multiplicity*; note that no arc may connect two places or two transitions.

The *flow relation* is the set of arcs: $F = \{(x,y) \mid W(x,y) > 0\}$. In many textbooks, arcs can only have multiplicity 1, and they often define Petri nets using F instead of W. A Petri net graph is a bipartite multidigraph $(S \cup T, F)$ with node partitions S and T.

The *preset* of a transition t is the set of its *input places*: ${}^\bullet t = \{s \in S \mid W(s,t) > 0\}$; its *postset* is the set of its *output places*: $t^\bullet = \{s \in S \mid W(t,s) > 0\}$.

A *marking* of a Petri net (graph) is a multiset of its places, i.e., a mapping $M : S \to \mathbf{N}$. We say the marking assigns to each place $s \in S$ a number $M(s)$ of *tokens*.

A ***Petri net*** (called *marked Petri net* by some, see above) is a 4-tuple (S,T,W,M_0), where

- (S,T,W) is a Petri net graph;
- M_0 is the *initial marking*, a marking of the Petri net graph.

2.2 Execution semantics

The behavior of a Petri net is defined as a relation on its markings, as follows. Note that markings can be added like any multiset:

$$M + M' = \{ s \to M(s) + M'(s) \mid s \in S \}.$$

The execution of a Petri net graph G = (S,T,W), can be defined as the transition relation →G on its markings, as follows:

- for any t in T:
  $M \to_{G,t} M' \leftrightarrow \exists M'' : S \to \mathbf{N} : M = M'' + \Sigma_{s \in S} W(s,t) \wedge M = M' + \Sigma_{s \in S} W(s,t)$
  $M \to_G M' \leftrightarrow \exists t \in T : M \to_{G,t} M'.$

In words:

- firing a transition t in a marking M consumes $W(s,t)$ tokens from each of its input places s, and produces $W(t,s)$ tokens in each of its output places s
- a transition is *enabled* (it may *fire*) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff:
  $\forall s \in S : M(s) \geq W(s,t)$.

We are generally interested in what may happen when transitions may continually fire in arbitrary order. We say that a marking M' *is reachable from* a marking M *in one step* if $M \to_G M'$; we say that it *is reachable from* M if $M \to^*_G M'$, where $\to^*_G$ is the transitive closure of $\to_G$; that is, if it is reachable in 0 or more steps.

For a (marked) Petri net $N = (S,T,W,M_0)$, we are interested in the firings that can be performed starting with the initial marking M_0. Its set of *reachable markings* is the set $R(N) = \{ M' \mid M_0 \to^*_G M' \}$. The *reachability graph* of N is the transition relation $\to_G$ restricted to its reachable markings $R(N)$. It is the state space of the net. A *firing sequence* for a Petri net with graph G and initial marking M_0 is a sequence of transitions $\sigma = \langle t_{i1} \ldots t_{in} \rangle$ such that $M_0 \to_{G,ti1} M1 \wedge M_1 \to_{G,ti2} M_2 \wedge \ldots \wedge M_{n-1} \to_{G,tin} M$. The set of firing sequences is denoted as $L(N)$.

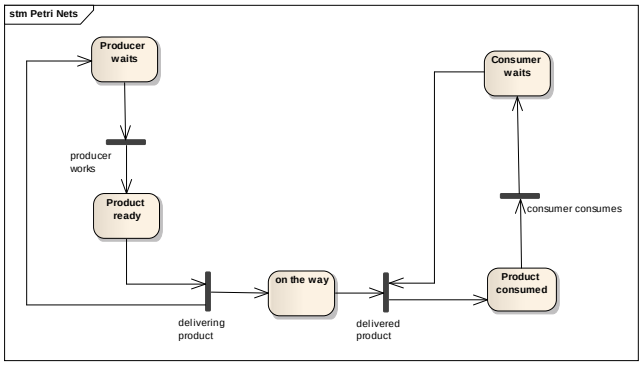


Fig. 1. Petri Net for Producers-Consumers

As an example of a Petri net on the **Fig. 1** there is presented the specification of the classical problem of Producers-Consumers.

3 UML State Machines

UML state machine 3, formerly known also as UML statechart, is an object-based variant of Harel statechart 1 adapted and extended by the Unified Modeling Language.

UML state machines overcome the limitations of traditional finite state machines while retaining their main benefits. UML statecharts introduce the new concepts of hierarchically nested states and orthogonal regions, while extending the notion of actions. UML state machines have the characteristics of both Mealy machines and Moore machines. They support actions that depend on both the state of the system and the triggering event, as in Mealy machines, as well as entry and exit actions, which are associated with states rather than transitions, as in Moore machines.

The term "UML state machine" can refer to two kinds of state machines: *behavioral state machines* and *protocol state machines*. Behavioral state machines can be used to model the behavior of individual entities (e.g., class instances). Protocol state machines are used to express usage protocols and can be used to specify the legal usage scenarios of classifiers, interfaces, and ports.

State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events, that could occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and track the different states of its objects through the system" 3.

3.1 Syntax

The ***UML state machine*** is a 7-tuple $(Q,\Sigma,Z,\Delta,\Omega,q_0,F)$, where in the classic form of a state diagram for a finite state machine it is a directed graph with the following elements:

- *States* Q: a finite set of vertices normally represented by circles and labelled with unique designator symbols or words written inside them;
- *Input symbols* Σ: a finite collection of input symbols or designators;
- *Output symbols* Z: a finite collection of output symbols or designators;
- *Edges* $\delta \in \Delta$ represent the "transitions" between two states as caused by the input (identified by their symbols drawn on the "edges"). An 'edge' is usually drawn as an arrow directed from the present-state toward the next-state. This mapping describes the state transitions, that is to occur on input of a particular symbol. This is written mathematically as $\delta : \Sigma \times Q \rightarrow \Sigma$;
- The *output function* $\omega \in \Omega$ represents the mapping of input symbols into output symbols, denoted mathematically as $\omega : \Sigma \times Q \rightarrow Z$;
- *Start state* q_0: (not shown in the examples below). The start state $q_0 \in Q$ is usually represented by an arrow with no origin pointing to the state. In older texts, the start state is not shown and must be inferred from the text.
- *Accepting states* F: If used, for example for accepting automata, $F \subseteq Q$ is the accepting state. It is usually drawn as a double circle. Sometimes the accept state(s) function as "**F**inal" (halt, trapped) states.

For a deterministic finite state machine (DFA), nondeterministic finite state machine (NFA), generalized nondeterministic finite state machine (GNFA), or Moore machine, the input is denoted on each edge. For a Mealy machine, input and output are signified on each edge, separated with a slash "/": "1/0" denotes the state change upon encountering the symbol "1" causing the symbol "0" to be output. For a Moore machine the state's output is usually written inside the state's circle, also separated from the state's designator with a slash "/". There are also variants that combine these two notations.

For example, if a state has a number of outputs (e.g. "a= motor counter-clockwise=1, b= caution light inactive=0") the diagram should reflect this : e.g. "q5/1,0" designates state q5 with outputs a=1, b=0. This designator will be written inside the state's circle.

3.2 Execution semantics

The behavior of a state machine $A=(Q,\Sigma,Z,\Delta,\Omega,q_0,F)$ is defined as a *transition relation* $\rightarrow_A$ relation on its states, as follows. Let A be in the state q and there are some multiset E of events currently happened. There also can exists a countable number of global variables forming the global state of the machine. The state machine selects all possible transitions and evaluates their guards. Then select one of the resulting set of possible transitions and fires the appropriate change. Firing a transition changes the active states, and runs attached actions defined by the output function ω. The behavior of the state machine is done by the transitive and reflexive closure of $\rightarrow_A$ denoted as $\rightarrow^*_A$. We are generally interested in what may happen when transitions may continually fire in arbitrary order.

4 Example: DFA, NFA, GNFA, or Moore machine

S_1 and S_2 are states and S_1 is an accepting state. Each edge is labeled with the input. This example shows an acceptor for strings over {0,1} that contain an even number of zeros.

4.1 Example: Mealy machine

S_0, S_1, and S_2 are states. Each edge is labeled with "*j* / *k*" where *j* is the input and *k* is the output.

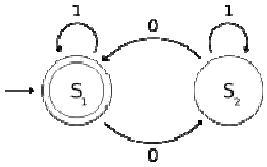


Fig. 2. An example of DFA

4.2 Harel statecharts

Harel statecharts 1are gaining widespread usage since a variant has become part of the Unified Modeling Language. The diagram type allows the modeling of superstates, orthogonal regions, and activities as part of a state.

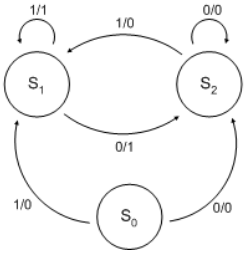


Fig. 3. An example of Mealy machine

Classic state diagrams require the creation of distinct nodes for every valid combination of parameters that define the state. This can lead to a very large number of nodes and transitions between nodes for all but the simplest of systems (state and transition explosion). This complexity reduces the readability of the state diagram. With Harel statecharts it is possible to model multiple cross-functional state diagrams within the statechart. Each of these cross-functional state machines can transition internally without affecting the other state machines in the statechart. The current state of each cross-functional state machine in the statechart defines the state of the system. The Harel statechart is equivalent to a state diagram but it improves the readability of the resulting diagram.

4.3 UML state diagram

The ***UML state diagram*** is essentially a Harel statechart with standardized notation, which can describe many systems, from computer programs to business processes. The following are the basic notational elements that can be used to make up a diagram:

- Filled circle, pointing to the initial state.
- Hollow circle containing a smaller filled circle, indicating the final state (if any).
- Rounded rectangle, denoting a state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activities that are done in that state are indicated.
- Arrow, denoting transition. The name of the event (if any) causing this transition labels the arrow body. A guard expression may be added before a "/" and enclosed in square-brackets:
  (***eventName*[guardExpression]**),
  denoting that this expression must be true for the transition to take place. If an action is performed during this transition, it is added to the label following a "/"
  (***eventName*[guardExpression]/action**).

- Thick horizontal line with either x>1 lines entering and 1 line leaving or 1 line entering and x>1 lines leaving. These denote join/fork, respectively.

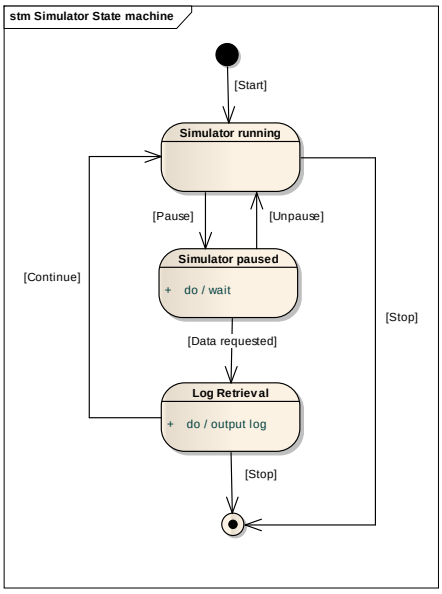


Fig. 4. Example UML State diagram.

5 State diagrams versus flowcharts

Newcomers to the state machine formalism often confuse state diagrams with flowcharts. For a long time, the UML specification wasn't helping in this respect because it used to lump activity graphs in the state machine package (the new UML 2 3has finally separated activity diagrams from state machines). Activity diagrams are essentially elaborate flowcharts.

The figure **Fig. 5** below shows a comparison of a state diagram with a flowchart. A state machine (panel (a)) performs actions in response to explicit events. In contrast, the flowchart (panel (b)) does not need explicit events but rather transitions from node to node in its graph automatically upon completion of activities.

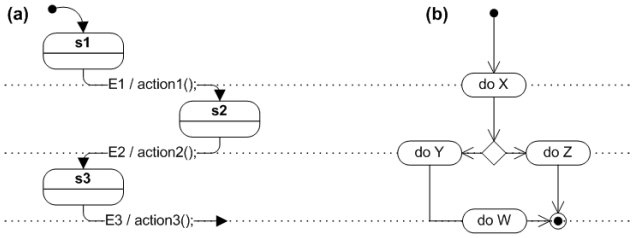


Fig. 5. State machine (a) versus activity diagram (b)

Graphically, compared to state diagrams, flowcharts reverse the sense of vertices and arcs. In a state diagram, the processing is associated with the arcs (transitions), whereas in a flowchart, it is associated with the vertices. A state machine is idle when it sits in a state waiting for an event to occur. A flowchart is busy executing activities when it sits in a node. The figure above attempts to show that reversal of roles by aligning the arcs of the state diagrams with the processing stages of the flowchart.

You can compare a flowchart to an assembly line in manufacturing because the flowchart describes the progression of some task from beginning to end (e.g., transforming source code input into object code output by a compiler). A state machine generally has no notion of such a progression. The door state machine shown at the top of this article, for example, is not in a more advanced stage when it is in the "closed" state, compared to being in the "opened" state; it simply reacts differently to the open/close events. A state in a state machine is an efficient way of specifying a particular behavior, rather than a stage of processing.

The distinction between state machines and flowcharts is especially important because these two concepts represent two diametrically opposed programming paradigms: event-driven programming (state diagrams) and structured programming (flowcharts). You cannot devise effective UML state machines without constantly thinking about the available events. In contrast, events are only a secondary concern (if at all) for flowcharts.

6 Petri Nets versus State Machines

An interesting question is whether we can substitute arbitrary Petri net by a state machine with equivalent behavior, and vice versa. Let us suppose a (marked) Petri net $N = (S,T,W,M_0)$, and we have to construct a state machine with the same behavior. One possible solution can be an extension - to allow arcs to flow from any number of states to any number of states. This only makes sense if the system is allowed to be in multiple states at once, which implies that an individual state only describes a condition or other partial aspect of the overall, global state. The resulting formalism is known as a Petri net.

We will model marked Petri net $N = (S,T,W,M_0)$, by the state machine $A = (Q,\Sigma,Z,\Delta,\Omega,q_0,F)$ and the external memory M_A: $Q \rightarrow \mathbf{N}$. The state machine A has the finite set of states $Q=S$. Each place $s \in S$ will be represented by the appropriate state $q \in Q$. Each transition $t \in T$ will be represented by an appropriate fork or join element $f \in F=T$. If the transition t has one input arc, we will use the fork element, if it has one output arc, we will use join element, otherwise we will use both of them. Arcs from W will be represented by edges from states to forks or from forks to states. Whenever an arc $w(s,t) = n \in (S \times T) \rightarrow \mathbf{N}$, we add an egde δ from the state s into element f (fork or join element), and we add the guard condition for f requiring n tokens at the input state s. Whenever an arc $w(t,s) = n \in (T \times S) \rightarrow \mathbf{N}$, we add an egde δ from the element f (fork or join element) to the state s, and we add the action to f generating n tokens to the output state s.

For the initial marking M_0, we will copy its setting into the memory M_A, therefore the initial state of this memory is duplicate copy of the initial marking: $M_A(s) = M_0(s)$.

We add a start event α to all edges in A. Whenever we generate the event α, all transitions are activated, but only those satisfying their guards are enabled. The state machine determines one of them and fires appropriate transition. The result of this transition is the change of the active state and also change of the external memory M_A - it also changes the set of satisfied guards. We again generate the event α, and the process continues. The start of the machine is realized by the stating state attached to an appropriate state. The included action sets the initial marking into M_A.

The Petri net does not specify the final state. The Petri net stops only if it reaches a state, where no transition can be fired. It could be find out with the help of the reachability graph. In such a case we have to decide, whether this situation represents successful or failure state.

6.1 The Example

Let us imagine the Petri net on the **Fig. 1** for the problem Producer-Consumer. The above conversion gives us the state machine on the **Fig. 6**.

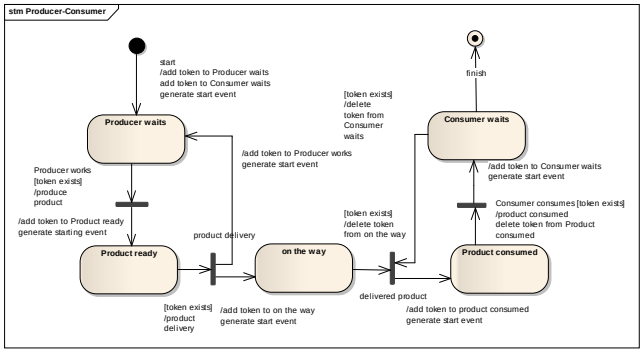


Fig. 6. Producer-Consumer state machine

7 State machines versus Petri Nets

Another interesting question is whether we can substitute arbitrary state machine by a Petri net with equivalent behavior. Let us suppose a state machine $A = (Q,\Sigma,Z,\Delta,\Omega,q_0,F)$, and we have to construct a (marked) Petri net $N = (S,T,W,M_0)$. One possible solution can be an extension - to allow arcs to flow from any number of states to any number of states. This only makes sense if the system is allowed to be in multiple states at once, which implies that an individual state only describes a condition or other partial aspect of the overall, global state. The resulting formalism is known as a Petri net.

We will model a state machine $A = (Q,\Sigma,Z,\Delta,\Omega,q_0,F)$ by a (marked) Petri net $\Phi(N) = (S,T,W,M_0)$, but for the sake of simplicity, and for the purposes of this paper we limit these state machines to only such machines, having only fork and join transitions. The finite set of states Q is simulated by places S, $S=\Phi(Q)$. Each edge $\delta \in \Delta$ will be represented by the appropriate transition $\Phi(\delta)=t \in T=\Phi(\Delta)$. Each transition $t \in T$ represents the appropriate fork or join element. If it is a fork, the transition t has one input arc. If it is a join element, the transition will have one output arc. Arcs from W will represent appropri-

ate edges from states to forks (joins) or from forks (joins) to states. We also add guard conditions to generated arcs. As an example we can apply the above procedure to the state machine from the **Fig. 6**. The result is the Petri Net on the **Fig. 7**.

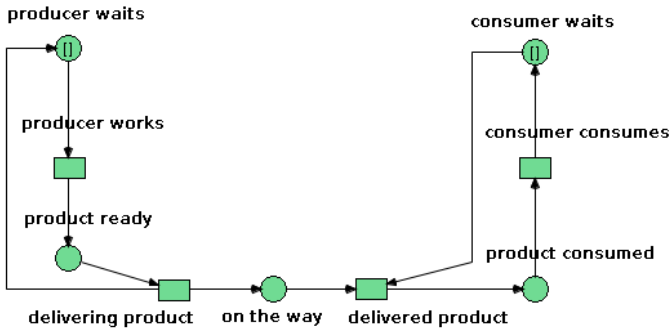


Fig. 7. Resulting Petri Net

8 Conclusions

In the foregoing chapters we have shown that any Petri net can be replaced by an equivalent state machine. So it is the base idea that developers can be freed from Petri nets. It's not strict restrictions - Petri nets can be used where there is a description of the network easier to use than using the state machine description.

The conversion allows the integration of flowcharts within Harel statecharts and Petri nets. This extension supports the development of software that is both event driven and workflow driven.

In the future work we have to try more experiments on more complex examples. We also think of applications in various fields (i.e. medicine, data mining, etc.). This contribution is partially based on the former work presented in the paper 2.

Acknowledgments

This work has been supported by the Ministry of Education, Youth and Sports under Research Program No. MSM 6840770014, and also by the grant project of the Czech Grant Agency (GA_CR) No. GA201/09/0990, and also by the AVAST Foundation.

References

1. Harel, David (1987). Statecharts: A Visual Formalism for Complex Systems. *In: Science of Computer Programming 8 (1987)*, pp. 231-274, North-Holland 1987.
2. Kubátová, Hana – Richta, Karel – Richta, Tomáš: Can Software Engineers be Liberated from Petri Nets? *In: ITAT 2013*, pp. 121, CreateSpace Independent Publishing Platform, ISBN 149095208X. Donovaly, Slovakia 2013.
3. OMG (February 2009). *OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2*. http://www.omg.org/spec/UML/2.2/Superstructure/PDF. *Retrieved 2009-02-15*.
4. Peterson, James L. (1977). *"Petri Nets"*. ACM Computing Surveys **9** (3): pp. 223–252. doi:10.1145/356698.356702
5. Petri, Carl A. (1962). *Kommunikation mit Automaten*. Ph. D. Thesis. University of Bonn (*in German*).
6. Petri, Carl A. - Reisig, Wolfgang: *Petri net*. http://www.scholarpedia.org/article/Petri_net. *Retrieved 2008-07-13*.
7. Wikipedia [Petri Nets]. *Retrieved 2013-10-11*.
8. Wikipedia [UML State Diagrams]. *Retrieved 2013-10-11*.

Case Study of Legacy Systems: Converting and Improvement

Martin Chlumecký

Department of Computer Science and Engineering, Czech Technical University,
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
chlumma1@fel.cvut.cz

Abstract. This work summarizes existing approaches of reverse engineering with focus on procedural origin software; for short legacy software. Today, there is much legacy software that are still used and requested for their modifications which are still required. Our aim is to improve, extend or create a new methodology for converting procedural legacy software into object oriented architecture. We show an instance of legacy software that was written in FORTRAN-77 and our observation that we received during a process of transformation into a new platform. The instance also has been improved about optimizing algorithm and we present notes of improvement process.

Keywords: reverse engineering, legacy software, procedural code, call graph, transformation, FORTRAN

1. Introduction

Software reverse engineering is a very interesting part of software engineering. It is defined as the process of discovering system through analysis of its structure, function, and operation [24]. Today, it is well known that massive computer programs are complex and very difficult to maintain, especially procedural legacy software that includes millions of source code lines. Reverse engineering is usually conducted to obtain missing knowledge, ideas, and design. In some cases, the information is owned by someone who does not want to share them. In other cases, the information has been lost or destroyed.

Legacy software is origin software, a computer system or an application program which has been written in a programming language which is not current. Understanding and modifying code in legacy software is very challenging because it is time consuming and costly. The documentation is usually out-of-date because designers had not maintained it [4]. The main purpose of this level is refactoring and an expandability of the analysed system. Typically, the level is used on a transition to a new technology or a new platform. A secondary purpose is a sustainability of implementation documentation in up to date.

Nowadays, there are many methods of reverse engineering which are used on object oriented architectures because they are very popular and extended. But even today, there are many systems that were written in procedural language and they are

still being used and the requests for modifying and extending legacy software are still involved.

As regards a legal issue, I refer to the book [9] the chapter: “Piracy and Copy Protection”.

1.1 Motivation

This time we get to the main problem and the motivation of this work. How to reverse origin procedural legacy software?

There is still much legacy software that was written in procedural language; e.g.: COBOL, FORTRAN, ADA etc. Each language has been primarily designed for another purpose. The FORTRAN has been designed for scientific computing. Conversely, the COBOL, as the name of the language indicates (COmmon Business Oriented Language), has been designed for commercial and later also database applications.

The aim is to transform legacy software into a new platform or a new technology and take advantages of knowledge that legacy software implements. There are many reasons to do it and I can pick up the most frequent:

1. Hardware restriction – binary files of legacy software are not usually compatible with new hardware architectures. If there is a need to replace a hardware which the original runtime environment (or virtualized) cannot be restored on, legacy software becomes unusable.
2. Software restriction – legacy software often has a limited amount of input and output data. The limit hinders from an effective use. Very old legacy software allows only text input.
3. Software bugs – if a bug is found in legacy software during software life cycle and when a support is not available, there is no possibility how to fix the software.

The documentation is another reason for using the reverse engineering. If the documentation has been lost of any reasons or it is out-of-date, it is necessary to create a new one or to update it. After reversing, a user or implementation documentation is a by-product that is captured e.g.: with UML or another notation.

1.2 Problems

However, reversing of legacy software brings problems which inhibit from easy transformation into a new platform.

Large legacy software are sometimes composed of several hundred thousand lines of source code. This case naturally makes more difficult to separate a code into two categories. The first part represents a code which contains knowledge of legacy software and the second part comprises only code with system routines without knowledge (memory management, management of global variables, etc.).

The next problem is a procedural paradigm of legacy software. If an object oriented source code is reversed then the situation is straightforward because of the code which is composed from elements of object world such as classes, packages etc.

Nevertheless, if we want to transform a procedural code into an object code, we have to determine potential objects which will be used for compilation into a new architecture used in object world.

Legacy software almost has no support. Usually there is no current software documentation which could be used to ascertain the basic architecture and functionality. Mostly there are available only users of the legacy software who naturally have no experience in the field of software engineering thus they can provide only functional requirements.

Very old legacy systems have no graphic user interface and after the transformation it is necessary to design a new GUI so that current users can easily orientate in this new software.

2. Related Work of Reverse Engineering

An analysis of the object oriented source code is much easier than an analysis of the procedural code. In the object source code it is evident to distinguish what is a class and what is a package. This implies that refactoring or a transformation of an object oriented code into other architecture is a straightforward.

Nevertheless, the transformation of a procedural code into an object code is many times more difficult. A procedural code does not contain classes neither packages. The code is only composed from methods and global variables. The structure of a new class is unclear. It is ambiguous which methods and attributes will be used by the class and which function will be performed by the class. Another difficulty is to recognize useful parts of the procedural code and parts not including knowledge of legacy software which will be used in a new system.

In the next section there are described basic methods for analysis of procedural source code.

2.1 Call Graph

In [18] Ryder has described a call graph as a directed graph that represents calling relationships between methods in a computer program. Each node represents a procedure and each edge *(f, g)* indicates that procedure *f* calls procedure *g*. A cycle in the graph indicates recursive procedure calls.

The author of the paper [2] has described a progress how to create a call graph from individual methods of object source code. However, each program has an indeterminate call graph because its structure is depended on input data. A doubt of a call graph structure is described in [14].

Methods described above are from object oriented world, but the method how to create an applicable procedural call graph of legacy software is still missing on the field of reverse engineering, eventhough it is a very important tool.

2.2 Identifying Domain Variables

All knowledge which is hidden in legacy software is defined by its output which is described by its users. In a code it means printing of knowledge variables. These variables are called domain variables. The paper [23] proposes a solution for identifying domain variables automatically from legacy code. Domain variables are captured by Data Dependence Graph (DDG). DDG is created by following four steps [3]:

1. generating the DDG of legacy system
2. identifying pure domain variables
3. identifying all domain variables which has an effect to output domain variables
4. domain variables management

Variables with knowledge data can be identified on the basis of these steps. This brings us to a next problem – the problem of identification of objects in procedural source code.

2.3 Identifying Object in Procedural Source Code

An object is a collection of operations that share a state and its operations determine the behaviours. The shared state is hidden from the outside world and is accessible only to the operations of an object. The object identification is very thorny issue in reverse engineering of procedural code. This issue is described in papers [6, 19 and 5] in the field of legacy software. All papers have very similar approaches that are finally classified according to two dimensions:

1. the ability to identify volatile objects versus the ability to identify persistent objects;
2. the ability to simply separate up a legacy system into objects versus the ability to abstract an architecture (i.e. relations between objects).

An identification of business objects [25] is not pure programming but it can help with first analysis of legacy software. The principle is similar as in [6, 19] but the output is not implementation or analytical objects but objects of business processes.

Sward and Thomas in [21] have chosen another tactic. They have created a set of definitions so called classifications which sort procedural methods into groups. For each group a transformation rule has been deduced which defines a new object formally. This approach can process also user defined data structures.

A creation of object oriented paradigm from a procedural source code brings problems of high level of cohesion and a low level of coupling. Sahraoui in [20] tries by set operations to reduce an impact of a transformed object source code. He uses a genetic algorithm to reduce the impact. The algorithm tries to find as good as possible sequence of set operations.

The article [1] aims with an identification of objects in FORTRAN-77 source code. The output of the method is a modified call graph which contains nodes that represent objects and edges represent the association between objects. The previous work described the identification in the general procedural source code. However, each

procedural language has specific features. These features are not captured in general methods for object identification.

2.4 Graph Transformations

Further useful methods are graph transformations which are used to graph simplification that represent a structure of analysed programs; see the section above. The main problem of a call graph is its complexity. If an analysed program has a large number of source codes then the graph is confusing. Much of the code in legacy software is likely to be redundant, often providing the same or similar capabilities in different subsystems that make up the overall structure [7].

Taentzer in his work [22] describes a mapping of a transformation problem from general model into a graph transformation. It is a common process how to simplify the model without causing a lot of important knowledge information.

In [10] Fahmy describes how graph transformations help to understand and to improve software. He shows different levels of an abstraction and he focuses on software architectures. He has found that extracted architecture has deviated from our mental model of analysed software. There was developed a process how to shift origin architecture and thereby to transform a software architecture into other which is clearer.

In paper [8] there is described in a detail way a tool and a methodics for maintenance of legacy systems that documents have been lost or have not been kept up-to-date with the actual implementation. It focuses only on source code of COBOL language and its code analysis, re-design, and code transformation to an object-based architecture.

3. Approach on Real Instance of Legacy Software

All experiments were performed on a procedural code that was written in FORTRAN-77 in 1980 and contains about 30,000 lines of source code. It was first deployed on mainframe EC 1021 in our country. The software served as a tool for hydrologists in the Institute of Hydrodynamics of the Academy of Sciences of the Czech Republic. It had used legacy software for simulations of hydrological models.

Before the transfer we tried to recompile legacy software by using new compilers of FORTRAN. Unfortunately, the legacy source code contains features of FORTRAN-77 which are not supported in new compilers. Moreover, it would be much time-consuming to modification the legacy code in such way that the new compiler would be able to process it. It would take more time than to rewrite whole legacy software into new programming language. The origin compiled file was subjected to decompilation. However, the EXE file could not be processed in 100 %. Current tools were not able to process the old structure of the file.

The software was transformed into object design and was implemented in C++. The process of the transformation was following:

1. To create the call graph captured by GXL [15]. We have created a simple metamodel of FORTRAN language for the purpose of the analysis and also the structure of the call graph which fulfils the GXL standard.
2. The procedural architecture of legacy software was transformed into the object design.
3. The object architecture was refactored and implemented. The used design patterns simplified the transformed architecture and made it more extensible. Further, a graphic user interface has been designed.
4. The legacy software contained implementation bugs that were fixed in the new software. The bugs have been discovered by users of the legacy software.
5. The new software has been extended by an optimizing algorithm which makes the work faster and more effective.

The most interesting section is item no. 2. In the first instance, it is necessary to convert FORTRAN source code into the target language. In our case, it is the C++ language. This process is allowed by the Objexx F2C++ program. The result of F2C++ is a set of C++ methods without object oriented features.

The next step is to create a single superclass which will encapsulate the methods described above. In other words, it is necessary to compile and to test the functionality of the converted software. The single superclass is very chaotic. The DDG approach will help us to determine which of the methods will be excluded from the single superclass.

To the gradual tuning of C++ source code comes about in the step 3. This means that the single superclass is split into more classes in accordance to object oriented features; e.g. Liskov substitution principle, low coupling and high cohesion. This step is very demanding for testing. The converted software must be tested to verify its functionality after every major refactoring. This step includes a possibility to use design patters.

Great helpers for all these steps are UML diagrams and an architecture captured by various graphs. These graphs help to classify the methods of the single superclass into individual classes according to their functionality.

The purely object part of the work has been situated into the step no. 5. It was only to integrate a new functionality into the existing system. The whole process has been realized by the classical software development through an analysis, a design and an implementation. The analysis related mainly to the study of hydrological models. The design has been made by using UML diagrams and design of the interface – adapter – for the converted and newly designed software plugin. For your information, we state that the converted software has been extended by a genetic optimization algorithm, which streamlines the original hydrological model [26].

4. Experiments and Results

4.1 Call Graph

The structure of the call graph has been inspired by [11]. A doubt that accompanies the generation of the call graph [14] was ignored. The call graph was generated as a full call graph which contains all procedures even the procedures depended on input data. The use of the graph got more difficult by its size because it contains about 120 nodes. The call graph has been refactored by tools which support the work with standard GXL. The structure of the graph was described and published in [27].

4.2 Identifying Domain Variables

The call graph has been simplified by the identification of domain variables. The variables have been detected by methods which print knowledge data into files. The domain variables have helped to determine the path from the leaf to the root. The leaf represents a procedure which prints data and the root is the main procedure. This approach is simpler than the approach described in [14]. The graph has been simplified and has contained about 80 nodes and in addition to that this approach helped to determine which procedures contain knowledge and which not.

4.3 Transform to Object Oriented Architecture

The transformation into object architecture was performed manually because it was necessary to transform only about 12 % of the functionality of the legacy software. Potential objects have been identified by [6]. However, the results have not been satisfying. The method has generated many classes which often contained only an attribute or a method. These classes have been pontificated manually afterwards. The modifications have been realized in Enterprise architect tools where heads of the classes and methods for new objects have been generated. The implementation of individual methods has been manually converted into the newly created classes.

4.4 Design Pattern

The final object oriented architecture has been implemented and tested if it returns the same results as the legacy software. Thereafter it was necessary to refactor the new architecture by design patterns. However, the detection of available design pattern is not applicable. The methods described in [16, 13, 12] are devised for an analysis of object oriented software which has been designed as object oriented but for the converted architecture the methods do not provide good results. This state is probably caused by impurity of the converted architecture. Design patterns used in the new architecture have been identified manually again. The patterns have improved both

the readability of the new source code and the asymptotic complexity of legacy algorithms.

5. Conclusion

The process of converting procedural code into object code described above has induced several findings which should be researched again.

In our example there has been created a single purpose call graph for FORTRAN based on GXL standard that helped to analyse the structure thanks to the tools which support a graph transformations. The future research would be focused on creating a universal call graph which implements GXL standard. This call graph should support all features of the most common procedural languages.

Our example was converted manually. The future research should be focused on formal definition of FORTRAN code transformation into object design by UML diagrams. It would be necessary to create a metamodel of the FORTRAN language and a metamodel of a target object oriented language. The next phase will be to define transformation rules from one model to another one.

Design patterns are one of the major features of object oriented architecture. Nevertheless in the reverse it is effective to use design patterns. For object oriented architecture there exists DTTL language [17] which is a language for a description of design patterns. If it could be already possible to identify design patterns in procedural code, it would help to easier identification of the objects. In our example we have used the design pattern Iterator and the architectural design Pipeline. This design has been detected manually by the call graph and by the paths of the domain variables. If the design patterns could have been described for example as a subgraph of the call graph, it would be possible to recognize a similarity in the call graph thanks to graph patterns and thereby there could be the possibility to estimate potential design patterns.

The experiments have also showed that the identification of the objects is possible by graph operations applied on a call graph. All circles have been extracted from the call graph. All the methods (nodes) lying on the one circle are included into the same package. The tree created by the identification of domain variables determined the methods containing knowledge. The future work should be focused on the graph operations and their impact to the call graph.

6. References

1. ACHEE, B. L.; CARVER, D. L. Identification and extraction of objects from legacy code. In: Aerospace Applications Conference, 1995. Proceedings., 1995 IEEE. IEEE, 1995. p. 181-190.
2. ALI, Karim; LHOTÁK, Ondřej. Application-only call graph construction. In: ECOOP 2012–Object-Oriented Programming. Springer Berlin Heidelberg, 2012. p. 688-712.
3. AUSTIN, Todd M.; SOHI, Gurindar S. Dynamic dependency analysis of ordinary programs. In: ACM SIGARCH Computer Architecture News. ACM, 1992. p. 342-351.
4. CANFORA, G and CIMITILE A, E. 1998. Software Maintenance. In Proc. 7th Int. Conf. Software Engineering and Knowledge Engineering, 478-486.
5. CANFORA, Gerardo, et al. Decomposing legacy systems into objects: an eclectic approach. Information and Software Technology, 2001, 43.6: 401-412.
6. CIMITILE, Aniello, et al. Identifying objects in legacy systems using design metrics. Journal of Systems and Software, 1999, 44.3: 199-211.
7. COMELLA-DORDA, Santiago, et al. A survey of legacy system modernization approaches. Carnegie-Mellon univ pittsburgh pa Software engineering inst, 2000.
8. CREMER, Katja; MARBURGER, André; WESTFECHTEL, Bernhard. Graph-based tools for re-engineering. Journal of software maintenance and evolution: research and practice, 2002, 14.4: 257-292.
9. EILAM, Eldad. Reversing secrets of reverse engineering. Indianapolis: Wiley, 2005, xxviii, 589 s. ISBN 07-645-7481-7.
10. FAHMY, Hoda; HOLT, Richard C. Software architecture transformations. In: Software Maintenance, 2000. Proceedings. International Conference on. IEEE, 2000. p. 88-96.
11. GRAHAM, Susan L.; KESSLER, Peter B.; MCKUSICK, Marshall K. Gprof: A call graph execution profiler. ACM Sigplan Notices, 1982, 17.6: 120-126.
12. GUÉHÉNEUC, Y.-G.; ANTONIOL, Giuliano. Demima: A multilayered approach for design pattern identification. Software Engineering, IEEE Transactions on, 2008, 34.5: 667-684.
13. GUÉHÉNEUC, Yann-Gaël; JUSSIEN, Narendra. Using explanations for design-patterns identification. In: IJCAI. 2001. p. 57-64.
14. HASHEMI, A.; KAELI, D.; CALDER, Brad. Procedure mapping using static call graph estimation. In: Workshop on Interaction between Compiler and Computer Architecture, San Antonio, TX. 1997.
15. HOLT R. C., WINTER A. GXL: Toward a Standard Exchange Format. In Seventh Working Conference on Reverse Engineering. IEEE Computer Society, Los Alamitos. pages 162–171. 2000.
16. METSKER, Steven John a William C WAKE. Design patterns in Java: advanced patterns, processes, and idioms. Upper Saddle River, NJ: Addison-Wesley, c2006, xiv, 461 p. ISBN 978-032-1333-025.
17. Ó CINNÉIDE, Mel; NIXON, Patrick. Program restructuring to introduce design patterns. Trinity College Dublin, Department of Computer Science, 1999.
18. Ryder, Barbara G. Constructing the call graph of a program. Software Engineering, IEEE Transactions on 3 (1979): 216-226.
19. SAHRAOUI, Houari A., et al. Applying concept formation methods to object identification in procedural code. In: Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference. IEEE, 1997. p. 210-218.
20. SAHRAOUI, Houari, et al. Object identification in legacy code as a grouping problem. In: Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International. IEEE, 2002. p. 689-696.

21. SWARD, Ricky E.; HARTRUM, Thomas C. Extracting objects from legacy imperative code. In: Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference. IEEE, 1997. p. 98-106.
22. TAENTZER, Gabriele, et al. Model transformation by graph transformation: A comparative study. In: Proc. Workshop Model Transformation in Practice, Montego Bay, Jamaica. 2005.
23. WANG, Xinyu, et al. Automatically identifying domain variables based on data dependence graph. In: Systems, Man and Cybernetics, 2004 IEEE International Conference on. IEEE, 2004. p. 3389-3394.
24. WATERS, R. G. and CHIKOFSKY, E. 1994. Reverse engineering: progress along many dimensions. In Communications of the ACM, 37, 5, 22-25.
25. WIGGERTS, Theo; BOSMA, Hans; FIELT, Erwin. Scenarios for the identification of objects in legacy systems. In: Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on. IEEE, 1997. p. 24-32.
26. CHLUMECKÝ, Martin. Optimizing of Parameters Soil Moisture Accounting Model (SAC-SMA). In: POSTER 2013: 17th International Student Conference on Electrical Engineering. Praha: Czech Technical University in Prague, 2012, s. 1-6.
27. CHLUMECKÝ, Martin. Practice Use of GXL in Reverse Engineering. In: Objekty 2012. Praha: Vysoká škola manažerské informatiky a ekonomiky, a.s., 2012, s. 84-89.

A NEW APPROACH TO THE PREDICTION OF SOFTWARE PROJECTS - THE DYPREP METHOD

J. Bartoška[1]- J. Doležal[2] – B. Lacko[3]

[1] Czech University of Life Sciences, Faculty of Economics and Management, Kamýcká 129, 165 21 PRAGUE
[2] PM Consulting, Husova 86, 565 01 CHOCEŇ
[3] Technical University of Brno, Facuty of Mechanical Engineering, Technická 2, 616 69 BRNO

bartoska@pef.czu.cz, jd@pmconsulting.cz, lacko@fme.vutbr.cz

Abstract. This article describes the basis and characteristics of the DYPREP method for making predictions about software projects. The method employs finite state machines and Markov chains. These allow the behavioural dynamics of complex software development projects to be captured. It may serve as a means for determining a project development forecast, particularly in terms of milestones and at moments the project state is examined.

1 Introduction

Contemporary software development projects and information system development projects are highly complex and undergoing implementation in the current turbulent global economic environment. What had been an improvisational and, to some extent, chaotic management style has been supplanted by one which [4] employs efficient methods for successful project completion. Contemporary project management, as promoted by the International Project Management Association via the Spolecnost pro Projektove Rizeni v ČR *(Project Management Association of the CR),* requires that flexible project management contain not only quality reporting to the project team on the real state of the project, but also that the project parties and the team deal with the issue of future project development in their situation reports.

Project development prognosis is rather new to the Czech Republic and a number of companies lack sufficient experience with it. In most cases, project prediction is not carried out or is replaced by promises claiming that the final deadline and that planned project costs will be maintained. It need not be emphasized that these promises are very often based upon exaggerated optimism or are pure fantasy.

Another approach may also be encountered in which the project prognosis is refused in principle, justified by the current turbulent environment. Proponents of this approach thus resign themselves to the difficulties that come with project forecasting.

In the western countries, a projection of future project development is required as a matter of course as a part of the common project development report and some methodologies make use of it as a key factor in deciding whether the project should be continued. One example is the State Gate Method.

Projects which focus on introduction and use of information technologies require more intense forecasting because they must take into account the consequences of the rapid development of these technologies and the changes in customer requests with regard to the changes in the current global market.

This article demonstrates that a systematic approach brings positive results in the search for a solution.

2 Basis for and a systematic approach to project forecasting

A proper approach to project management and forecasting must be based upon an evaluation of past project development and take into account potential future circumstances.

Because the conditions influencing the originally planned path to the goal and the goal itself change in the current turbulence, the management team must continuously forecast future conditions during project management to do its work properly.

At the system level, attention must be paid to past states of the project and their impact on both the current and future situation. In current practice, situations arise in which the state of a particular project may be rather poor, with deadlines not met and costs increasing. In spite of this, project managers and members of the project team continue to be convinced that the project will be completed on time while maintaining planned costs. And company management shares these expectations! But no current or past events justify such optimism (think of projects carried out under Czech public administration, e.g., the vehicle registry, payment of social benefits, etc.).

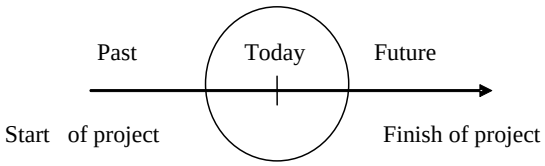


Fig. 1 Three timeframes considered

In the course of these projects, there are characteristic periods (the transition from problem formulation to analysis, from program design proposal to testing, to beta operation) during which significant cyclic deviations from the time and cost values planned occur [11].

Projects which also include program equipment development – today these include automation, informatics and communication projects – are very sensitive to proper allocation of resources, particularly HR (programmers, analysts, testers and implementers), because they use a large number of microprocessor systems. In contrast to, e.g., investment projects, it is usually not possible to eliminate time delays by sufficiently increasing the number of workers.

Project behavioural dynamics are very important for their prediction from a system dynamics point of view (see Šviráková [9]). For forecasting projects with a fairly high amount of randomicity, Markov chains seem appropriate. They have been employed successfully by Kubiš [6] in research into software project risks.

Kubiš's work has shown the importance of using modelling and computer simulation in researching the impact of chance phenomena on project development and project futures by revealing patterns involving the count of random phenomena, the size of necessary reserves and the speed of degeneration of the entire system. Ranzenhofer [7], a PhD student at ÚAI FSI VUT, Brno, carried out research into the use of computer stimulation. The use of statistical methods for prediction was demonstrated by Šlechtová in her proposed use of KVM [5] methodology.

The basis noted above lays the groundwork for a new method, called DYPREP, allowing for advanced prediction of software projects.

3 The DYPREP Method

The DYPREP Method (an acronym for DYnamic PREdiction of Projects) is an advanced method for making forecasts in project development. The initial version was developed at ÚAI FSI VUT, Brno and described in the dissertation of Doležal [8]. Because of the complexity of the method, a detailed description cannot be provided in this article. Only the basic characteristics and principles will be provided here. A detailed description is available in the dissertation indicated and another description with an example and instructions for use is being prepared for publication.

Basic characteristics of DYPREP:

Project State Space DYPREP prediction is based upon defined project states characterized as part of the Earned Value Method (EVM) by means of the CPI and SPI indexes.

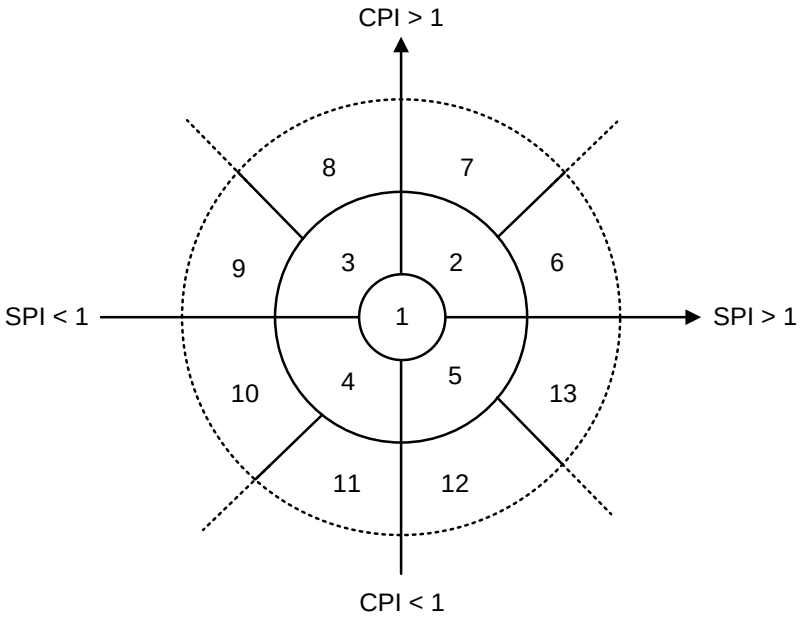


Fig. 2 DYPREP State Space

State transitions Potential transitions between states are expressed in a transition diagram based upon 13 possible states.

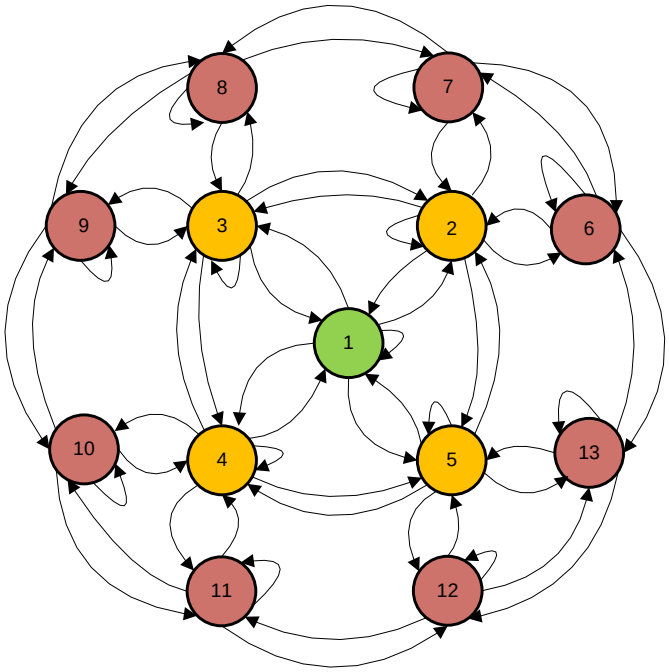


Fig. 3 Project state transitions

Modelling using finite state machines The finite state machine is a formally deterministic entity with input, output and a defined number of finite states. Any transition from a current state to a novel subsequent state depends upon the input character accepted, as well as on the current state of the machine. With adequate precision, such a finite state machine allows changes in the states of software development projects to be modelled.

Using Markov chains to capture the stochastic dependence of transitions between states, captured by a dependency matrix of potential transitions between the states of the project.

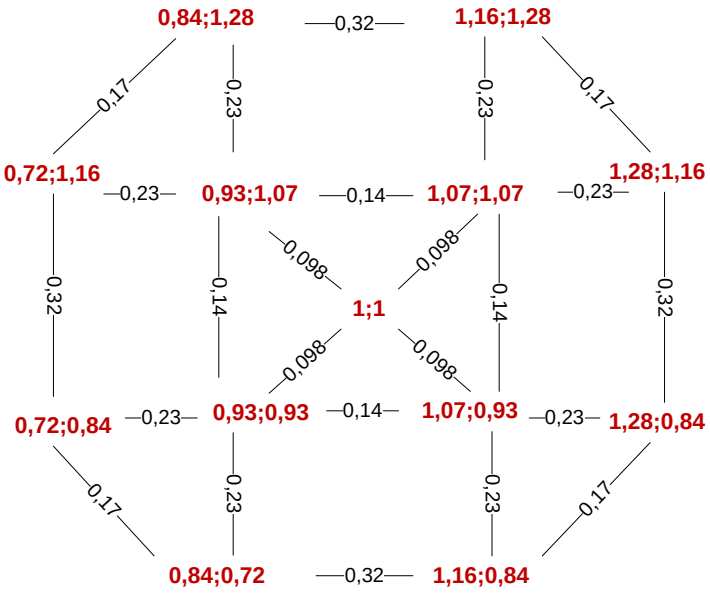


Fig. 4 Probability matrix for individual transitions

Taking the initial state into account This procedure allows the basic model to be adapted to real-world conditions according to the environment in which the project takes place, using the so-called context factor, i.e., for the conditions in the particular IT automation of a company. The context factor is primarily influenced by the size of the organization, the scope of the project, the maturity of the organization, the number of other projects, the quality of work provided by contractors, etc.

Capturing the impact of decision-making entities on the project The targeted impact on future project development is considered in projections of future program development by including the current impact. This may be positive, negative or neutral, and is expressed in the methodology using the so-called direction and power of impact on the project. This expresses the impact of the project manager or project team on the project period being projected.

4 Conclusion

A detailed description of DYPREP would exceed the usual article-length exposition. A text is being prepared for publication describing the method in detail and demonstrating individual steps in particular cases and case studies.

This article aims to draw attention to the method and inform readers of its existence.

To this point, only very little attention has been paid to the issue of quality forecasting of the future development of software and other projects. However, properly forecasting future project development is very important. It is not simply a matter of determining potential deviations from planned costs and deadlines. Timely identification of facts which may signal an upcoming crisis in a project and reacting to the potential crisis also figure in. [2] When a crisis arises, it is important to recognize situations in which it is impossible to eliminate the crisis and is instead more convenient to terminate the project and begin again with a modified version or another project. The current situation in project management requires that project team members and project managers get well acquainted with project forecasting and its methodology. DYPREP is one such methods and may also be used for forecasting software projects.

References:

1. Dujka, J: Cost project prediction of automatic control systems. Ph.D. Thesis, TU of Brno, 2003, 86 p. (in Czech)
2. Ondrejka, R.: Crisis of Projects . AUTOMA. Vol. 17, (2011) No. 1, p. 55-57 (in Slovac)
3. Matouš, V: Earned Value Analysis Method. In: Workshop IPRO2002. TU of Brno 2002, p. 35-44 (in Czech)

4. Mozga, J. - Vítek, M.: Project management and risk management. GAUDEAMUS, Hradec Králové 2002, 268 p. (in Czech)
5. Šlechtová, Y.: Project management method in mechanical company. Ph.D. Thesis. TU of Plzeň, 2001, 154 p. (in Czech)
6. Kubiš, J.: Risk project management of very important risks. In: Proceedings of conference Software Development 2002. TU of Ostrava 2002, p.109-116 (in Czech)
7. Ranzenhofer,T.: Simulation of random event in mechanical proceses. Ph.D. Thesis TU of Brno 2003, 71 p. (in Czech)
8. Doležal, J.: Prediction in projects with Markov chains. Ph.D. Thesis, TU of Brno, 2010, 50 p. (in Czech)
9. Šviráková, E.: Dynamics of projects. Verbum 2011 Zlín (in Czech)
10. on-line http://www.ey.com/ Publication
11. Lacko, B.: New point of view on software life cycle of automatic control (in Czech) In: Proceedings of conference Software Development 2003. Tanger Ostrava 2003, str. 76 – 84

Brief description of software architecture design patterns

Matej Meško[1]

1 University of Žilina in Žilina,
Faculty of Management Science and Informatics,
Univerzitná 8215/1, 010 26 Žilina,
Slovak Republic
mesko@kst.fri.uniza.sk

Abstract. Correctly chosen software architecture design pattern can greatly increase flexibility of the system itself. System can be easily modified through the whole software development cycle. Aim of this paper is to describe software architecture design patterns which divide application to multiple layers and makes them more transparent and maintainable. In the most cases there are three layers: presentation, logic and data layer.

1 Introduction

Many developers, from time to time (rather more frequently than less), hit an issue that was solved many times before and will be many times in the future by the other developers. As programmers are quite lazy creatures the *software design patterns* was invented. Software patterns are templates for solving specific problems in software design solutions and they are easy to modify and transform into code to solve a problem.

This is one of benefits attributed the *objected oriented* programming philosophy and writing *reusable code*. Those software patterns grow up to the *architecture software design* which consists of some basic object hierarchy and rules of behavior. Resulting system will be easier to develop and maintain. It will also be possible to make more parallel work be done. [1][2][3]

2 Software architecture patterns for GUI applications

Principle of the software architectures design is based on dividing application design into separated layer. Basic idea is to divide presentation code and data code from each other and this design is called *separated presentation*. Designs coming from separated presentation usually divide the application into three layers: *data, logic* and *presentation layer*. [4]

Supposed meaning of each layer is as follows (communication between each of individual layers is showed on following (fig. 1) :

- *The data layer* contains objects that make it possible for the rest of the application to do some operations on persistently stored data (objects that encapsulate the database tables, user data, items, etc.).
- *The presentation layer* represents user interface part of the application.
- *The logic layer* represents a mediator between the data and the *presentation layer*. Makes all operations on the data objects (loads them, saves them or edits them) and make them enviable for the user interface (e.g. encapsulate them for easer display in the user interface). Also the events generated in the presentation layer are captured and consumed here.

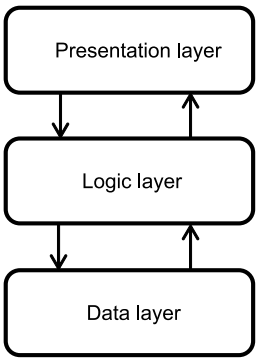


Fig. 1. Scheme of the communication flow in basic model.

Logic takes care of synchronizing state of data in *presentation* and *data* and can change them.

But on background this synchronization can be achieved by various solutions. One of the mostly used is *observer synchronization*. Object called *observer* is checking whether state of observed object is changed. If it is made then *observer* makes some action, usually update something. [5]

Great amount of *presentation* separation is allowed thank to other technique *the data binding*, which allow programmer to strictly separate *view* from *logic*, thanks to opportunity to invoke the calls instead call them directly. [6]

All of those mechanisms and layer division are applied in the models in following sub-chapters.

2.1 Model-view-controller

Very first patter is the MVC (Model-View-Controller). It was invented in 70's at *Xerox Prc.* by *Trygve Reenskaug* as part of a Smalltalk-80. First mention of its existence was published in *"A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80"* [7]. This model splits all application (as *separated presentation* does) so is data logic and the user interface separated. MVC

model has three layers: [8]

- *View* is separated from *model*. *View* visualizes information from *model*. The same information can be showed by various *views* (showing integer array as a table or as a function). Observes the *model* for state change.
- *Model* makes raw data to be in a right format for application and programmer to work with.
- *Controller* is standalone component or combined with *view* in pair and operates with the application logic. Every interaction of the user (mouse move, click, text input, etc.) runs adequate command on a *model*.

Clean MVC model works as follows: *Model* obtains data that application works with. *View* makes visualization of those data from *model* to the screen and observes *model* for state change (if it does, update of screen is executed). User interaction is captured by *controller* which also decides a preformed operation – executes the application logic. So when is a change of field value detected, at the first *controller* is notified. *Controller* runs application logic (e.g. controls input) and makes commit on *model* data. After data change in *model*, *view* updates itself because it observes it (Figure 1). It should be in mind, that each element of the screen vas its own *view* and *controller*. [9]

User interface is always consists of elements – *widgets* (check-boxes, buttons, text areas, text-fields, list boxes, labels, etc.) The MVC model comes with problem. How to change some widget property (e.g. selected item in list, font size, color of back-ground, etc.)? The data about setting of widget property does not fit into *model* definition, remember: *view* visualize raw data in encapsulation of *model*. This type of data is more a matter of *view* state than *model*. Only way to achieve this is in making a compromise of the MVC model purity. This is why there are so many of different descriptions of MVC (they simply contain compromised solution in the description) and misunderstanding of this architecture model. [10] [11]

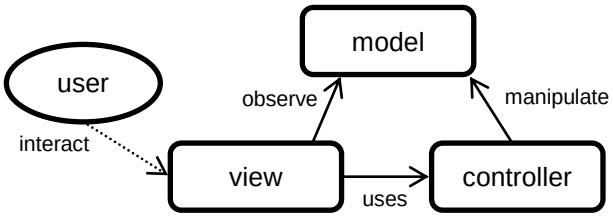


Fig. 2.MVC scheme

2.2 Model-view-presenter

MVP (model-view-presenter) is similar to MVC and solves MVC GUI attribute changing problem. MVP uses *presenter* instead of *controller* in MVC and the main

difference is that *presenter* can manipulate *view*. Whole *view* consists of widgets structure that corresponds to classics form and control model.

If there is a change in field, *presenter* is notified and it retrieves a new value. In next step the *presenter* modify the *model*. Change of *model* state sends notification back to the *presenter*. As the last step, the field GUI attribute can be changed (e.g. font color).

There are two ways how to implement MVP model: *supervising controller* and *passive view*. In *supervising controller* mode can *view* directly access model but only for simple updates, complex ones must be done by the presenter. In *passive view* must be all updates to view made by presenter. [9] [12]

- *View* visualizes information from *model* thought *presenter*. And passes user interaction to *the presenter*. In *supervising controller* mode the view observes *model* for simple changes, complex ones are provided by *presenter*.
- *Model* makes raw data to be in right format for application and program-mer to work with and notify *presenter* of state change.
- *Presenter* makes operation (application logic) based on notification from *model* or *view* and can make modifications to them either.

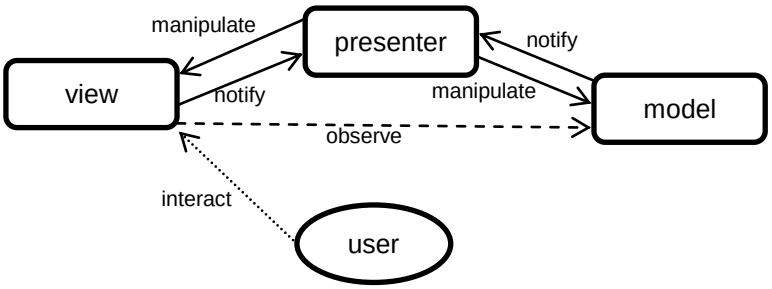


Fig. 3.MVP scheme

MVP in *passive mode* is great for unit testing, because all operation goes thought presenter – contains the most of the presenter logic.

2.3 Model-view-adapter

MVA (model-view-adapter) pattern is similar to MVP. The main difference is in layer separation. *Model* and *view* are completely speared of each and are connected in-line through *adapter*(like passive mode in MVP).

Software pattern *adapter* is mid-interface that can mediate connection between of other interfaces. Illustrated on real-life application (a very favorite demonstration) is power plug adapter (Fig. 3). In the US there is used a different plug than in the Europe so the mid-plug is needed (our *adapter*). [5]

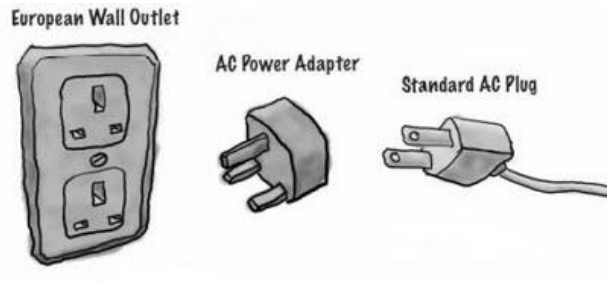


Fig. 4. Adapter real-life demonstration [6]

In MVA, the *adapter* obtains nearly whole application logic. The reason lies in the possibility to change *model*, *view* or *adapter* itself (which comes from *adapterpattern* and the model name). The model should allow:

- Allow to connect various *views* to one *adapter*
- Allow to use various adapters between same *view* and *model*
- Allow to change *model*

The big disadvantage is in complexity of *adapter* itself, which can be enormous because of these possible changes. Functionality is the same as in the passive MVP:

- *View* is separated from *model*, showing data that had adapter filter and simples from *model*. Catches user input and sent it to *adapter*.
- Model makes raw data to be in right format for application and programmer to work with. Make notification of state change only to *controller*.
- *Adapter* catches events from *view* and makes changes to *model*; it is noticed of *model* change and can update *view*.

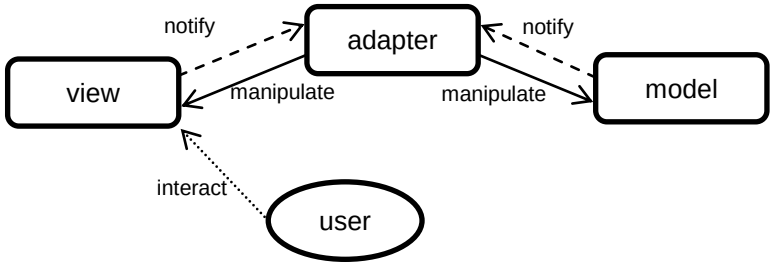


Fig. 5.MVA scheme

2.4 Model-view-ViewModel

MVVM (model-view-view model) is based onthe *presentation model* (known also as *application model*) in which GUI consist of widgets (alike the MVP). Main

difference is that *view* can use data binding to get data or call methods and *theview model*contains logic that is too specific to be included into *view* or *model.*[7]

User interaction is forwarded to *theview model* by *data binding*. Next can be *the model*modified and if it is synchronization notification must be triggered, to make view update changes. [8][9][7]

- View visualizes information from model thought view model. It also sends notification of user interactions by data binding to view model.
- Model makes raw data to be in a right format for application and programmer to work with.
- View model contains only presentation logic and logic that can be included in view or model. Also it notifies the view for update.

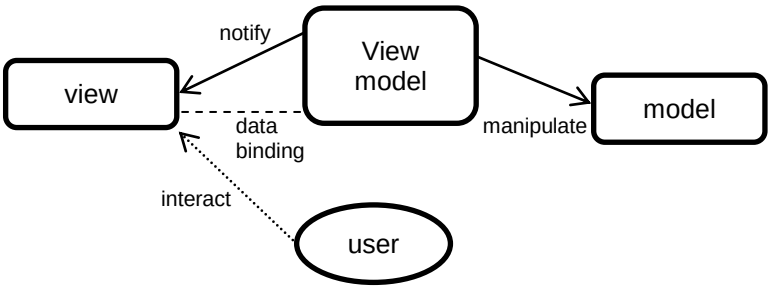


Fig. 6.MVVM scheme

2.5 Presentation-abstraction-control

PAC (presentation abstraction control) is derivate from MVC. PAC creates a hierarchical structures sub PAC elements. Elements communicate with each other through their *control* part. PAC element consists of: [10]

- *Control* is the same as the MVC based *controller*, bud it notify its parent about change.
- *Abstraction* contains data (or only part of the application data structure - abstraction) as MVC *model* do.
- *Presentation* is the same as MVC *view*, displaying a data from the *abstraction*.

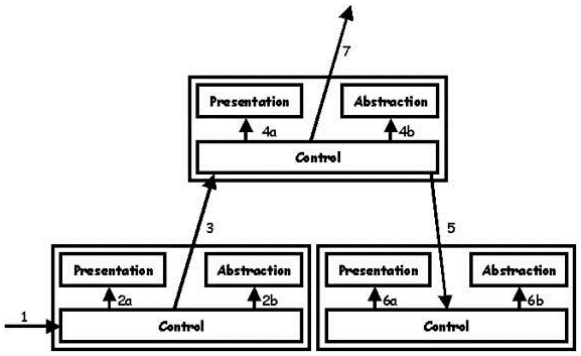


Fig. 7. Notification flow in PAC structure [10]

If is change detected, local PAC element updates its own *presentation* and*abstraction*. After that it sends notification about state change to its parent element. Parent element notifies all its children and waits until their update end and the higher parent can be notified. [17]

Conclusion

In common each of the above described models provides code reuse, greater flexibility and makes application parts more depend-less. This helps to make project code and architecture tided up which is the main point of all models. There is no clear way to comparedescribed architecture design patterns. All I could do was to write down their differences.

MVC if the oldestone and it is suitable for use in *web applications*. In such applications the*view* (web browser) is greatly separated from *themodel* and *thecontroller* (server side). Using MVC makes unit testing harder – elements in this architecture are not all completely isolated.

MVP in *passive mode* is great with using of unit testing - *model* and *view* are isolated. Communication flows thought the *presenter*, which contain presentation logic. Example of using MVP could be the *WinForms* library.

MVA is usable in case of instantly changing and adapting application. Benefit is in creating *adapter* that can control more *views and* use more variants of *model,* which can cause big complexity of the*adapter* code. Unit testing is supported because of element separation.

MVVM is very good with using of unit testing too. It was created for use in the WPF library, targeting to write less code and use more of data binding.

Main difference of PAC to the other described models is that PAC component is “numb” and power of this solution depends in PAC layering. Typical application for using this model is monitoring system (train, car or aircraft monitoring system) where PAC component can act as an agent.

ACKNOWLEDGEMENT

This contribution/publication is the result of the project implementation:

Centre of excellence for systems and services of intelligent transport II.,

ITMS 26220120050 supported by the Research & Development Operational Programme funded by the ERDF.

"Podporujeme výskumné aktivity na Slovensku/Projekt je spolufinancovaný zo zdrojov EÚ"

References

1. C. Zhang a D. Budgen, „What Do We Know about the Effectiveness of Software Design Patterns?," rev. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 2012.
2. R. H. R. J. Erich Gamma, Elements of Reusable Object-Orented Software, AddisonWesley, 1995.
3. R. M. H. R. P. F. Buschmann, Pattern-Oriented Software Architecture – A System of Patterns, Chichester, UK: John Wiley and Sons, 1996.
4. M. Flower, „Separated Presentation," 2013. [Online]. Available: http://martinfowler.com/eaaDev/SeparatedPresentation.html. [Cit. 2013].
5. oodesign.com, „Adapter Pattern," 2013. [Online]. Available: http://www.oodesign.com/adapter-pattern.html. [Cit. 2013].
6. A. K. Singh, „Adapter Pattern," 2013. [Online]. Available: https://sites.google.com/site/akstechtalks/tech-topics/Architecture/patterns/implementation-patterns/structural-patterns/adapter. [Cit. 2013].
7. J. Gossman, „Introduction to Model/View/ViewModel pattern for building WPF apps," 8 11 2005. [Online]. Available: http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx. [Cit. 10 2013].
8. M. Fowler, „GUI Architectures," 18 6 2006. [Online]. Available: http://martinfowler.com/eaaDev/uiArchs.html. [Cit. 10 2013].
9. E. v. d. Valk, „The difference between model-view-viewmodel and other separated presentation patterns," 14 8 2009. [Online]. Available: http://blogs.msdn.com/b/erwinvandervalk/archive/2009/08/14/the-difference-between-model-view-viewmodel-and-other-separated-presentation-patterns.aspx. [Cit. 10 2013].
10. G. s. architectures, „Presentation-Abstraction-Control," 2013. [Online]. Available:

http://www.dossier-andreas.net/software_architecture/pac.html. [Cit. 2013].

11. M. Fowler, „Observer Synchronization,“ 9 2004. [Online]. Available: http://martinfowler.com/eaaDev/MediatedSynchronization.html. [Cit. 2013].
12. Microsoft, „Data Binding Overview,“ 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms752347.aspx. [Cit. 2013].
13. G. Krasner a S. Pope, „A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80,“ *JournalOfObjectOrientedProgramming (JOOP),* 1988.
14. I. Cunningham & Cunningham, „Model View Controller History,“ Cunningham & Cunningham, Inc., [Online]. Available: http://c2.com/cgi/wiki?ModelViewControllerHistory. [Cit. 2013].
15. H. Mcheick a Y. Qi, „Dependency of components in MVC distributed architecture,“ *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference,* pp. 000691 - 000694, 2011.
16. T. Reenskaug a J. O. Coplien, „The DCI Architecture: A New Vision of Object-Oriented Programming,“ March 2009. [Online]. Available: http://www.artima.com/articles/dci_vision.html. [Cit. 2013].
17. Microsoft, „Model-View-Presenter Pattern,“ 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff647543.aspx. [Cit. 2013].

Recognition a structure in Java source code by modified graph matching algorithm

Tomáš Bublík[1]

Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Trojanova 13, 120 00 Praha 2, CZE,
WWW home page: http://www.fjfi.cvut.cz

Abstrakt There exist a lot of techniques to detect a code pattern in a program. This paper introduces a tool which is able to recognize both the properties of a code snippet and even its structure. This tool uses the Sripthon language to describe a snippet. A syntax tree is created from a Scripthon source, and it is compared with a given Java source code. Many optimizations take place during the tree matching process. In conclusion, this tool offers a programmable searching while preserving an acceptable speed. Therefore, the complete recognition process is very fast, and can be used to scan the large programs.

Keywords: Scripthon, Java, abstract syntax tree, Java compiler API, recognition, compiler

1 Introduction

In case of a growing project, on which a lot of developers working, soon or later occurs the situation, where is it necessary to start with a codebase maintenance. Programs consisting of a large source code are becoming chaotic, and many times described illnesses start to appear (code duplicity, weak reusability, etc.). It becomes necessary to maintain the source code somehow. There exist a lot of tools and ways of how to approach this issue. In this paper described tool serves to a programmable Java source code scanning. This tool is based on the Scripthon language which was developed for these purposes in scope of this paper. In this language, a script which describes a source code structure and its properties can be written. Next, an abstract syntax tree (hereinafter AST) is created dynamically from this script. On the other side, a similar tree is created from a Java source, and these two trees are matched by a graph matching algorithm. However, not only the shapes are compared, also the trees properties are considered. To obtain the results faster, several graph optimizations are used during this process. It is possible to scan a high amount of the classes of a source code during a relative short time.

2 Graph matching

A graph is defined as a four-tuple $\mathtt{g} = (\mathtt{V}, \mathtt{E}, \alpha, \beta)$, where V denotes a finite set of nodes, $\mathtt{E} \subseteq \mathtt{V} \times \mathtt{V}$ is a finite set of edges, $\alpha : \mathtt{V} \to \mathtt{L_V}$ is a node labeling function,

and $\beta : \mathtt{E} \rightarrow \mathtt{L_E}$ is an edge labeling function. $\mathtt{L_V}$ and $\mathtt{L_E}$ are finite of infinite sets of node and edge labels, respectively. All the graph in this paper are considered to be directed.
A subgraph $\mathtt{g_s} = (\mathtt{V_s}, \mathtt{E_s}, \alpha_\mathtt{s}, \beta_\mathtt{s})$ of a graph $\mathtt{g}$ is a subset of its nodes and edges, such that $\mathtt{V_s} \subseteq \mathtt{V}, \mathtt{E_s} = \mathtt{E} \cap (\mathtt{V_s} \times \mathtt{V_s})$
Two graphs $\mathtt{g}$ and $\mathtt{g'}$ are isomorphic to each other if there exists a bijective mapping $\mathtt{u}$ from the nodes of $\mathtt{g}$ to the nodes of $\mathtt{g'}$, such that the structure of the edges as well as all node and edge labels are preserved under $\mathtt{u}$. Similarly, an isomorphism between a graph $\mathtt{g}$ and a graph $\mathtt{g'_s}$ of a graph $\mathtt{g'}$ is called subgraph-isomorphism form $\mathtt{g}$ to $\mathtt{g'}$.

A tree is connected and undirected graph with no simple circuits. Since a tree can not have a circuit, a tree can not contain multiple edges or loops. Therefore, any tree must be a simple graph. An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.
The two graphs matching problem is actually the same as the finding the isomorphism between them. Moreover, matching the parts of a graph with a pattern is the same challenge as the finding the isomorphic subgraph.
There are many approaches to this topic [1]. Subgraph isomorphism is useful to find out whether a given object is part of another object or even of a collection of several objects. The maximum common subgraph of two graphs $\mathtt{g}$ and $\mathtt{g'}$ is the largest graph that is isomorphic to a subgraph of both $\mathtt{g}$ and $\mathtt{g'}$. Maximum common subgraph is useful to measure the similarity of two objects. Algorithms for graph isomorphism, subgraph isomorphism and maximum common subgraph detection have been reported in [2], [3], [4] and [5].

3 Scripthon language

For the effective source code patterns description, it was developed the Scripthon language. This language has been constructed iteratively according to the needs which arised together with demands on a source code scanning process.
Scripthon is a dynamically typed, interpreted, and non-procedural language. Its translation into a tree-expressing form and its usage is very similar to the usage of other modern script languages. The language was formed on the basis of the references [6–13]. Because the language is designed to be a scripting language only, there are no special constructions starting a script. Neither this language is pure object-oriented. An input for a compiler is a text with a sequence of commands. This sequence describes consecutive statements in a Java source code. The commands with a variable detail degree correspond to a variable length code segment. The detail level is not fixed and can vary in every command. One command can correspond to a line of a source code; other one can describe the whole class in Java. The Scripthon structure is very close to other contemporary dynamic programming languages. The individual commands are separated by lines. There is no command separator in Scripthon. Inner parts of blocks are tab

nested. A block is not delimited by any signs; just a hierarchy of tabulators is used. An example:

```
Block() a
   Loop(Type=while)
```

The complete definition of the Scripthon language semantics is beyond the scope of this paper. It can be found in [19].Therefore, only the commented building blocks of the denotational semantics and syntax will be given. Syntactic domains:

$$
\begin{aligned}
n &: Num\,(\,numerals\,)\\
V^{+} &: Lang\,(\,language\,)\\
x &: \,variable\,names\\
a &: Aexp\,(\,arithmetic\,expressions\,)\\
b &: Bexp\,(\,boolean\,expressions\,)\\
Str &: structures\\
SAtr &: structure\,attributes\\
AVal &: attribute\,values\\
S &: statements\\
D &: declarations
\end{aligned}
$$

Sets definition:

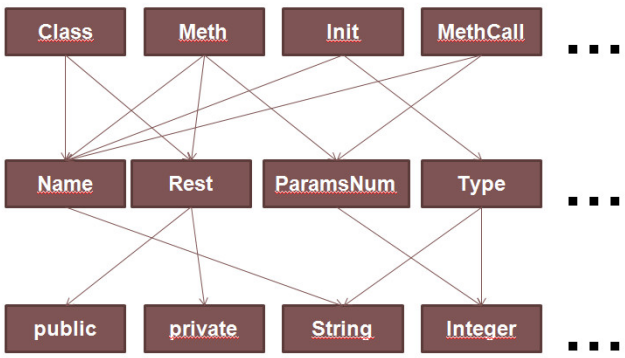
$$
\begin{aligned}
digit &::= 0\,|\,1\,|\,\ldots\,|\,8\,|\,9\\
numeral &::= <digit> \,|\, <digit><numeral>\\
integers &::= \mathbb{Z} = \{\ldots,-2,-1,0,1,2,\ldots\}\\
V &= \{A,B,\ldots,Z,a,b,\ldots,z\}\\
\mathbb{N} &= \{1,2,3,4,\ldots\}\\
W &= \{a^{n}\}_{i=1}^{n}\,\forall a_i \epsilon V\\
V^{*} &= \{w_j\}_{j=1}^{m}\\
e &\ldots empty\,word\\
V^{+} &= V^{*}\backslash\{e\}\\
x &\;\epsilon\; V^{+}\backslash\{Str\}\backslash\{SAtr\}\backslash\{Aval\}\backslash n\\
\mathbb{B} &= \{true,false\}\\
Str &= \{Meth,Init,Block,Class,\ldots\}\\
SAtr &= \{Name,Lenght,Rest,Val,\ldots\}\\
AVal &= \{public,private,\ldots\}\\
AVal &\subseteq V^{*}
\end{aligned}
$$

$$AVal \subseteq Num$$

The first few sets are similar to usual definitions. Therefore, a comment is not needed. The interesting domain is "Str". This domain specifies the structural keywords. It is the set of words, starting with a capital letter, which corresponds to every source code structure in Java. For example, Meth, Init, Block, Stmt correspond to a method, a variable initialization, a block of code, and a statement in the same order. The next domain "SAtr" contains the structure attributes. The Java code structure can have a lot of attributes. For example, an attribute, or a method can have a scope, a variable can have a name and type, and so on. Therefore, an examples of the set "SAtr" could be: Scope, ParamType, Type, etc. The last interesting set is AVal. This set contains the values of the structure attributes. The typical values for a scope are "public", "private" and "protected". On the other hand, this set can contain even other text values. Most often, these values are the names of the methods, variables and its values. The usage in a program can look like this:

```
Init(Name=increment)
```

It means that a variable is initialized (Str) with an attribute "Name" (SAtr), and the value of the attribute is "increment" (AVal). All three sets are linked with a dependency graph. This graph determines which specific structures are able to use. Moreover, it determines the structure attributes and its values. While compiling, the source code is check if it complies the requirements of the dependency graph. However, in fact, the dependency graph is much larger, a small example as follows: The Scripthon compiler is written in Java and does



Obrázek 1. Dependency graph

not pass all the phases of the compilation process. If the compilation would be

complete, the last stage of the process were the instructions in the form of a byte code for a language interpret. In this case, however, the compilation ends with an AST. The next steps are not necessary, because an AST is used in the matching phase. The phases of the Scripthon compiler are as follows:

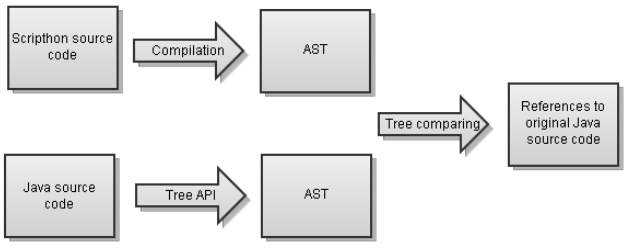
- Tokenization: The particular text elements (including spaces and spacing) are divided into the so called tokens here. The tokens are buckets which contain significant text elements for translation.
- Syntax checking: The next step is the syntax control. In this step, all tokens are checked whether its content can be interpreted, and if its sequence is correct.
- Blocks of command dividing: Thanks to the given dividers, the tokens sequences can by divided into the semantic units called statements.
- Semantic analysis: In this step, it is checked whether the sequence of the nested statements or the consecutive statements corresponds to the given semantics.
- Creation of an AST: The last stage is creation of an AST from the particular statements.

3.1 Complete flow

This paper deals with a programmable searching of a Java source code. It differs from a common textual searching or a searching with regular expressions by the ability of describing even a structure of a source code and its properties. Within this work, a new programming language named Scripthon, and capable of this properties, was developed. With help of this language, it is possible to describe a code structure with properties, and it is even possible to change the properties of the searching sample in dependence of properties of the searched segment.
An abstract syntax tree, which corresponds to a given description, is built from this language, and is compared with syntax trees created from a given Java source code. Basically, in the end, it is the application of a sub-graph matching algorithm where many optimization tools take place. Results of the described process are the references to a given original Java source code.

4 Optimizations

While browsing a Java source code, the tree with the nodes enhanced by four numbers is created. These numbers are the natural numbers named left, right, level and level under. The first and the second number (left, right) denote the order index of a node in the tree preorder traversal. Therefore, an ancestor's left index is always smaller than its children left index, while the right index is always bigger than any children's right index. The level number denotes the level in a tree hierarchy of vertices, and the level under number denotes a number of levels under the current node (compare with the method described in [14]). Suppose that `x` and `y` are two nodes from a tree.



Obrázek 2. Scripthon chart

- The `y` node is an ancestor of `x` and `x` is a descendant of `y` if `y.left` < `x.left` < `y.right`
- The `y` node is an parent of `x` and `x` is a child of `y` if 1) `y.left` < `x.left` < `y.righ` and 2) `y.level` = `x.level` − `1`

All these data are acquired during a single pass through the tree. Obtaining this information is not a time consuming operation, because it is made during the tree production process. On the other hand, the number of comparisons can be significantly reduced with these numbers. Moreover, while comparing the trees, it is very easy to detect:

- How many elements have a given structure
- Whether a node is a leaf
- How many sub-statements are included in a given structure

The comparison of two trees is much more time consuming without this information. In summary, this information is used in cases where the shape of the given structures and its coupling is considered more than its properties.
A line reference to the source code is important information which is also added to the tree as a metadata. Therefore, it is easy to link the results with the original source position and show it to the user. There are some more elements in a node metadata. For example, some of the other metadata information is a filename of the source file.
Because the number of the comparisons is a key indicator for the algorithm speed, it is necessary to keep the number of nodes as small as possible. Therefore, only the supported structures and its properties are considered while creating a tree from the source code. Thus, the same Scripthon definition set is used during the tree creation process. Other elements are omitted.

5 Tree matching

The simple and many times described backtracking algorithm is used for the graph matching. Basically, it is the problem of finding an isomorphic tree to the given tree from a large database of trees. Comparing to the common tree matching, there are two differences. The first one is that the node properties need to be considered during the process. The second difference is that not every Scripthon node corresponds exactly to one Java structure node. For example, the already mentioned keyword Any() could correspond to more nodes.
The source trees are created from the corresponding classes. The classes and the trees are mapped one-by-one. Each tree corresponds to exactly one class. In the first step, the algorithm checks whether the shape of the structure match, and then the properties are compared. This is because the properties matching is much more time consuming operation than shape detection. Many structures are eliminated very quickly from the process in the case that the shape does not fit. If the shape of the structure corresponds to the required shape, the structure parameters are compared. All the parameters of a given node must be met. The node properties are provided by the Java compiler.
Many aspects are considered during properties matching process. Not only keywords and Java nodes properties are considered. According to the previous section, it is possible to exclude quickly the mismatched parts, because some additional data are known about a shape of the sub-tree.
The typical size of a class graph depends on the source size and on the number of supported structures. About 80 nodes of the graph are created from a Java class with length about 200 lines nodes in the current version of Scripthon. In future versions, when more structures will be supported, may the number of the nodes significantly increase.
Unfortunately, because all the Java classes with all their nodes must be compared with all the Scripthon statements, the number of complexity rapidly grows. According to [15], the sub-graph isomorphism problem has $O(N^3)$ complexity in worst case. Since the number of occurrences can be more than one, each class must be browsed more than once. Each class needs to be traversed until the number of results is 0. According to [16], [17], the graph isomorphism problem is polynomial. Therefore, even in this case, the complexity of our algorithm remains polynomial. On the other hand, with the above outlined optimizations, the number of node comparisons is significantly decreased. More on the similar graph matching techniques can be found in [18].

6 Current Work and Preliminary Results

The described way of scanning a source code is very effective. Thanks to the mentioned optimizations, it was achieved a significant acceleration of the whole process. In the initial attempts without optimization where the method one-by-one was used, the results were not satisfying. The tests were performed on the various kinds of programs with hundreds of classes. The first results were

fluctuating around tens of seconds. With the current optimization implementation, the performance of the searching algorithm applied on the same programs moved to up to units of seconds. The accurate results are difficult to measure, because each program is different, and it depends on the number of hits. However, it can be said that a big improvement can be seen. This is because the fact that thanks to additional information, it is possible to exclude high number of the non-matching sub-trees, and the significant number comparisons can be saved. Roughly speaking, the searching process is transformed into the exclusion method: exclude as quick as possible as many non-matching results.

Reference

[1] Irniger, Ch-A. M.: Graph Matching - Filtering Databases of Graphs Using Machine Learning Techniques. ISBN 1-58603-557-6. 2005.

[2] McKay, B.D.: Practical graph isomorphism. In Congressus Numerantium. volume 30, pages 45-87, 1981.

[3] Ullmann,J.R.: An algorithm for subgraph isomorphism. Journal of the Association for Computing Machinery, pages 31-42, 1976.

[4] Levi, G.: A note on the derivation of maximal common subgraphs oftwo directed or undirected graphs. Calcolo, page 341-354, 1972.

[5] McGregor, J.: Backtrack search algorithms and the maximal common subgraph problem. Software-Practice and Experience, pages 23-34, 1982.

[6] Krishnamurthi, S.: Programming Languages: Application and Interpretation. Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License Version, 2007.

[7] Pierce, B. C.: Types and Programming Languages. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, ISBN 0-262-16209-1

[8] Grune, D., Jacobs, C. J. H.: Parsing TechniquesSecond Edition. VU University Amsterdam, Amsterdam, The Netherlands Published by Springer US, ISBN 978-1-4419-1901-4, 2008.

[9] Steedman M.: The Syntactic Process. MIT Press, 2000.

[10] Selinger P.: Lecture Notes on the Lambda Calculus. MIT Press, 2000.

[11] Kalleberg K. T., Visser E.: Fusing a Transformation Language with an Open Compiler. Report TUD-SERG, ISSN 1872-5392, 2007.

[12] Cordy J. R.: TXL - a language for programming language tools and applications. ENTCS, pages 3-31, 2004.

[13] Chlipala A.: A Certifed Type-Preserving Compiler from Lambda Calculus to Assembly Language. University of California, Berkeley, 2007.

[14] Yao, J. T., Zhang, M.: A Fast Tree Pattern Matching Algorithm for XML Query. In Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI '04). IEEE Computer Society, Washington, DC, USA, pages 235-241, 2004.

[15] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L.: Clone Detection Using Abstract Syntax Trees. In Proceedings of the International Conference on Software Maintenance (ICSM '98). IEEE Computer Society, Washington, DC, USA, pages 368-377, 1998.

[16] Kbler, J., Torn, J.: The Complexity of Graph Isomorphism for Colored Graphs with Color Classes of Size 2 and 3. In Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS '02), Helmut Alt and Afonso Ferreira (Eds.). Springer-Verlag, London, UK, UK, pages 121-132, 2002.

[17] Filotti, I. S., Mayer, J. N.: A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In Proceedings of the twelfth annual ACM symposium on Theory of computing (STOC '80). ACM, New York, NY, USA, pages 236-243, 1980.

[18] Bunke, H., Irniger, Ch., Neuhaus, M.: Graph matching - challenges and potential solutions. In Proceedings of the 13th international conference on Image Analysis and Processing (ICIAP'05), Fabio Roli and Sergio Vitulano (Eds.). Springer-Verlag, Berlin, Heidelberg, pages 1-10, 2005.

[19] Bublík, T., Virius, M.: New language for searching Java code snippets. In ITAT 2012. Proc. of the 12^{th} national conference ITAT. Ždiar, Pavol Jozef Šafárik University in Košice, pages p. 35-40, 2012.

Parallel Programming with NVIDIA CUDA

Vladimir Spanihel[1]

Czech Technical University, Prague 110 00, CZE,
vladimir.spanihel@seznam.cz

Abstract. This paper sumarizes elementary knowledge about NVIDIA CUDA programming. It brings description of basic terms nearly related to GPU programming and an introduction to architecture of NVIDIA GPUs.

1 Introduction

Nowadays, many people are concerning with HPC (High Performance Computing). Their common goal is to get the highest possible throughput of applications solving their issues. In most cases, parallel programming is used to achieve good results within a small amount of time. Since todays computers are powered by multi-core CPUs, the CPU parallelization of tasks is widely used. However, solutions of a relatively big amount of these tasks are composed of a sequence of simple operations applied on a huge input data set. Thanks to developers from NVIDIA, there is an alternative to the CPU computing, which can dramatically increase performance of the computations. Before a few years, their effort resulted in the CUDA architecture. It allows concurrently perform simple computations on many-core GPUs. This paper is aimed at an introduction to CUDA programming. It describes HW architecture of Fermi cards, GPU occupancy calculation, application interfaces and useful tools and libraries used for GPU programming.

1.1 Basic Terminology

It is importat be familiar with basic terminology of CUDA programming. Kernel, thread and warp are the most frequently used terms. The term kernel denotes a method to be invoked on GPU. The CUDA thread is the name for one data element processed by a kernel. Due to hardware restrictions, threads can be processed in groups of 32 threads called warps.

The next common words are host and device which are used for distinguishing between CPU (host) and GPU (device) parts of code, memories etc.. The rest of terms will be explained further.

1.2 CUDA Compute Capability and SDK

NVIDIA uses so-called compute capability to differentiate GPU architectures with various capabilities. Compute capability is a number comosed of major

and minor part e.g. 2.1. Here major is equal to 2 and minor equals to 1. The major part defines hardware architecture of NVIDIA GPUs. All till now released architectures named Tesla, Fermi and Kepler have major numbers 1, 2 and 3 respectively. As NVIDIA has already published, Maxwell and Volta are names for next two generations of CUDA capable cards. Besides NVIDIA graphics card, CUDA Toolkit is necessary for writing applications computing on GPU. It consists of various tools such CUDA compiler nvcc, debuger cuda-gdb and useful IDE Nsight. CUDA Toolkit 5.0 is the latest stable release, however, version 5.5 RC is now prepared.

2 Chip architectures

Since GPUs are designed to simultanously process graphics data and display it on screen, architecture of GPUs is massively parallel. The field of GPU programming was established to take advantage the parallel architecture of graphics cards.

The architecture of all GPUs is based on cooperation of many simple cores. These cores are suitable to peroform one instruction on huge data set in parallel. Because of the fact, GPUs could be marked as SIMD (Single Instruction Multiple Data) in the Flynns taxonomy (see Table 1). Some people classify this architecture as SIMT (Single Instruction Multiple Threads).

Table 1. Flynn's taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

While GPU processors are very simple compared to CPUs, their production is much cheaper. So that, a GPU card containing hundreds of CUDA cores costs a fraction of the price of CPU with tens of cores. GPUs was designed to quickly process many graphics data.

The first version of NVIDIA CUDA was announced in 2006. (see [1]). NVIDIA made many improvements resulting in very useful Fermi architecture.

2.1 Fermi

The Fermi chip can be composed of one or more streaming multiprocessors. One streaming multiprocessor (SMP) contains computing units called streaming processors (SP) and special function units (SFU). Although both SP and SFU computes in single precision, their roles are separated. Each SP does simple mathematics operation, whereas, SFUs performs calculations of more complex operations e.g. trigonometric functions and multiplication. Double precision on Fermi and later architectures is achieved by coopereation of two cores (SPs).

This is the reason why calculations in signle precision are about a half faster than computing in double precision on Fermi GPU.

Nowadays, newest GPUs originate from Kepler series. The architecture of such cards is not very different from Fermi. More about CUDA achitectures can be read in [2].

2.2 CUDA Execution model

An execution of each kernel starts certain number of threads. These threads are organized into so-called blocks and blocks into a grid. The thread hierarchy is depicted in Figure 1. Grid and blocks can be up to 3-dimensional. Its particular dimension size has to be specified before invoking the kernel in code. For this purpose, tripple angle brakets with execution specification `<<<blockDim, gridDim>>>` must be used while calling the kernel. Variables blockDim and gridDim can be of type int for one dimensional blocks and grid or of type dim3 to specify higher dimensionality of blocks and grid. Here is an example of kernel invocation:

```
kernelName<<<10, 128>>>(param1, param2);
```

Related to the compute capability of GPU, there are one, two or four warp schedulers, reponsible for distribution of warps on streaming multiprocessors, on GPUs. Warps are processed on SMs independently. Whole warp executes same instruction. Therefore, the best throughput is achieved when all threads can work. When threads of a warp start diverging, according to data-dependent conditionals, processing of each branch is serialized. This phenomenon is called warp divergence.

3 Available Memory Types

The working CUDA code can be simply written using just global device memory which is readable and writable from each thread. However, CUDA developers can take advantage of several memory types, available on NVIDIA GPUs, to increase their application performance.

3.1 Global Memory

The most common attribute of graphics cards, presented by IT shops, is the size of global memory. Since GPU can not directly access host memory, all required data has to be copied in GPU memory. Such data transfers can be very time consuming parts of code, so that, their count should be minimized. Situation, when whole solved issue is not greater than the global memory of card used to solve the problem, is perfect.

Even though, an access rate to the global memory reaches up to 28 GB/s, this memory type is the slowes one relative to the other types. So utilizing of the other memory types may significantly speed-up the application.

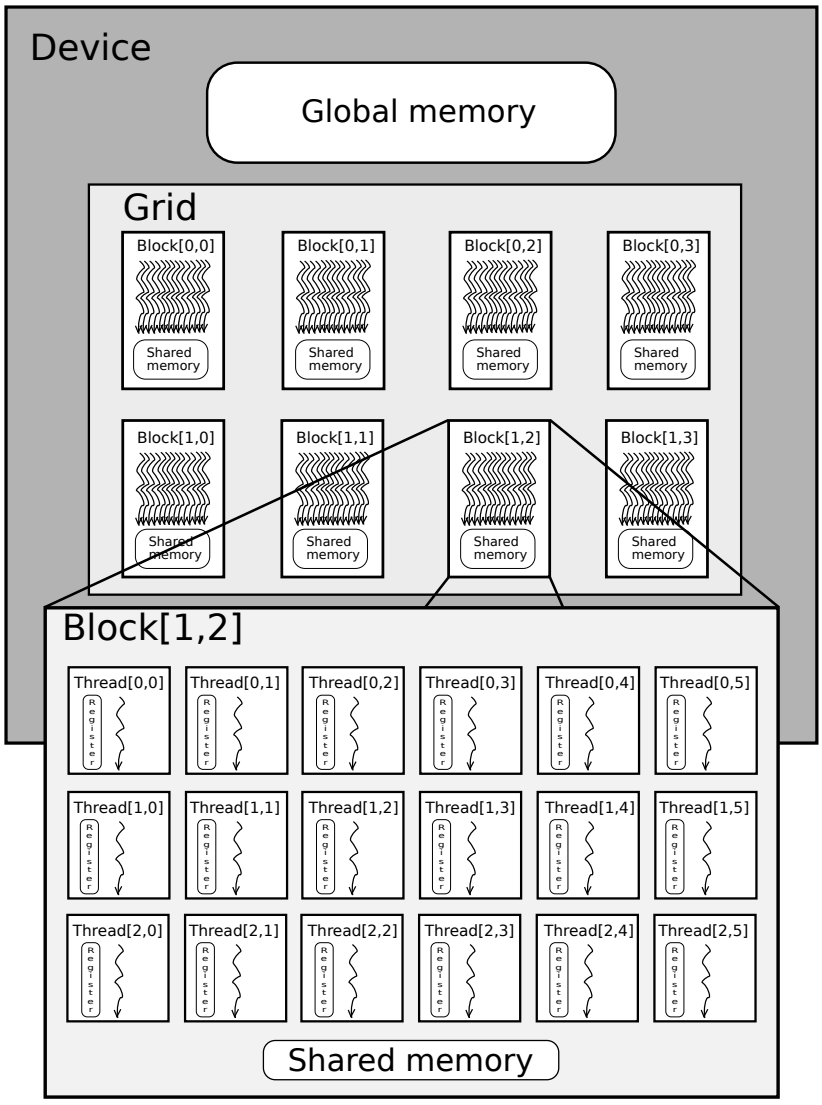


Fig. 1. Threads hierarchy example

3.2 Private Memory

Besides thread hierarchy, Figure 1 illustrates GPU memory hierarchy. As can be seen from the figure, each thread has its own private memory called register. It contains e.g. build-in variables important for unique identification of each thread. Data stored within this memory can be read just from the corresponding thread. Examples of such variables can be found in Table 2.

Table 2. Examples of build in variables stored in private memory of each thread

`threadIdx.x`	index of the thread within the coordinate x of the block
`threadIdx.y`	index of the thread within the coordinate y of the block
`blockIdx.x`	index of the block within the coordinate x of the grid
`blockIdx.y`	index of the block within the coordinate y of the grid
`blockDim.x`	dimension of the blocks coordinate x
`gridDim.y`	dimension of the grids coordinate y

3.3 Other Memory types

Shared memory is available from all threads located within one block and provides both read and write access.

Constant and texture memories are parts of the global memory. Both are accessible from each thread.

3.4 Variable Declaration

CUDA provides qualifiers to specify memory where a variable will be created. There are available these qualifiers `__device__`, `__shared__`, `__constant__` in the CUDA API. Declaration of variable `i` in the shared memory looks like:

```
__shared__ int i;
```

4 NVCC Compiler

CUDA C language brings some extensions to the C language. Tripple angle brackets or variable type qualifiers mentioned above are not allowed by C standard [3]. Due to this fact, standard C compilers can not be used for building CUDA source code. So NVIDIA wrapped standard C compiler to accept CUDA extensions. The new compiler has name NVCC. Running NVCC compiler accepts the same parameters as C/C++ compiler.

Ussually, source code containing device code is stored in file with extension .cu.

5 Useful Tools and Libraries

As time passed, various libraries and tools was added into CUDA SDK. The installation includes the NVCC compiler [4], Nsight (IDE) [5] and profiler tool [6].

Probably the most famous libraries from the package are CUBLAS [7] and CUSPARSE [8]. The CUBLAS library solves basic tasks from linear algebra in parallel on GPU. The CUSPARE library performs parallel algorithms on sparse matrices.

6 Acknowledgement

This work is supported from grant SGS11/167/OHK4/3T/14 and GA TA CR TA01010490.

References

1. NVIDIA: What is CUDA. [ONLINE] [cit. 2013-07-20]
   URL: `https://developer.nvidia.com/what-cuda`
2. NVIDIA: CUDA C Programming Guide. [ONLINE] [cit. 2013-07-20]
   URL: `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`
3. The Standard - C. [ONLINE] [cit. 2013-07-20]
   URL: `http://www.iso-9899.info/wiki/The_Standard`
4. NVIDIA: CUDA COMPILER v5.0. October 2012
5. NVIDIA: NSIGHT ECLIPSE DG-06450-001_v5.0. October 2012
6. NVIDA: PROFILER DU-05982-001_v5.0. October 2012
7. NVIDIA: CUBLAS LIBRARY v5.0. October 2012
8. NVIDIA: CUSPARSE LIBRARY v5.0. October 2012

Economic Time Series as Objects and Principal Component Analysis

Radek Hřebík

Czech Technical University Faculty of Nuclear Sciences and Physical Engineering,
Department of software engineering, Břehová 7,
115 19 Prague, Czech Republic
Radek.Hrebik@seznam.cz

Abstract. Paper deals with principal component analysis in sphere of economic data. The aim is not to deal primary with principal component analysis but to show the possible way of use in interpreting economic indicators. As it is well known principal component analysis reduce the dimensionality of origin data set, the input for this research is very simple, statistic data describing the economic situation of more than thirty states during twenty two years. Paper presents three basic ways of interpreting these data as input to principal component analysis and shows the results.

1 Introduction

The contribution is focused on principal component analysis (PCA). The aim is not to describe the principal component analysis itself in detail. The main idea of principal component analysis is reduction of dimensionality of some data set that consists of a large number of interrelated variables. The reduction retains as much as possible of the variation present in the data set. The aim is achieved by transforming to a new set of variables called the principal components. These principal components are uncorrelated and ordered so that the first few retain most of the variation present in all of the original variables. [2] In this research is the aim the reduction to two principal components (PC1 and PC2).

Paper deals with the basic economic data and shows the ways of possible interpretation to serve as input for principal component analysis. The aim is to search the main indicators, monitor the potential trend of concrete objects and finding objects having something in common. It goes hand in hand with principal component analysis goal defined by Abdi and Williams – extracting the important information from the table to represent it as a set of new orthogonal variables called principal components and to display the pattern of similarity of the observations and of the variables as points in maps. [1]

Paper presents three basic ways of using principal component analysis to interpret economic data. The way means to interpret the data set as objects. As the reason for doing such research can be also trying to predict the future development of some country and find the position of state if we know the basic

economic prediction. There is also very interesting to capture some progress in time.

The study dealing with principal component analysis to forecast a single time series with many predictors was presented by Stock and Watson. [4] The use of principal component analysis in connection with gross domestic product is discussed for example in [5]. Favero deals with comparison of two competing methods to estimate large-scale dynamic factor models based, respectively, on static and dynamic principal components. [6] Principal component analysis as alternative way to predict gross domestic product is presented in [7].

1.1 Used data

To do such research play the key role the input data set. As already said it should be some economic time series. Used economic data has been selected from Statistical Annex of European Economy presented by European Commission in spring 2013. [3]

As input to analysis serve the thirty five countries from the whole world, majority are the European countries. The observation take place in years 1993 to 2014. Selected indicators are the total population, unemployment rate, gross domestic product at current market prices, private final consumption expenditure at current prices, gross fixed capital formation at current prices, domestic demand including stocks, exports of goods and services, imports of goods and services and gross national saving. So totally nine indicators are monitored. As the time series go to year 2014 it is clear that years 2013 and 2014 represent predictions.

2 State in year as object

As first possible interpretation of the data set is the object represented by a state in a given year. So the number of objects is relatively high. The total number of object is in this case seven hundred and eighty, it represents number of states multiplied by the number of observed years.

As the number of object is high, the origin data set dimensionality is relatively small. It is created just by nine indicators. The result of principal component analysis is that two principal components are created mainly by combination of population and gross domestic product as shown the indicators weights in table 1.

The figure 1 shows that main points are concentrated by vertical axis. As representative state of vertical line can be selected for example Germany. As the state represented by movement also in horizontal line can be mentioned for example France. Because the number of objects is quite high, for better interpretation there are the objects grouped by the same colour for a given year in figure 2. The weights of components are in table 1. The first principal component explains almost all of the variance.

Table 1. PCA – State in Year as Object

Indicator	PC1	PC2
Total population	-1,106	1,683
Unemployment rate	-0,000	0,025
Gross domestic product	-0,113	-16,492
Private final consumption expenditure	-0,000	0,013
Gross fixed capital formation	0,000	-0,021
Domestic demand including stocks	-0,000	0,004
Exports of goods and services	-0,000	-0,062
Imports of goods and services	-0,000	-0,060
Gross national saving	0,000	-0,026

Fig. 1. PCA – State in Year as Object

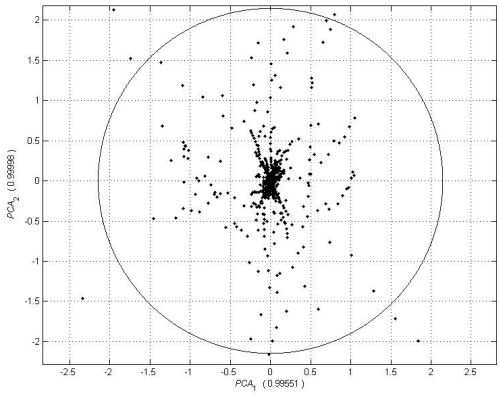
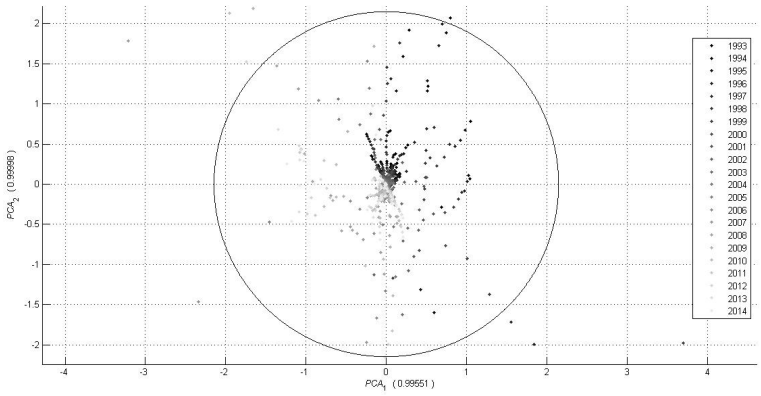
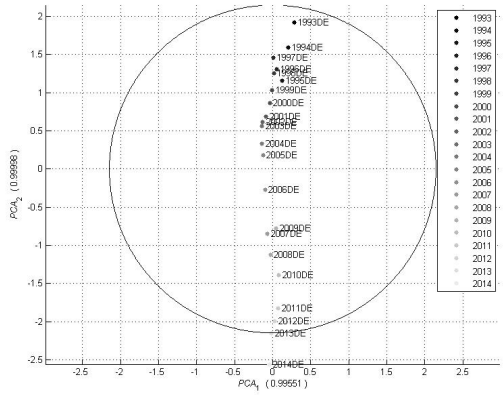


Fig. 2. PCA – State in Year as Object with legend



The detailed view on values of principal components for three selected countries is shown in table 2. As already mentioned Germany is represented by points in vertical line as can be seen in figure 3.

Fig. 3. PCA – State in Year as Object – Germany



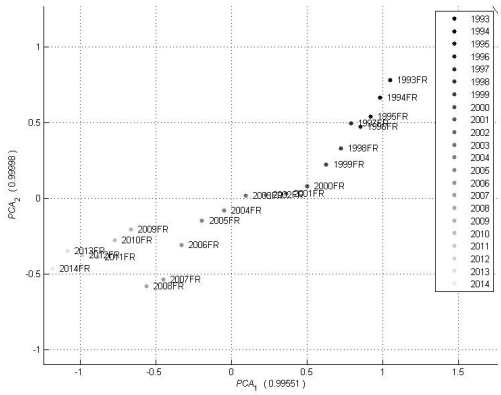
In case of France there is the result of principal component analysis shown in figure 4 from which is evident that growing gross domestic product is connected with growing population. So in this case the growing gross domestic product goes hand in hand with growing population. That is the different between France and

Table 2. Values of principal components of selected countries

Year	CZ - PC1	CZ - PC2	DE - PC1	DE - PC2	FR - PC1	FR - PC2
1993	0,0067	0,2861	0,2827	1,9173	1,0522	0,7847
1994	0,0050	0,2647	0,2052	1,5870	0,9852	0,6696
1995	0,0063	0,2370	0,1282	1,1589	0,9195	0,5430
1996	0,0110	0,1980	0,0564	1,3097	0,8555	0,4740
1997	0,0144	0,1865	0,0091	1,4556	0,7919	0,4969
1998	0,0170	0,1607	0,0148	1,2527	0,7238	0,3324
1999	0,0206	0,1501	-0,0047	1,0328	0,6289	0,2273
2000	0,0235	0,1200	-0,0371	0,8603	0,5009	0,0847
2001	0,0383	0,0600	-0,0854	0,6842	0,3648	0,0330
2002	0,0450	-0,0031	-0,1299	0,6133	0,2278	0,0244
2003	0,0446	-0,0073	-0,1421	0,5579	0,0936	0,0207
2004	0,0429	-0,0389	-0,1377	0,3294	-0,0474	-0,0789
2005	0,0342	-0,0848	-0,1273	0,1840	-0,1925	-0,1445
2006	0,0236	-0,1322	-0,1000	-0,2735	-0,3284	-0,3074
2007	0,0060	-0,1683	-0,0720	-0,8476	-0,4502	-0,5334
2008	-0,0275	-0,2207	-0,0296	-1,1202	-0,5596	-0,5795
2009	-0,0458	-0,1366	0,0486	-0,7816	-0,6633	-0,2043
2010	-0,0540	-0,1618	0,0810	-1,3902	-0,7716	-0,2752
2011	-0,0481	-0,1983	0,0712	-1,8237	-0,8814	-0,3854
2012	-0,0523	-0,1764	0,0276	-1,9930	-0,9842	-0,3722
2013	-0,0549	-0,1612	-0,0166	-2,1568	-1,0835	-0,3459
2014	-0,0566	-0,1736	-0,0425	-2,5564	-1,1822	-0,4637

Germany, where the gross domestic product is growing in conditions of almost the same population.

Fig. 4. PCA – State in Year as Object – France



The example of Czech Republic shows that the population is almost constant as in case of Germany, but the potential to grow the gross domestic product is much smaller. The differences between years are very small.

3 States as objects

In second case of possible use of principal component analysis there are the object represented by each state. So the properties are made of indicators in selected years. The number of object is thirty five.

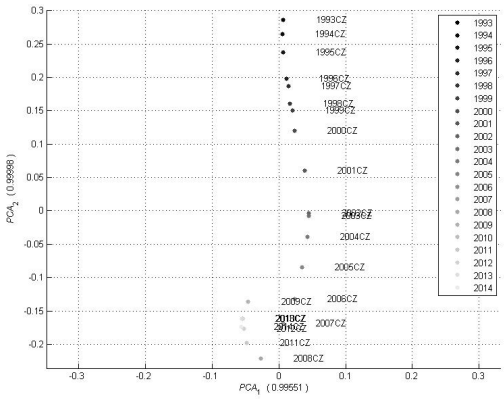
In comparison to first case of use the number of objects is dramatically fallen down. So the representation will be very simple and it will be clear which states are closed to each other. From graphic representation are easily noticed the groups of states. When one point represent one state there is very easily seen the groups of states with similar type of economy. The result of principal component analysis is shown in figure 6. The first principal component explains almost ninety nine percent of variance in origin data set. The values of principal components of each state are summarized in table 3.

The principal components are in this case counted from nearly two hundred indicators. So the reduction of dimensionality is high in this case. These values are created by the nine economy indicators in twenty two years. As in the first case of using principal component analysis also here are the biggest weights on gross domestic product and population. In case of first principal component is the population values included with bigger weight than in case of gross domestic

Table 3. PCA – States as objects

State	PCA 1	PCA 2
Belgium	0,195	0,376
Germany	0,258	-0,115
Estonia	0,317	0,090
Ireland	0,179	0,044
Greece	0,223	-0,118
Spain	-0,547	-0,509
France	-0,443	0,558
Italy	-0,202	1,178
Cyprus	0,279	0,075
Luxembourg	0,290	0,061
Malta	0,299	0,020
Netherlands	0,148	0,089
Austria	0,239	0,141
Portugal	0,230	-0,205
Slovenia	0,296	0,062
Slovakia	0,296	0,047
Finland	0,267	0,129
Bulgaria	0,431	-0,001
Czech Republic	0,280	0,240
Denmark	0,262	0,090
Latvia	0,356	0,022
Lithuania	0,377	-0,032
Hungary	0,347	0,067
Poland	0,279	0,388
Romania	0,468	0,333
Sweden	0,219	0,361
United Kingdom	-0,318	1,921
France	0,324	0,118
F.Y.R. of Macedonia	0,294	0,016
Iceland	0,297	0,022
Turkey	-1,189	-4,953
Montenegro	0,304	0,021
Serbia	0,322	-0,169
United States	-5,431	1,033
Japan	0,056	-1,400

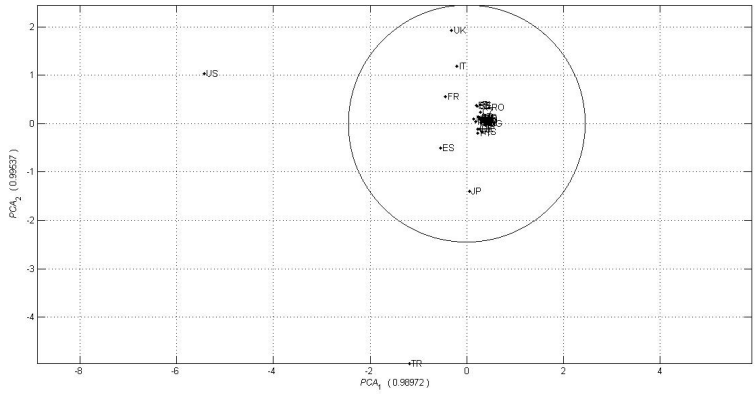
Fig. 5. PCA – State in Year as Object – Czech Republic



product. Second principal component is preferring the values of gross domestic product in years.

The values of first principal component are in most cases very close to zero, following the weights that implies that the population is without big changes having affect to component values. Second principal component is mostly counted from gross domestic product values. There also apparent the bigger range in values.

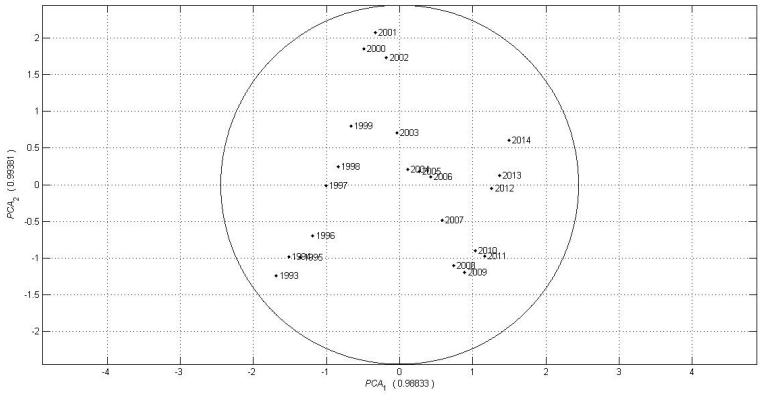
Fig. 6. PCA – States as objects



4 Years as objects

The third kind of data interpretation is by objects representing calendar year. So there is only twenty four objects in this case. As the number of objects is decreasing, the number of properties of each object is increasing. The total number of indicators of each object is created by number of countries mal number of describing properties. The number of properties is totally over three hundreds. The result showing principal component values is shown in figure 7. The first principal component explains almost ninety nine percent of variance in origin data set.

Fig. 7. PCA – Years as objects

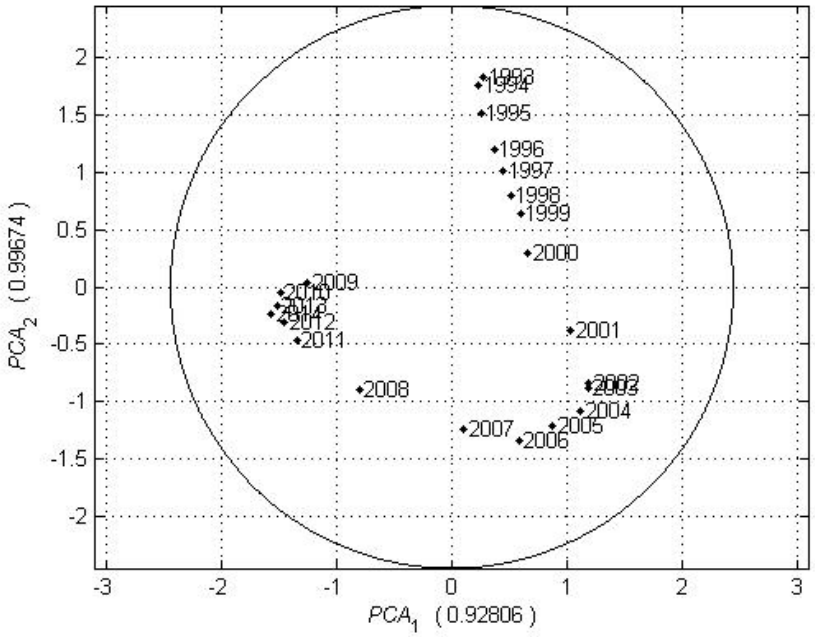


The advantages of such approach is the very clearly seen the progress in time. The next possible use of this approach is to do the analysis just for national data and see the development of separate country. Example of Czech Republic is shown in figure 8. In this case explains the first principal component almost ninety three percent of variance in origin data set. Both principal components explain almost all variance in origin data set.

5 Summary

It was shown that principal component analysis can be also very useful in interpreting the economic data. It represents some other way of interpreting time series and shows how the states position in comparison to others. To fully interpret the results there is need to study the weights of principal components to know what stands behind the components values. The third case of use – the years as objects – gives very clear representation of changing economic situation.

Fig. 8. PCA – Years as objects – Czech Republic



References

1. H. Abdi and L. J. Williams. *Principal Component Analysis.* (2010). [online]. [cited 2012-08-21]. `http://www.utdallas.edu/~herve/abdi-awPCA2010.pdf`.
2. I. T. Jolliffe. *Principal Component Analysis – 2nd Ed.*. Springer (2002).
3. European Comission. *Statistical Annex of European Economy: Spring 2013.* Economic and Financial Affairs (2013). [online]. [cited 2012-08-21]. `http://ec.europa.eu/economy_finance/publications/european_economy/2013/pdf/2013_05_03_stat_annex_en.pdf`.
4. J. H. Stock and M. W. Watson. *Forecasting using principal components from a large number of predictors* Journal of the American Statistical Association, Vol. 97, No. 460, p. 1167-1179. (2002).
5. C. Schumacher. *Forecasting German GDP using alternative factor models based on large datasets.* Bundesbank Discussion Paper 24/2005. (2005).
6. C. Favero, M. Marcellino, F. Neglia. *Principal components at work: The empirical analysis of monetary policy with large datasets.* Journal of Applied Econometrics, 20, p. 603–620. (2005).
7. G. Chamberlin. *Forecasting GDP using external data sources.* Economic and Labour Market Review vol. 1(8), August 2007, p. 18-23. (2007).

Model-Driven Development of a Banking Multichannel Solution

Petr C. Smolik

Metada s.r.o., Prague, Czech Republic
petr.smolik@metada.com

Abstract. Development of large information systems requires large amount of knowledge gathering. In order to implement the needed functionalities, many things need to be understood and figured out. Conventionally, this knowledge gets encoded deeply into the program code and when technologies are replaced the knowledge goes away with them. We present experience from a project in banking where large multichannel solution was built based on model-driven engineering techniques. Metamodel of the solution was created and all business-relevant aspects of the solution were modeled in a modeling environment supporting parallel development and versioning. In the runtime, models are being executed by a model interpreter. It is now possible to inspect the system functionalities on a higher level of abstraction and the organization has the added flexibility to replace the runtime technologies as needed without losing the expensively gathered knowledge.

Keywords: model-driven engineering, model interpretation, multichannel solution, software product lines, FSPL

1 Introduction

Banking is a software-intensive industry where large information systems need to be used to support various financial products and services. Historically, because of easier management, banks operated with separate vertical divisions for different types of products (e.g. accounts, mortgages, loans, cards). Each division built its own support systems and often even held its own client data associated to its own product information. This way banks ended-up with several core backend systems and dispersed customer data. Nevertheless, with advances of information technologies, especially web and mobile technologies, financial services are forced to become more customer-centric. Banks need to have one place with customer data, so called "single view of customer", and need to serve customers uniformly over all different types of products and communication channels. For this reason banks are building "multichannel" solutions where one system integrates all interactions of a customer with the bank. Currently the channels may include internet banking, mobile banking, telephone banking, ATMs, social media, or branch banking.

adfa, p. 1, 2011.

Multichannel solutions as information systems cover three important areas: frontends, business logic, and integration. Frontend is the presentation layer via which users directly interact with the system. Integration ensures that the multichannel solution communicates with all the numerous existing backend systems within the bank. The business logic within multichannel solutions takes care of all processing that is not implemented within existing backend systems.

A multichannel solution development project may take several years and may include over a hundred of people involved. Most of the work is about trying to understand all the details of different products and processes. The knowhow is then encoded into a form executable by computers. Conventionally developers write code in a specific general purpose programming language (like Java) as part of some application development framework. A multichannel solution may be composed of few relatively simple concepts, mainly operations and forms. The problem is the scale, because relatively simple solution may include several hundred up to few thousand forms and operations. Another problem is that most of the solution is user-facing. The advances of user interface technologies in the past decade were immense. Banks need to keep up with these advances and have to be able to update technologies as new ones become mainstream.

We have seen cases where banks had to throw away their old multichannel solution and create a new one just to keep up with technologies. The worst thing was that they threw away not only the old technology but also their expensive knowhow that was encoded into it. This is exactly where Model-Driven Engineering (MDE) can come to the rescue. The only thing that needs to be done is to encode the knowhow in an executable model that is not implementation dependent, and enable code generation or model interpretation within the multichannel solution.

The following sections describe how a particular multichannel solution was created using model-driven techniques. The section on Domain Engineering is focused on the architecture, metamodel, framework, and tooling used as foundation used for building the solution. The section on Application Engineering discusses how the solution was modeled, how channel applications were created. Some details about the development process are discussed in the Discussion and Parallel Development sections. The terms domain and application engineering were borrowed from the area of Software Product Lines (SPLs) [1], because they nicely express the two main streams of activities when designing a model-driven solution. There seems to be a natural relation between software product line engineering and model-driven engineering, which is also supported by [2].

2 Domain Engineering

The domain of multichannel solutions is concerned with client interactions over different communication channels. It includes various presentations of relatively similar functionalities, some common business logic, and a lot of integration to many existing systems. Using model-driven engineering, all this essential information needs to be expressed in a model.

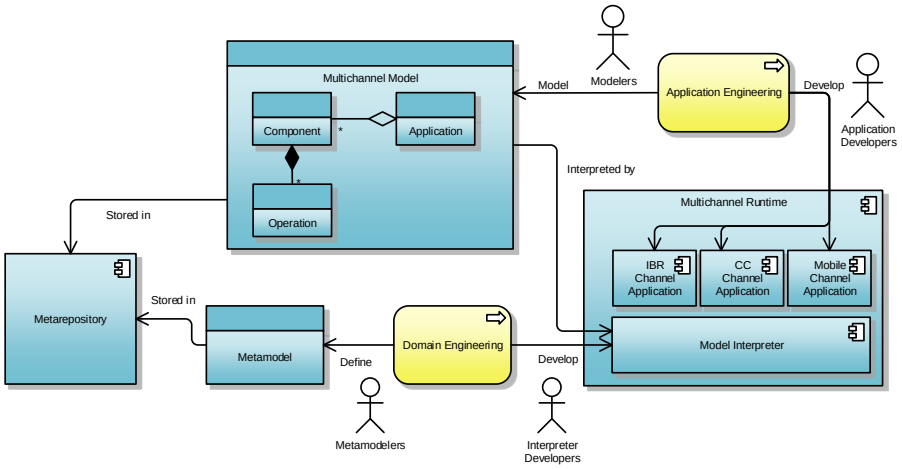


Fig. 1. Overview of components, objects, processes, actors, and their relations

Fig. 1 shows the overall architecture of the model-driven approach used. Models and metamodels are stored in the Metarepository, a metamodeling tool that allows web-based access to both modelers and metamodelers. The Multichannel Runtime is composed of various channel applications and models are fully interpreted by the Model Interpreter. The following subsections describe these components and objects, and explain their relation to development processes and actors.

2.1 Metamodel

In the beginning we had an ambition to create a "Financial Services Product Line" (FSPL) for fast configuration of banking products. Nevertheless, that would require replacing also the product backends, which was not in the scope of the multichannel solution. Therefore we had to settle down with modeling not the banking domain itself, but just the multichannel solution domain. This section will briefly describe this multichannel solutions domain. Generally there are channel applications composed of various frontend and business components. Each component offers some operations as shown on **Fig. 2**.

Fig. 2. Channel applications composed of components and operations

Channel applications thus may be assembled from several reusable components. Operations are the only thing exposed by components and may have several types as shown on **Fig. 3**.

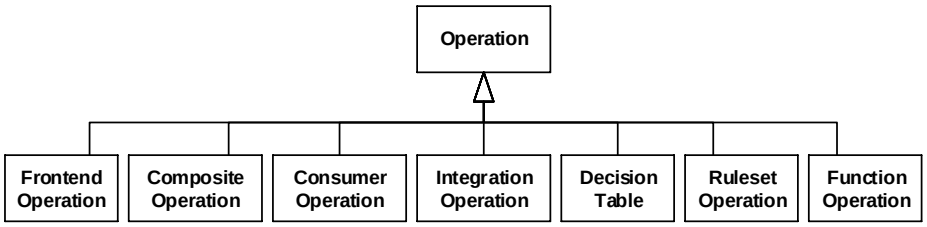


Fig. 3. Types of operations

Frontend, composite, and consumer operations have flows of actions, where call actions may call other operations. Integration operations ensure integration to external systems via various adapters. Decision tables and ruleset operations enable business decisions to be described and integrated into decision flows. Function operations enable computational expressions to be included.

From the user interface perspective, the most interesting operation is the frontend operation shown in **Fig. 4**. Decision actions allow for flow branching. Call actions allow other operations to be called and their results may be used by other actions in the flow. Data are always mapped from context of the flow to the action called. The context itself is a set of all action results.

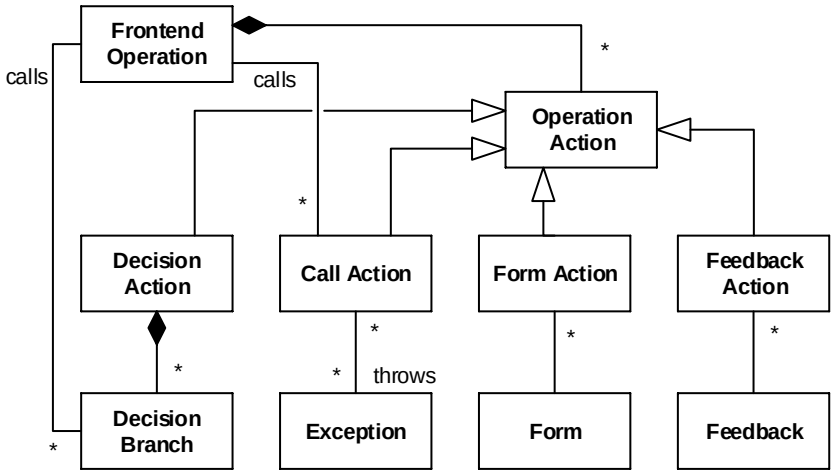


Fig. 4. Frontend operation and its flow actions

Form actions represent user interactions where the user is presented with information and decides about further processing. Feedback actions are used to provide users with feedback on the actions performed, mainly signaling exceptional states.

Composite operations are similar to frontend operations, but they do not include user interactions (forms). They are used mainly for defining business logic reusable across channels. Consumer operations are operations that may be called by external systems to provide or retrieve information from the multichannel solution.

There are other concepts used such as application menus, operation mocks, more types of operation actions, mappings, interfaces, form widgets, exception handling, tests, etc. that are also part of the multichannel metamodel, but their details are not necessary for overall understanding of the presented model-driven solution.

2.2 Model Interpreter

Based on our experience with both code generation and model interpretation (presented in [3]) we decided to use full model interpretation. For a large modeling project, the biggest advantage of model interpretation is that the modelers are able to see quickly the running application with the new functionality they are currently working on. The usual cycle is to create form, put it into a flow, map some data to it and see how it looks and how it works. The path from model to running application has to be as fast as possible. In such a case immediate model interpretation via purposely implemented Java-based framework was superior over Java code generation.

The model interpreter is able to execute the various types of operations based on the models represented in the form of XML configuration files. It renders screens with forms, maps data to flow actions, calls integrated systems, makes decisions, evaluates expressions, etc.

Within model-driven solution, the value of model interpreter is in its reusability for different models confirming to the same metamodel. The models contain the valuable information expressed in technology-independent way and the model interpreter is the particular technology that is able to execute it. Furthermore, the model interpreter itself is replaceable by another model interpreter written different way in another technology, or model interpretation may be replaced by code generation into yet another technology.

2.3 Metarepository

During the domain engineering process metamodelers define the metamodel and interpreter developers implement the model interpreter that enables execution of models that confirm to the metamodel. To enable multichannel applications to be created by modelers modeling tools are needed. The best model-driven way to create a modeling tool is to configure a metatool with a metamodel. For this purpose we used the Metada Metarepository, which is a metamodeling tool based on the Mambo Metamodeling Environment [4] and is similar to Adex [5]. Metarepository was used to define the metamodel that in turn is interpreted by the Metarepository to create the multichannel solution modeling environment.

3 Application Engineering

It took several iterations of the domain engineering process to create a modeling environment usable by application modelers to model the multichannel solution. Even as application was being modeled the metamodel, model interpreter, and even the metatool had to evolve based on the gained knowledge from the project. For example, the metamodel and interpreter had to accommodate support for various new widgets or exception handling mechanisms. Metarepository had to be extended to support private development branches, so that each modeler would have a sandbox and would not be continually distracted by ongoing changes of other developers.

3.1 Multichannel Solution Model

Five different channel applications were modeled. After several adjustments, components were divided into two types: frontend components and business components. Only frontend components include frontend operations and there is only one frontend component per channel as shown in **Fig. 5**. Business components (about twenty) are channel-independent reusable functionalities divided by business subdomain such as payments, deposits, investments, or loans. Several business components also cover other aspects such as statements, notifications or security.

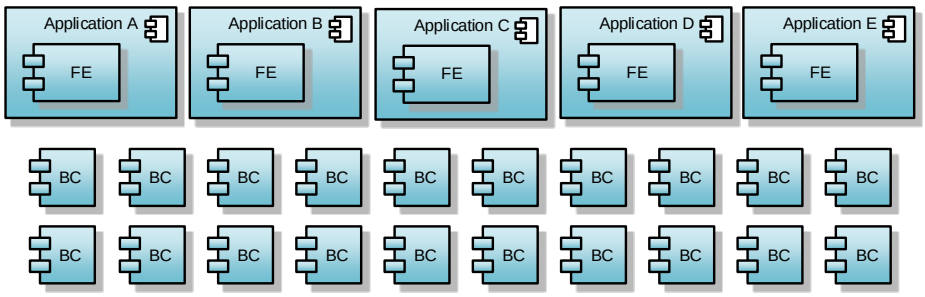


Fig. 5. Applications, frontend components, business components

The business components also include integration and consumer operations representing all communications with external systems for the given business domain.

3.2 Channel Applications

There were five different channel applications each with its specifics that were not necessary to express in a model. The look and feel of a mobile application is different than that of the web application. Client facing applications have different non-functional requirements than applications for call-center employees. Each channel application may require also different ways of user authentication. For these reasons

the base framework for each channel application was hand-coded and included points through which the modeled functionalities (mainly menu and frontend operations) were exposed. There was a special team of application developers that took care of this part of development.

4 Discussion

The project was originally planned as a two year project that should implement a new multichannel solution to fully replace the old existing one. The expected effort was high and over a hundred of people had to participate on the project in different roles at most times. There were in total 160 people that used the Metarepository tooling during the course of the project and there were peak times when there were about 50 concurrent users. If there are specific problems to be named they are:

- Scale of functionalities to be implemented in relatively short time
- High parallelism of development tasks
- Relative shortage of project management skills

Model-driven techniques are relatively good in managing complexity by increasing the level of abstraction. This way variety of the system being implemented is lower and thus it is easier to control its implementation (based on the law of requisite variety [6]). Nevertheless, the scale of the requirements to be realized was so big that there was no one on the project that could alone take hold of. By the principle of performance load [7] that states that "the greater the effort to accomplish a task, the less likely the task will be accomplished successfully", the project was lucky to actually successfully complete its main goals within three years. There is no way to know how the project would do without the model-driven approach, but we believe that it would not be as successful.

5 Parallel Development and Versioning

The high parallelism of tasks was something we could help with on the modeling tools side. Before the project, Metarepository used Subversion versioning system to version models. This made release management possible and it was a good model distribution and backup solution. Nevertheless, what was not easy with Subversion was sandboxing, because we had just one central instance of the Metarepository accessed by all modelers via an internet browser, and did not want to decentralize the tooling. We only needed sandboxing that would allow individual modelers to be shielded from changes of other modelers in the related parts of models.

Since the models were executable, the changes of one modeler were necessarily breaking executability for others. We therefore looked for a solution that would effectively enable large number of concurrent development branches that could be easily updated with changes of other users only when needed, and allow changes to be published to other users only when they are done and tested. Also we needed a solution

that would enable changes from one release branch to be merged into another release branch (e.g. a production fix from a production branch to a development branch).

The solution was to use Git versioning system and extend it with the support for sparse checkouts. When a private branch (i.e. sandbox) is created, only a new Git branch pointer is created, when changes are made, only those changes are written to the sparse. No full checkouts are needed, which makes large number of private branches to be continually in use (in our case around 150 at times).

The reason for that many branches was not only the number of modelers, but also parallelism of the project releases. When the first release was migrated to production, a release branch had to remain for production fixes, there was also a development release branch for next minor release, and a third release branch where the next major release was being prepared. Thus there were three parallel release branches with its associated private branches, one or more per individual modeler.

6 Conclusion

We have applied model-driven techniques on a relatively large project in banking, where a new multichannel banking solution was created within three years' time to replace an existing obsolete solution. We used Metarepository, our own metamodeling tool, to create metamodel of the multichannel solution, and to enable modelers to fully model all business-relevant functionalities of the solution in an executable model. For the purposes of model execution a model interpreter was built to provide for all the needed user interactions, business logic, and integrations.

The metamodel was relatively simple, but we had to deal with the large amount of functionalities that had to be implemented in relatively short time by numerous modelers in a parallel and coordinated fashion. To support the parallelism and multi-user aspect of modeling, we had to extend our tooling with the ability to support multiple parallel model branches so that individual modelers would be able to develop and debug their executable models in a sandbox without distracting and without being distracted by all the other ongoing development of other modelers.

The major project milestones were reached, though with some delays that could have been accounted for with such a complexity and effort expected to be realized. The metamodel and the tool helped abstracting the system into smaller number of clear concepts that could be then more effectively defined and manipulated. It was also possible to develop large amount of functionality in manageable and coordinated fashion in a working environment with many people and some lack of management skills.

From the model-driven engineering perspective we see that the biggest value for the given organization is that the three years' worth of work of so many people is cleanly expressed in a platform independent model and not encoded into a specific technology again, and will not have to be thrown away when the technology becomes obsolete the next time.

References

1. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin, Heidelberg, New York (2005)
2. Arboleda, H., Royer, J.-C.: Model-Driven and Software Product Line Engineering. ISTE Ltd and John Wiley & Sons (2012)
3. Smolik, P., Vitkovsky, P.: Code Generation Nirvana. In: Modeling Foundations and Applications, 8th European Conference on Modelling Foundations and Applications (ECMFA), Denmark. Springer, Heidelberg (2012)
4. Smolik, P.: Mambo Metamodeling Environment, Doctoral Thesis, Brno University of Technology, http://www.mambomde.com/MamboMDE.pdf (2006)
5. Reddy, S., Mulani, J., Bahulkar, A.: Adex – a meta modeling framework for repository-centric systems building. In: 10th International Conference on Management of Data, COMAD 2000, Pune, India. Computer Society of India (2000)
6. Ashby, W.R.: An Introduction to Cybernetics. Chapman & Hall (1956)
7. Lidwell, W., Holden, K., Butler, J.: Universal principles of design. Rockport Publishers (2010)

Teaching Object-oriented Programming using Object Benches: Practical Experience

Jakub Livovský, Miroslav Biňas, Jaroslav Porubän

Department of Computers and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia

{jakub.livovsky, miroslav.binas, jaroslav.poruban}@tuke.sk

Abstract. In this paper we summarize our long-term experiences with teaching object-oriented programming in university courses using teaching support tools called object benches. We describe four software tools: BlueJ, Greenfoot, Alice and Visual Studio's Object test bench. Common feature of those tools is runtime access to objects, their attributes and operations. We present our experience with using object benches in programming courses, we evaluate the above mentioned tools in context of teaching object-oriented programming and compare them with our OAT tool we have developed to support object-oriented programming teaching. The comparison is based on several criteria - solution's architecture, level of interactivity and domain specificity. As a result of the evaluation, we summarize the advantages and disadvantages of different approaches to design educational tools for teaching of object-oriented programming. We also discuss the main decisions behind the design and development of our OAT tool.

1 Introduction

Object-oriented programming (OOP) is relatively complex programming paradigm when we take into account the number of different concepts used in OOP [1], [2] (e.g. classes, constructors, attributes, operations, polymorphism, inheritance, subtyping). Therefore OOP teachers count on educational tools when learning OOP concepts and paradigm. A lot of research has been done in the field of educational software for teaching OOP [3] and many tools are available today [4], [5]. Amongst them the object benches are the most successful tools in teaching basic OOP principles.

Common features provided by object bench tools are runtime access to objects, their attributes and operations. Using object bench, users can create new instances of classes, access attributes of objects and execute operations on objects. User interface of those tools typically consists of three main components: standard class diagram well known from UML, object bench displaying existing/created objects and inspector providing access to attributes and operations. Using object bench the basic OOP concepts can be explained on practical examples. Object bench's users can practice class instances creation using constructors and can manipulate created objects and

their relationships. At the same time user can observe impacts of his/her actions directly within the object bench.

2 Experiences with Object Benches

During last few years we used a few object benches in OOP course at our university. Following section characterizes these tools and describes our experience with them.

BlueJ is an integrated Java environment specifically designed for introductory teaching of object-oriented programming [6]. It is developed by La Trobe University (Australia), University of Kent (UK) and Oracle. Its aim is to provide an easy-to-use teaching environment for the Java language. Anyway, its source editor is not comparable with industrial IDE in terms of functionality. There were a few unsuccessful attempts to integrate the BlueJ with professional IDE. According to our five years experience with BlueJ, students want rather use professional IDEs. The side effect of using BlueJ for too long during the courses is that students will start to use it for development of more complex projects later.

Greenfoot is an integrated Java environment based on BlueJ. It is also designed for introductory teaching of object-oriented programming. BlueJ is focused on development of simple 2D games [7]. It provides much more interactivity with objects than BlueJ directly in developed game. We tried Greenfoot for one week long introductory course of simple game development for teenagers and they adopted it very quickly. Because it is based on BlueJ it has same disadvantages as stated in previous section.

Alice is a product of multi-university initiative to create educational software for teaching computer programming in 3D environment [8]. However, complexity of the IDE and 3D world interaction can be seen too complex a for novice programmers.

For one year we also used **Object Test Bench** (OTB), the integral part of Microsoft Visual Studio (VS). Because of its tight integration with VS, it has the biggest potential of all mentioned tools. The main problem was to achieve interactivity with objects painted on canvas (because of single thread architecture) and support for encapsulation (private fields were visible). OTB VS was probably not seen by authors of Visual Studio as teaching tool but more as a testing tool. If you wanted to use OTB VS, you had to install Team version of VS. However, OTB is no longer supported in VS from version 2010.

After a brief description of object benches, let us compare them from different viewpoints in the following sections. The comparison is concluded in the **Tab. 1**.

Architecture

The one of the most important thing when comparing object benches is its architecture. An object bench could be developed as a standalone integrated development environment for developing software applications (IDE), an IDE extension (plug-in), or a library. Let us look briefly at these approaches.

The most complex approach is to build an object bench as an IDE. While building the object bench as an IDE requires a lot of development effort it could be customized to the higher level providing all the features required for optimal teaching. Object

bench's designers throw away any specific features not related directly to teaching of object-oriented programming making it simpler to navigate and use for a novice programmer.

Nowadays, industrial IDEs are very complex tools with lot of features supporting and simplifying the development of software applications, e.g. syntax highlighting, refactoring, code completion, code formatting, software visualization, unit testing, source code versioning. It takes significant development effort when one decides to build such a complex object bench as a standalone IDE. We should take into account that developed IDE should incorporate the today major IDE features. Otherwise students will be disappointed with the quality of the IDE and they will criticize it. When team decides to build object bench as a standalone IDE then teaching benefits from IDE usage must clearly outweigh all other mentioned disadvantages taking the cost of development into the mind. The features for the object bench IDE must be selected very carefully with respect to the most important practices and tools used in the industry and teaching methods at the same time. Finally, designer should also realize that industry tools are constantly evolving, so it will be necessary to evolve the object bench IDE to stay in touch with current trends. In our comparison, the BlueJ and Alice are representatives of standalone IDE approach.

Another approach to build an object bench is to extend an existing IDE with required feature for learning of OOP. Using these approach authors of an object bench can reuse a whole existing IDE infrastructure. This kind of object bench could be quite complicated because it provides all the features from extending IDE. It is good idea to use a possibility of an IDE to hide unimportant features simplifying navigation and understanding of a novice user and use IDE as a platform. Otherwise student can get stuck with all the unnecessary details for learning. The Greenfoot and Object Test Bench for Visual Studio are the representatives of this approach. The Greenfoot is built on the top of BlueJ IDE while OTB for VS is a part of Microsoft Visual Studio.

The last presented architectural approach is to build object bench as a lightweight standalone library. The object bench will be used as a library while running the application allowing introspecting objects and modifying running application. The library itself is an IDE independent and can be embedded into any regular Java application as a common library. On the other side, such a tool will lack the direct navigation to source code and does not contain source code manipulation support. The main benefit is that user can work in any IDE without any restrictions. At the same time tool and IDE can evolve independently.

Of course, it is easy to imagine that architectural approaches could be mixed. It is possible to develop object bench as a library and IDE extension at the same time benefiting from both approaches, i.e. lightweight architecture and source code navigation.

Interactivity

This property defines the possibility of changing object's attributes and executing the object's method while the application is running in background. Only Object Test Bench for Visual Studio does not support this behavior because of its single threaded implementation.

Domain

Some object benches are suitable for developing of all kinds of applications and some are domain specific. Object benches in our comparison are general-purpose except the Greenfoot designed for building simple interactive 2D computer games and Alice for developing 3D interactive scenarios.

Programming language

Since object benches are primary used for teaching of OOP they have to support one or more object-oriented programming languages in which students express their solutions. The Java is currently favorite object bench developer choice.

Visualization

Visualization of objects directly in an application can support the understanding of object programming concepts. The BlueJ and OTB VS do not have any special features/support for visualization of in-application objects directly in the application. On the other side, Greenfoot and OAT both have support for 2D visualization. Finally, the Alice is built around the concept of attractive 3D visualization. Alice's authors are using the 3D visualization concept as an important motivating factor for learning, providing natural visualization of objects.

Table 1. Comparing object benches

Tool	Architecture	Interactivity	Domain	Programming language	Visualization
BlueJ	IDE	Yes	general	Java	no
Greenfoot	IDE extension	Yes	games	Java	2D
OTB VS	IDE extension	No	general	.Net family	no
Alice	IDE	Yes	scenarios	alice / Java	3D
OAT	Library	Yes	general	Java	2D

3 The Object Access Tool

Coming out from our experiences with tools mentioned before, we created the object bench tool called Object Access Tool (**Fig. 1**) abbreviated as OAT. The OAT was primarily designed for use with 2D Java games. 2D games naturally contain objects with graphic representation and this fact can support student's projection of objects via physical metaphor. For this purpose the OAT tool contains optional support for visualization of objects in both target application and object bench. Although OAT is primarily designed for 2D games, it is implemented as the universal library and it is possible to bind it to any Java application.

Architecture

For the OAT implementation we considered three different architectural approaches described in the previous sections. Right at the beginning we rejected the approach to create a new IDE. Like we stated before it would be a quite demanding task to

create and maintain an IDE that could compete with industry grade IDEs. Instead, we wanted to take advantage of those professional IDEs and avoid students' transition from educational IDE to a professional one.

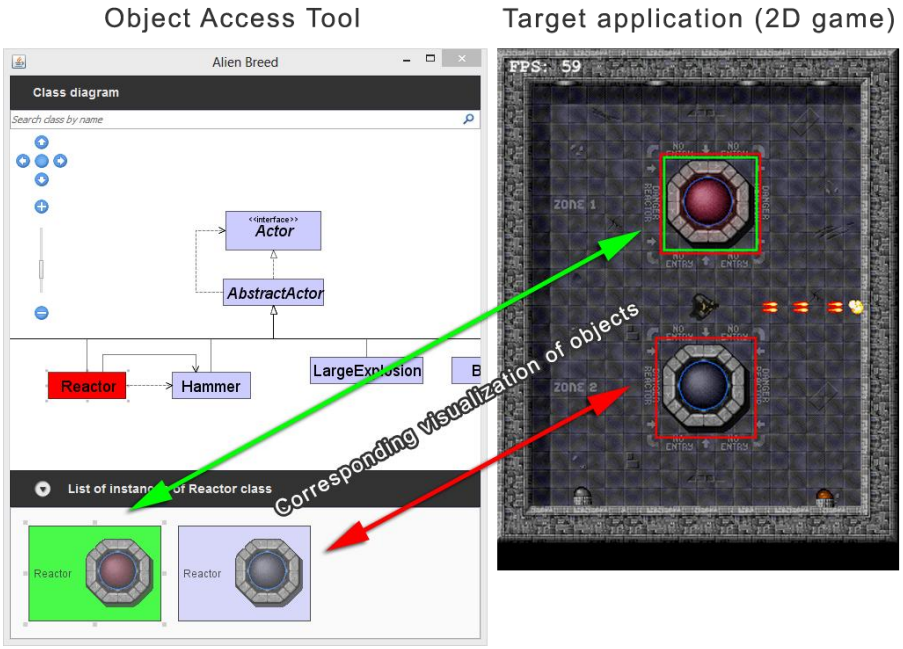


Fig. 1. OAT object bench with 2D game as target application

Both two remaining approaches have their advantages and disadvantages. Tight integration with an IDE is a significant advantage of IDE extension approach. In such a setting, user can directly navigate from an object bench to application's source code and vice versa. It means straight navigation between visualized object in an application and object's class sources code or navigation between selected attributed/operation in inspector and its source code. Tight integration would also allow object bench developer interact with many IDE functions, e.g. refactoring.

The advantage of a standalone library approach is in its lightweight infrastructure and minimum of dependencies. Library can be used with any development environment and is not dependent on any specific version of an IDE. The library can be embedded into any regular application.

Target application binding

Another important issue of the OAT tool development was the binding strategy to a target application. We considered two different approaches: defining a set of interfaces or use of aspect-oriented programming (AOP).

The main benefit of using AOP is in its composability. Using AOP we don't have to modify the source code of an inspected application. Object bench application

adapter could be implemented as an aspect that injects all necessary code to target application. The fact that we can't explicitly define the contract between the OAT and a user application is one of shortcoming of this solution. Our goal was to make a universal tool, which could target various applications. Because the architecture of each application is different, it is not possible to write universal set of pointcuts and advices. They have to be modified for each specific target application. The problem is we can't formally define what has to be modified and how to modify it. This kind of modification would not be possible without detailed understanding of OAT implementation.

OAT was primarily designed to support teaching object-oriented programming - for example, to test the new classes implemented by student in an existing project. With help of object bench student can create and test instances of his classes without having to modify the existing source code. Using AOP in this case would require the aspect language compiler to be available in student's environment due to weaving into new classes. It also should be noted that the aspect-oriented programming is not as widespread as, for example, object-oriented programming. A requirement to use AOP for binding with target application could discourage people from using OAT tool.

On the other hand interfaces are providing a method to define communication requirements between object bench and target application. Our solution consists of two interfaces specifying the contract between OAT and target application:

InspectableApplication - defines the communication from object bench to target application. As a consequence it also specifies functional requirements for target application. The target application adapter must implement this interface in order to OAT can access the application.

ApplicationListener - defines the communication from target application to object bench. Target application propagates events and changes to object bench class through calling this interface's operations.

The form of *InspectableApplication* interface implementation is left to the user. In our sample solution (**Fig. 2**) we implemented *ApplicationListener* by the adapter class - *ApplicationAdapter*. Adapter is extending target application with necessary functionality whilst monitoring changes in application and propagating them to observer implementing *ApplicationListener*.

Keeping track of all objects existing in target application is one example of a problem in which AOP would be more useful than interfaces. Object bench in OAT needs access to an up-to-date list of application's object to work with. OAT tool was designed primarily for use with 2D games which is a specific category of target applications. Majority of 2D games contains one class representing the game world. Game world holds a list of actor belonging to this world. In this case it is relatively easy to implement this feature in Adapter.

The use of OAT tool is not limited only to games. In applications without central list of objects where objects are created on many different places, it would be much more complicated to keep track of all objects in adapter. This is where AOP comes in handy. With AOP it is possible to target all constructors executions to monitor creating of objects.

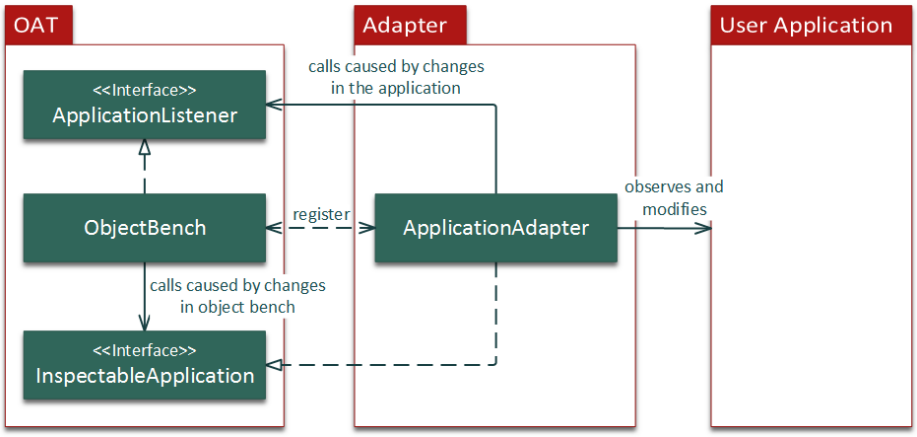


Fig. 2. OAT architecture

Saving the application state

One common problem of object bench tools mentioned in first section is that they do not keep the state of application. When a change in source code of target application is made, they need to recompile source codes and restart application. Which means the state of application is lost. All objects created with object bench are gone and they have to be recreated again by user.

We are trying to keep the state of application in our solution. Our solution needs to recompile and restart between changes too, but we are using the fact that the target application in our sample solution is a game. The state of game is represented by the state of its actors. We are using serialization for saving the state of actors between restarts. The main problem with using serialization is that our solution allows users to expand the target application. Due to this fact, we cannot rely on compliance of their codes with the naming conventions or the presence of annotations or default constructors.

Three approaches to serialization were considered - XML serialization (Java beans), standard Java serialization and custom serialization solution. The XML serialization is not suitable, because it requires presence of getter / setter pairs of operations with standardized naming. This solution would require modifying the target application so it satisfies this condition and we cannot guarantee this requirement in source codes added by users.

The problem with using Java serialization is that it is impossible to de-serialize saved object the class file was changed between application restarts. Java serialization calculates the hash code of the class files, and if it finds that they have changed, it won't de-serialize the object. With Java serialization we can restore only objects whose code has not been changed.

We could implement our own solution for serialization into XML. By using reflection we could store the attributes and their values during serialization of objects and

we could re-set them during de-serialization. Thus, we could reconstruct all attributes that have not been changed. If only operations in the class were changed, we would be able to de-serialize the entire object to its previous state. The remaining problem is how to create a new object during de-serialization. User can create a new class without an empty constructor. In this case we would not know which constructor and arguments values to use for de-serialization.

4 Conclusion

In this paper we summarized our long term experience obtained during lecturing object-oriented course with object benches at our university. Object benches are important tool for OOP teaching, but they lack support known from professional IDEs. We presented the design of our object access tool with the aim of minimizing all of the disadvantages of existing object benches we have experiences with. If you decide to use our OAT, you will get traditional object bench tool for your favorite IDE with no extra limitations for target application you want to create.

Acknowledgement
This work was supported by KEGA Grant No. 021TUKE-4/2011 Platform for integration of study guides and tools within the learning process.

References

1. N. Liberman et al.: Difficulties in Learning Inheritance and Polymorphism. In: ACM Transactions on Computing Education (TOCE). vol. 11, n. 1, article no. 4. Februar 2011.
2. A. Robins et al.: Learning and Teaching Programming: A Review and Discussion. In: Computer Science Education, 2003, Vol. 13, No. 2, pp. 137172.
3. A. Pears et al.: A survey of literature on the teaching of introductory programming. In: ACM SIGCSE Bulletin, v.39 no.4, December 2007.
4. S. Georgantaki and S. Retalis: Using Educational Tools for Teaching Object Oriented Design and Programming. In: Journal of Information Technology Impact, vol. 7, no. 22. 2007. p. 111-130.
5. M. Ben-Ari and N. Ragonis, Noa: On understanding the statics and dynamics of object-oriented programs. In: SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education. 2005. pp. 226-230.
6. M. Kölling et al.: The BlueJ system and its pedagogy. In: Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, Vol 13, No 4. 2003.
7. M. Kölling: The Greenfoot Programming Environment. In: ACM Transactions on Computing Education (TOCE), vol. 10, no. 4, article no. 14, November 2010.
8. W. Dann, et al.: Mediated transfer: Alice 3 to Java. In: SIGCSE'12 Proceedings of the 43rd ACM technical symposium on Computer Science Education. 2012. p. 141-146.

Development of the Dictionary Writing Software

Kamil Barbierik[1,2,3], Martina Holcová Habrová[3], Vladimír Jarý[1,2,3], Pavla Kochová[3], Tomáš Liška[1,2,3], Zdeňka Opavská[3], Miroslav Virius[1,3]

{kamil.barbierik,tomas.liska}@foxcom.cz
{holcova,kochova,opavska}@ujc.cas.cz
miroslav.virius@fjfi.cvut.cz

[1] Czech Technical University in Prague
[2] FoxCom s. r. o., Prague
[3] Institute of the Czech Language of the Academy of Sciences of the CR, v. v. i.

Abstract. Currently, a new monolingual dictionary of the contemporary Czech language is being prepared at the Institute of the Czech language of the Academy of Sciences of the Czech Republic. As part of this project supported by the Ministry of Culture of the Czech Republic grant within the National and Cultural Identity (NAKI) applied research program a dictionary writing software is being developed. Firstly, the reasons for developing a brand new software are presented. Then, the overall software architecture is discussed in details; the software is implemented as a web application based on PHP, HTML, and MySQL technologies, users interact with it via the web browser. In the following part, the main modules of the system including editing module, list of entries module, administration module, or output module are introduced. Finally, the current status of the project is analyzed and future plans that include development of native clients for the iOS and the Android platforms are presented.

Keywords: dictionary writing system, dws, database system, lexicography.

1 Introduction

The Department of Contemporary Lexicology and Lexicography of the Institute of the Czech Language of the Academy of Sciences of the Czech Republic, v. v. i., is preparing a new monolingual dictionary of contemporary Czech since 2012. Its working title is Akademický slovník současné češtiny (The Academic Dictionary of Contemporary Czech). It is a medium-sized dictionary with the expected number of 120,000–150,000 lexical units.

As a support for this project, a new Dictionary Writing System (DWS) is developed. The detailed specification of the requirements from the lexicographer's point of view can be found in the article A New Path to a Modern Monolingual Dictionary of Contemporary Czech: the Structure of Data in the New Dictionary Writing System [1].

We describe selected aspects of the finished parts of our DWS implementation devoted to the lexicographer audience. First we shortly mention existing solutions and the tools we have used. Next we shortly describe the structure of the data and the user interface of the application.

Finally we evaluate the production run of the finished part of the application.

2 Existing DWSs

There are several foreign commercial DWSs (e.g. TshwaneLex [7], IDM DPS [4], iLEX [5]) as well as open-source systems (e.g. the Mātāpuna Dictionary Writing System [6]) available. The DEB II [2], [3], dictionary editor and browser, is available for the Czech language.

The lexicographic team of the Institute of the Czech Language of the Academy of Sciences of the CR, v. v. i., that prepares a new monolingual dictionary of contemporary Czech language, considered to buy one of the available DWSs, to use one of the open-source systems or to develop our own system. One the criterion used was the DWS price; the other was the amount of the necessary adjustments for the significant specifics of the compilation of the dictionary and the time devoted to this task, if we decide for a commercial or open-source system.

Our final decision was to develop our own DWS that will fully respect the significant specifics of the compilation of the dictionary.

3 Tools for the Data Processing in the DWS

Our DWS is intended for the lexicographic team that may be spread over different locations and that needs to share common data that may be changed by any member of the team. Thus we decided to develop it as a multi-tier web based application with thin client.

The data processed by the DWS are mostly textual, but highly structured. We have chosen the MySQL database as data storage that constitutes the data tier of our application.

On the other hand, the user of the DWS has to be able to input the data into the database structure, edit the data, define the relations etc. The presentation tier of our application is based on the HTML forms with JavaScript client-side scripting; this is flexible enough to create forms containing different types of input fields and other controls for data management that check the user input before it is sent to the middle tier for processing.

The middle tier (or the application server) is the engine between the database tier and the presentation tier. It gets data and commands from the user interface, processes it with the cooperation of the database and sends the results to the database, to the user interface or to both. We have built the application server using the PHP programming language.

It is not necessary to introduce the above mentioned tools in detail.

4 MySQL Data Structure

The data in the database are stored in the so called tables that are interconnected through the so called relations. The main table of the whole data structure of our database is a table called "heslo" (i.e. lemma; see Fig. 1). It consists of 8 fields of different types according to the types of stored data.

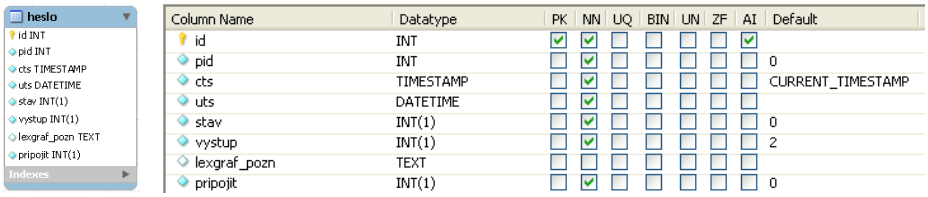


Fig. 1. The "heslo" table — icon and definition

The first field, "id", is the so called primary key; it is an integer number that uniquely identifies the lemma. What concerns the other fields, let us mention the "cts" field containing the time of creation, "uts" field containing the time of an update, the "vystup" field containing a flag indicating whether the lemma is intended to participate in the output and the "lexgraf_pozn" field containing the lexicographer's note.

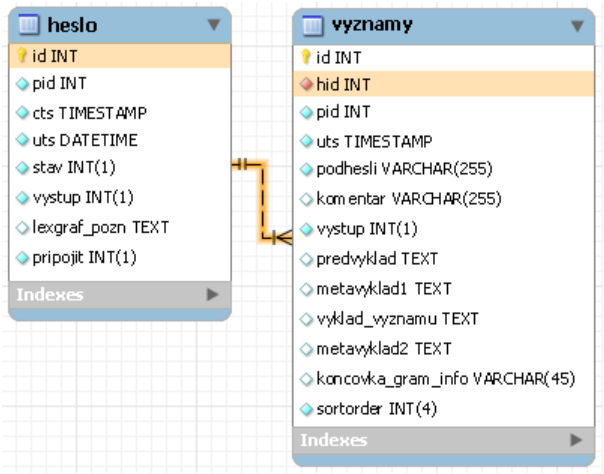


Fig. 2. Relation between "heslo" and "vyznamy" tables in the database notation

According to the requirements of lexicographers on the DWS listed in [1], it is often necessary to be able to duplicate some fields of the dictionary microstructure. The examples are the senses of the lemma: one word may have several meanings. This is implemented in the database as 1-to-many relation (see Fig. 2) using the foreign key "hid". This enables to attach several word meanings to one entry of the "heslo" table.

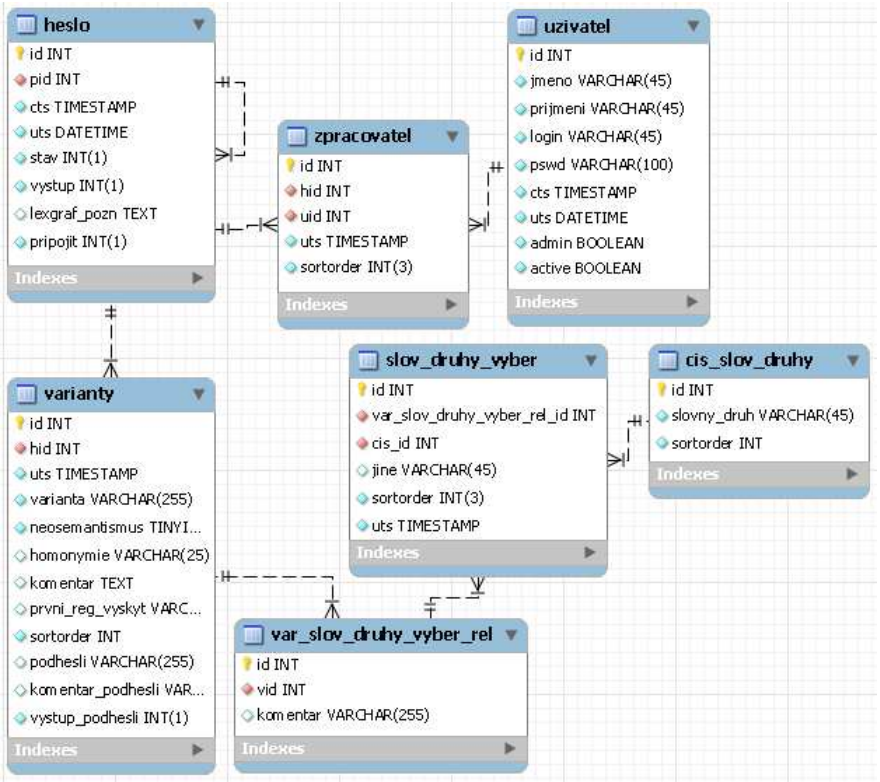


Fig. 3. Diagram describing the structure of tables containing the data for the List of entries module and their relations

5 HTML User Interface

The user interface of our DWS is divided into four main modules.

- List of entries module
- Editing module
- Output module

– Administrative module

We mention the List of entries and next we focus on Editing module in the following section.

5.1 List of Entries Module

The List of entries module contains information from several tables of the database. Therefore the relations between tables have to be used to extract and join relevant entries of these tables. Fig. 3 presents the so called entity-relationship diagram describing part of the DWS data structure containing all the information needed to generate the list of entries in the current form. The user interface of the List of entries module is shown in Fig. 4.

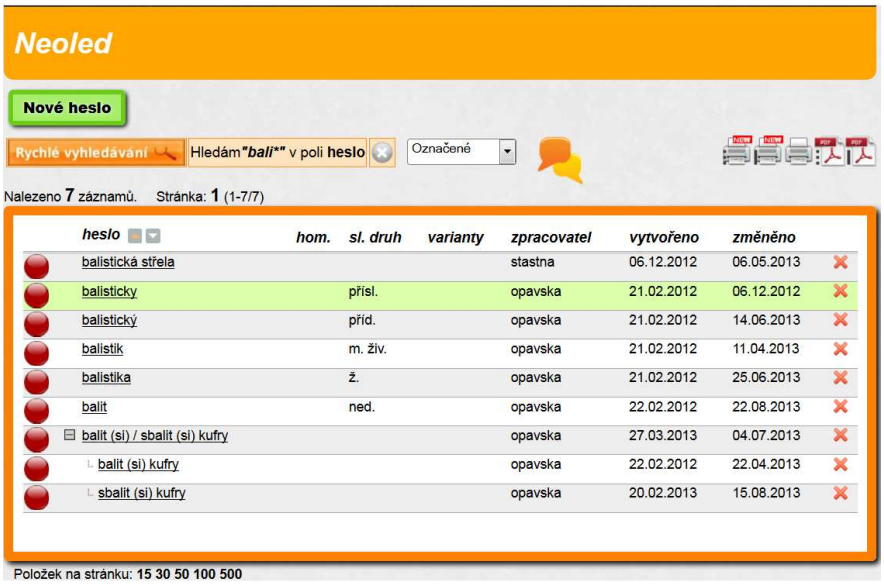


Fig. 4. User interface of the List of entries module

The Quick search function (Fig. 5), that is available in this module, is operating over the whole database. According to the selected field to search in, the corresponding tables are joined together and then the resulting structure is searched for matching entries. The output of the search engine is always the "id" of the lemma, i.e. the value of "id" column of entry in "heslo" table. The IDs that were outputted from the search function are passed to another function that generates the list of entries. This function puts together all respective information that belongs to the certain ID and finally outputs the list of entries that contains only these entries that matches the search value in the defined field.

The Basic filter function that is also present in this module contains several predefined filters which mostly operate over certain part of the database. The results of the filter function are again IDs of lemmas and the further processing is the same as described in the paragraph about Quick search function. The filters available here are as follows:

- My entries, which filters out all entries in which the currently logged user is mentioned as an editor.
- Selected entries, which return only entries that were selected before applying the filter. This filter uses rather the so called SESSION variables to keep the track of selected entries instead of saving them to the database.
- All entries, which deletes all constraints and returns all the entries without exception.
- Entries created/updated in interval, which returns all the entries that were created/updated in specified time interval.

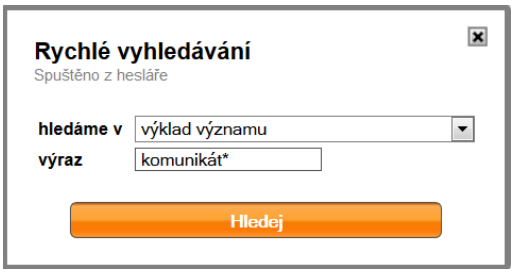


Fig. 5. The Quick search function

5.2 Editing Module

The Editing module is used to input or edit the data related to lemmas. It is basically a huge form composed of HTML elements like text input elements, text areas, select boxes, radio buttons, checkboxes and control buttons. The type of the utilized element depends on the type of data the user (lexicographer) will input.

Text Input Field. Text input (Fig. 6) allows collecting shorter text data that does not contain new lines. The text input allows for instance limiting the number of input characters. Data from such a field is stored in the database in the field of type VARCHAR. The example of data collected by this field is the pronunciation and its comment.

Data from the field of this type is then sent to the middle tier to be processed and saved to the database to the corresponding field. Specifically the mentioned

Fig. 6. Text input field

pronunciation (when it is a pronunciation referring to the variant) is saved to the table "vyslovnost_var".

As we can see on Fig. 7, there are fields "vyslovnost" for the pronunciation value and "komentar" for its comment; they are of type VARCHAR. The "vyslovnost" field is constrained to be nonempty, while the "komentar" is optional. This means that if an editor tries to save the pronunciation without any value, the system will notify him that it is not possible. The comment is optional, thus entries, where the pronunciation is properly filled and the comment is left blank, are valid.

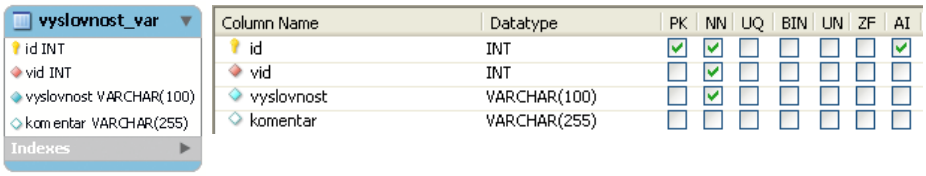


Fig. 7. The "vyslovnost_var" table — icon and definition

Text Areas. Text areas are multi line text inputs. The length of the text is not limited in any way. What is more, the majority of browsers allow resizing them by dragging a corner, thus the user can comfortably set its size according to his or her needs. Some of the text areas in our application automatically set their sizes to fit the text contained in them. This saves place when the text area does not contain any text; when there is a lot of text inside, the user does not have to scroll it and sees the entire text contained in the text area immediately.

The example of data collected by this input field is the lexicographic note (see Fig. 8).

The data from this input field is saved to database in the field of type TEXT. The lexicographic note is information referring to the lemma. It is therefore saved to the "heslo" table in the "lexgraf_pozn" field. As we can see on Fig. 1, this field is of type TEXT and is optional, thus an entry with an empty lexicographic note is valid.

For some textual inputs, it is necessary to allow the user formatting of the text in some way. This is provided by the WYSIWYG (What You See Is What

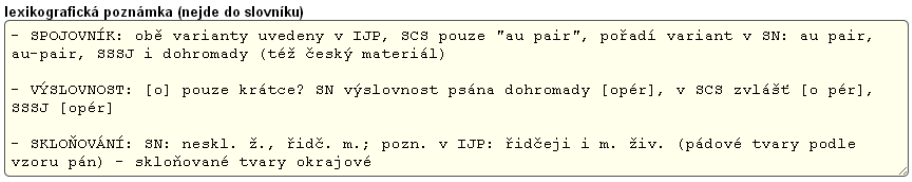


Fig. 8. Text area for the lexicographic note

You Get) form element. It is a text area input field equipped with formatting controls. This functionality is entirely provided by the JavaScript technology. The WYSIWYG function formats the text using HTML tags and displays it to the user in nicely formatted way. The user does not see these tags. Since the tags are textual information just like the formatted text, the process of saving data that contains formatting tags to database is the same as saving data from ordinary text areas. You can see an example of the WYSIWYG form element in the Fig. 9.

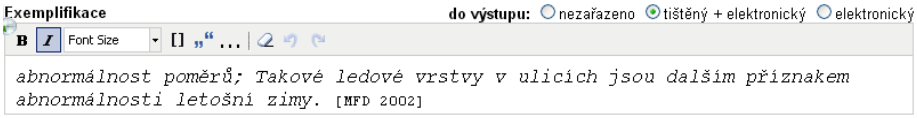


Fig. 9. The WISYWIG form element

The formatting options are: a bold text, an italic text, a font size, angular brackets (with the automatic smaller font size), Czech quotation marks and functions like “remove format” important when pasting the text from different sources, and also “undo/redo” function.

Radio Buttons. Radio buttons provide an ability to select exactly one among several options. The example is the field “vystup” tied together with the exemplification (see Fig. 10), which allows the editor to control whether certain exemplification will be present in the lemma output. Each option is represented by a constant value predefined in the application. The value of the selected option is saved to the database in the corresponding table field. The mentioned output state is saved in the “vystup” field of the “exemplifikace” table. The value to be stored in the database is of type integer. Thus the field type in the database is INT.

Select Boxes. The more complex form of element is the select box. Select boxes are used to fill in the information into the field by selecting from predefined

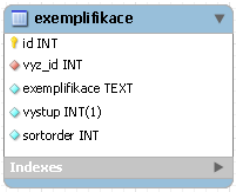


Fig. 10. The "Exemplifikace" (exemplification) table icon

values. These values are defined by administrators of the system. Thus, the values are prepared and managed by very few people, what helps to keep data consistent. Sometimes, of course, the editor will need to input the value that is not prepared in select box. This situation is also solved in the system.

The standard HTML select boxes do not satisfy our needs. Especially, they do not offer any option to add other value than those that select box offers or changing the order of selected values. Therefore we implemented our own select boxes by putting together several standard HTML inputs. Let us describe one example — the "type of lemma" select box.

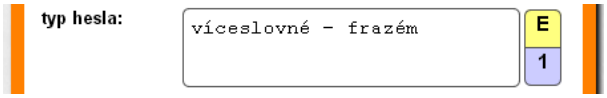


Fig. 11. New type of select box developed for our DWS

Selected values are displayed in the white window in the current version. In the case we see on Fig. 11, there is only one value selected; this is indicated in the little blue rectangle on the right hand side of the box. Of course, several values can be selected. But because there is a lot of other inputs in the form and we want to save as much place as possible, we only display the first two selected options. The number of options actually selected is indicated in the little blue window. To change the selection of entries in this select box we click the yellow "E" button, which brings up a popup window with the editing tool (Fig. 12).

All available options are shown in the left window. To select values, user has to move them from the left pane to the right one. When several values are selected and moved to the right pane, you can adjust the order of them by using the arrows on the right hand side. If the user does not find the value he or she needs in the left pane, he or she can select the "jiné" ("other") value and move it to the right window. After that he or she can write his or her own value in the "jiné" text field. When finished, the user pushes the "OK" button that changes the selection to the state set by user and closes the popup window.

The administrator manages the values of these select boxes from a separated module. He or she can add/remove elements and change their initial order. More-

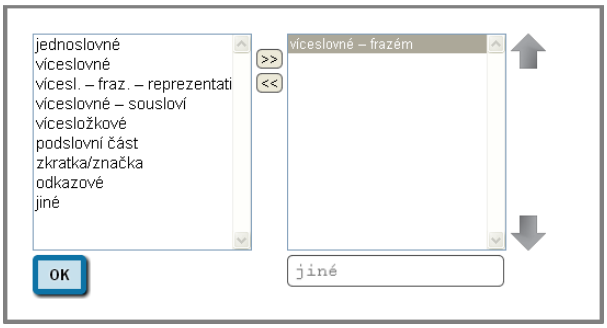


Fig. 12. Selecting more options

over, he or she can also monitor the statistics of using the "jiné" field and if a value is frequently used, he or she can decide to include it to the predefined values by a single click (on green arrow — see Fig. 13).

Fig. 13. Administration of the values for out select boxes and the statistics of the "jiné" ("other") value

The following Fig. 14 shows the data structure used for the values from select boxes.

The predefined values are stored in tables with "cis_" prefix. E.g. the predefined values for "typ hesla" select box by the administrator are stored in the table "cis_typ_hesla", as shown on the Fig. 14. The values chosen by the editor

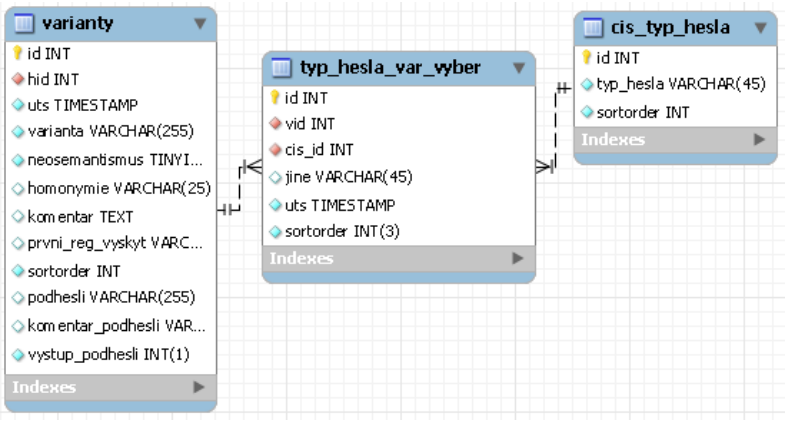


Fig. 14. Data structure for the values of the select boxes

are stored in the table "typ_hesla_var_vyber". Finally, this table is related to the table "varianty", which holds the information about variants of lemmas.

Cross-references. One very important part of the dictionary macrostructure is the cross-reference between lemmas. To allow lexicographer define the cross-reference, an implementation of a special form was necessary. This form consists of several input fields of different types.

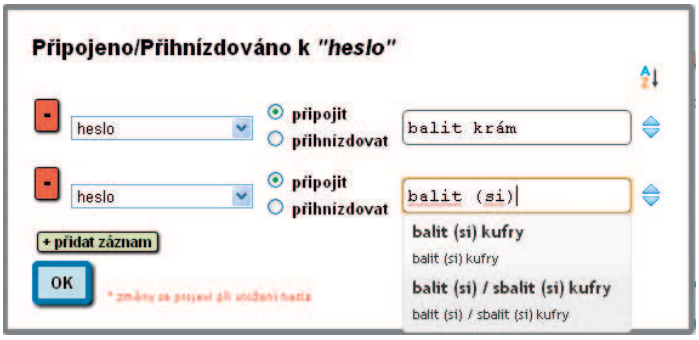


Fig. 15. Creating the relation

Two types of references are implemented: 1) run-on entries or references between one-word and multi-word lexical units (see Fig. 15) and 2) links to entries (see Fig. 16). Every relation has two ends: the master and the slave entry. Our DWS allows creating the relation from both of entries.

Fig. 16. The system offers the blind references

To create the run-on entry or references between one-word and multi-word lexical units, the reference from the master to the slave entry, we use the popup window, which is evoked by clicking the appropriate button in the main edit form.

We can select the place from which the slave will be referenced, the type of the reference and finally the referenced lemma in the form. When inserting the slave entry, the auto-complete function is offering existing lemmas. When we insert word that is not yet in the database, the blind reference will be created. In the future, when some user creates an entry that corresponds to the blind end of an already existing reference, the system will automatically replace it with the created one.

Similarly, links to entries are also edited in the popup window, when edited from the master entry.

When linking entries, we can define more places at once from which the slave entry will be referenced. Moreover, we can define places the slave will reference to. It is possible to select multiple places (word meanings) at once. The text field allows defining the slave entry. The auto-complete function is present in this form, too. Furthermore, we define the type of link by selecting one of the values proposed by the select box ("dok." for instance, see Fig.ă16).

The forms used to create references from the slave entries are as follows. They are very similar to these in popup windows (see Fig. 17 and 18).

Fig. 19 presents the entity-relationship diagram of the data structure storing the relations.

All the relations are stored in the "hesla_rel" table. Whether the relation is the "run-on entryŤ or the "link to entry", both are stored in the field "rel_type". Places referenced in the master entry are stored in the "hesla_rel_places" table and the places referenced in the slave entries are stored in the "hesla_rel_slave_places". If the relation is the "link to entry", we need to store the link type (for instance

Fig. 17. Interconnecting the lemmas: the slave entry

Fig. 18. Interconnecting the lemmas: the master entry

"dok." from the figure 16). The table "cis_hesla_rel_link_type" stores the predefined values for the select box in the HTML form. In the table "hesla_rel_link_type_vyber" the selection from the select box is stored. Even though only one link type can be selected in the current version, the data structure is prepared to store also multiple link types for the relation, if such functionality would be required in the future.

6 Conclusion

The DWS project was prepared during the year 2012. In 2013, the production run of implemented parts has been launched.

Up to now, 6,000 entries have been processed by the lexicographers of the Department of Contemporary Lexicology and Lexicography of the Institute of

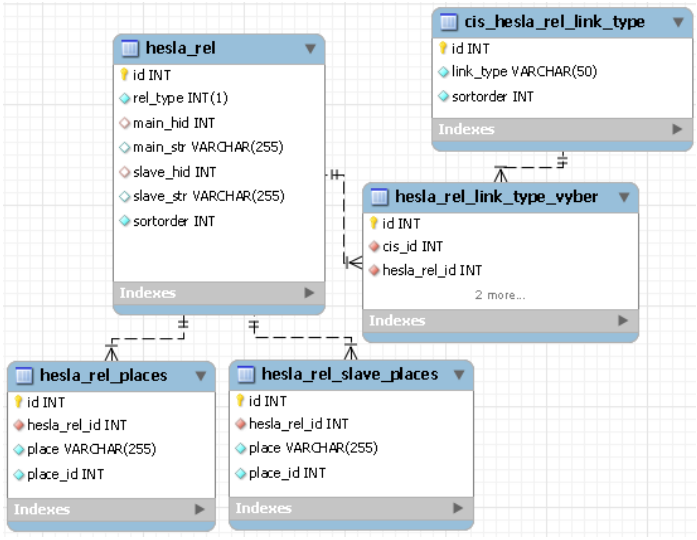


Fig. 19. Diagram of the data structure storing the relations between lemmas

the Czech Language using our DWS. This production run proved that the implemented part of the DWS fully complies the lexicographer's needs and requirements.

The DWS will be extended by the following modules:

- Editorial tool that makes easy the correction and proofreading process,
- Data tracking module, that serves to the tracing of all the important and interesting activities in the DWS,
- xFilter module, which is a complex search tool which provides fulltext search, exact search, interval search and other advanced search capabilities,
- Revision control system to keep track of the entry editing history,
- Dictionary web interface for the public in the form of the web application or an application for the mobile platforms,
- Automatic lemma processing module intended for the automation of selected processes.

Detailed description of the intended modules can be found in [1].

Acknowledgement. This work has been supported by the grant project of the National and Cultural Identity (NAKI) applied research and development programme A New Path to a Modern Monolingual Dictionary of Contemporary Czech (DF13P01OVV011).

References

1. Barbierik, K. et al.: A New Path to a Modern Monolingual Dictionary of Contemporary Czech: the Structure of Data in the New Dictionary Writing System. Accepted to the Slovko conference.
2. DEB II. http://deb.fi.muni.cz/index-cs.php
3. DEBDict. http://deb.fi.muni.cz/debdict/index-cs.php
4. IDM DPS. http://www.idm.fr/products/dictionary_writing_system_dps/27/
5. iLEX. http://www.emp.dk/ilexweb/index.jsp
6. http://sourceforge.net/projects/matapuna/
7. TshwaneLex. http://tshwanedje.com/tshwanelex/

Tool for Statistical Classification of Java Projects

Michal Rost*, Josef Smolka, Matej Mojzeš, and Miroslav Virius

Czech technical university in Prague, Faculty of nuclear sciences and physical engineering, Brehova 7, 115 19, Prague 1

Abstract. This paper describes design and implementation of software tool for statistical classification on a source code. In contrast to classic methods of defined structures detection, based on traversing of AST, statistical approach allows one to see the structures from a higher perspective. However, features, used for a subsequent classification, have to be collected from AST by conventional methods. Data for classifiers are collected from a XML representation of an abstract syntax tree using the XQuery language. This enables the tool to work potentially with any programming language. For the time being, the tool supports conversion of Java source code to XML AST. The tool implements various classification methods as well as methods for verification of trained classifiers.

1 Introduction

To transfer principles from image recognition to a source code, one has to deal with all the tedious work around statistical classification, e.g. feature definitions and data collection, features data preprocessing, classifier construction, training and validation. That all has to done, because there is no general tool (or library) to support statistical classification over source code. The first question is how to define a feature space and collect features data. To which attributes of source code should be features invariant? Is there equivalent of invariance to translation and rotation from image recognition? How to collect feature data? Should be data collected from text representation of source code, from abstract syntax tree (AST) [2, 5], from abstract semantic graph (ASG), or from runtime? This paper presents design and implementation of such a general tool that enables users to easily define their feature space, train, validate and use predefined classifiers. The paper also gives an example of tool application: features definition and classification using several predefined classifiers.

2 Requirements

Before the design of the software was specified, requirements have been formulated.

* Corresponding author, rost.michal@gmail.com

Functional requirements:

1. Manage features definitions and collect features data
2. Classify project's code
3. Manage classification results
4. Provide posterior analysis for classified data

Ad 1. Collect features from project's source code, store them in the corresponding objects and provide access to these objects.

Ad 2. Provide various statistical classifiers. Enable their configuration and classification of project's data types with the chosen classifier. Consequently, obtain results for all data types in the project and all the considered classes, where each result is represented by a probability that given data type belongs to one particular class.

Ad 3. Manage project's classification history, registered for all runs of all classifiers, because results from one uniformly configured classifier can vary over time.

Ad 4. Provide additional operations in order to measure quality of classifiers, perform cross validations, or apply balance criterions to classification results.

Non-functional requirements:

1. Modularity & extensionability
2. OS independence

Ad 1. Application has to be separated into individual components that will provide their interfaces to the rest of the application. This allows easy interchangeability of module implementation, or simply adding a new implementation.

Ad 2. Since java [6] is an OS independent framework, the designed tool is required to be also OS independent, in order to integrate it into Java IDEs in the future.

3 Tool design

The first non-functional requirement "modularity & extensionability" could lead to use of some modular framework like OSGi. But since the modularity and extensionability is not required from a viewpoint of an user (user plug-ins), but from a programmer's viewpoint, and considering the size of the project and a realization team (small), use of such complex framework is unnecessary. Desired goal can be easily achieved by clean object design. Four main components have been identified during the design phase.

The first component, the collector, is responsible for procession of source codes and collection of features data. A source code is parsed [2] and an abstract syntax tree created. Features are defined as mapping , where is a tree and is the set of real numbers. A set of collected data for a feature space can be defined as . The should be from an interval , but it is not a necessity as data are typically

normalized before further use. Let is a sub tree of connected with through a node . Let is created from by disconnecting from and connecting it to a node , so the only change is translation of sub tree . If the equations , where is a tree stripped of every occurrence of , are valid and then we say that the feature is invariant to translation. In contrast to image recognition, where an apple is apple regardless of position on picture and thus invariance to translation is a desired property of well-mannered feature, in source code recognition, position (context) can be significant. The big question during the design time of the collector component has been the level of detail on which the features should be collected. The consensus is to relate all features to data type definitions. Even features that describe relations between data types can be attached to particular data type. Such features are considered as descriptors of the data type neighborhood.

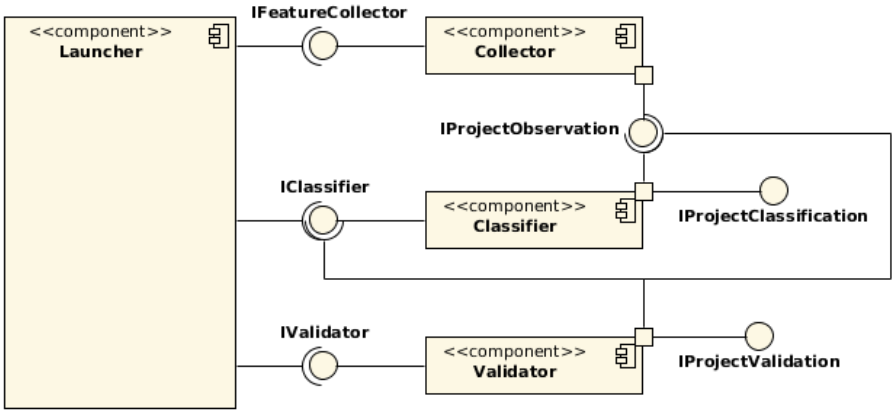


Fig. 1. A component diagram of designed tool. Four main components have been identified: *collector*, *classifier*, *validator*, and *launcher*.

The second component, the classifier, is the core of the whole solution. Classifiers can be either simple or compound, where a compound classifier consists of two or more other classifiers and a balance criterion. The balance criterion is a judge among the sub classifiers and does the final decision. A classifier is responsible for identification to which category an observation belongs to. Formally, a classifier can be understood as function , where is a dimension of the feature space and is a set of classes. To distinguish between different meanings of the term class (java data type vs. classification class), we would use data type for the first meaning and class for the second one. The decision is based on feature vector passed to classifier from the collector mentioned earlier. As the tool is intended to carry out statistical classification, each classifier has to be trained first by supervised training before application to real data. Unsupervised

learning (clustering) is not intended for the time being but can be easily added later.

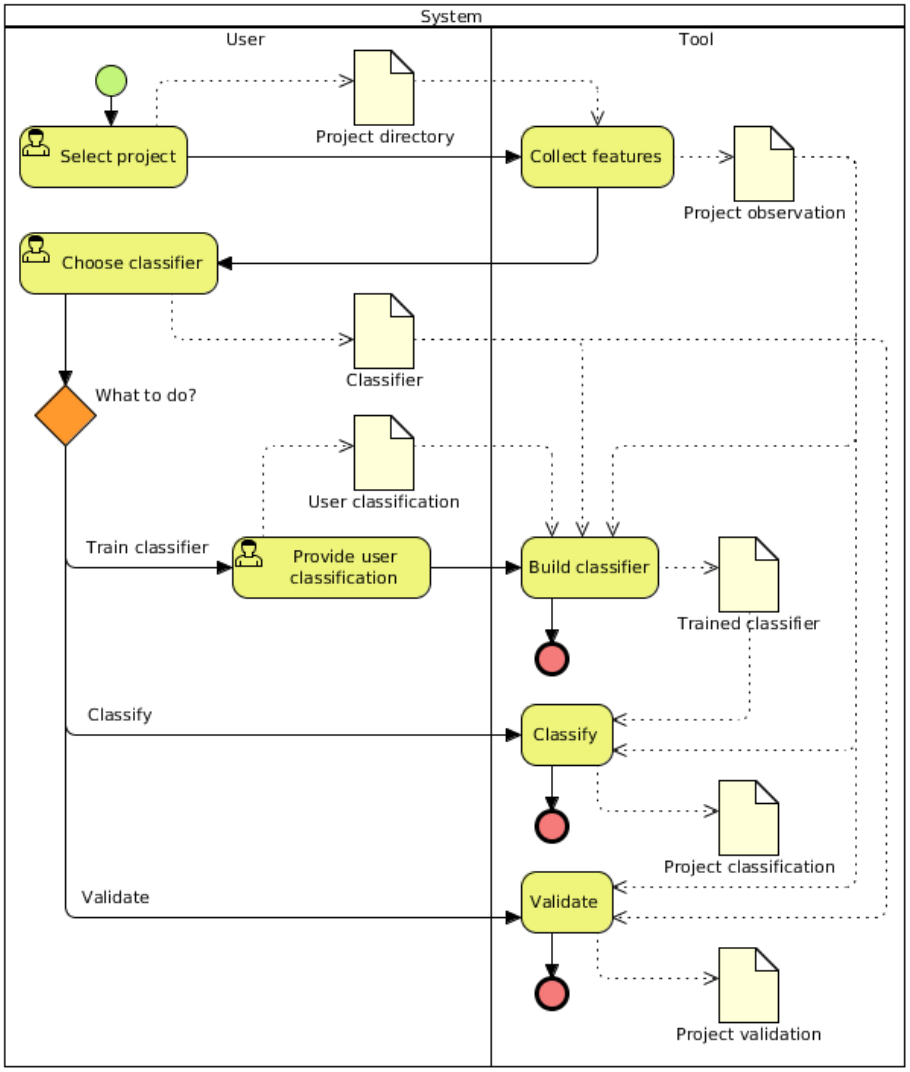


Fig. 2. BPMN diagram of users interaction with the tool.

The third component, the validator, is used for regression model validation, particularly a k-fold cross validation. This is a process of determination how results of a statistical analysis will affect independent data sets. During k-fold validation a project observation (a set of features of each data type) is split into

k disjoint subsets, then subsets are utilized for training of a classifier and one subset is used for validation (testing). Cross validation is finished, after all k subsets were used for validation.

The last component, the launcher, represents only a layer that performs top level operations over Classifier, Validator and Collector components. It will allow user to start classification or validation and configure their parameters.

4 Tool implementation

A core of the proposed tool has been implemented in Java. The collector is using parser component from Eclipse. An abstract syntax tree obtained by the parser is traversed by the visitor pattern [4] and XML representation in DOM format is created, where nodes in DOM has still reference to original AST nodes. This approach enables features to be defined as transformations of XML document. The tool uses XQuery [8, 7] as feature definition language, specifically an implementation from the Nux library as Java does not provide XQuery implementation. The collector crawls the directory structure of a project, parse the source code and create XML AST for each found data type. The tree is then stored in a database to be accessed later by XQuery scripts. Features data are collected by executing defined features scripts for each data type in the database. The transformation is carried out with particular data type, but the script can easily access AST of others data types to look up additional information.

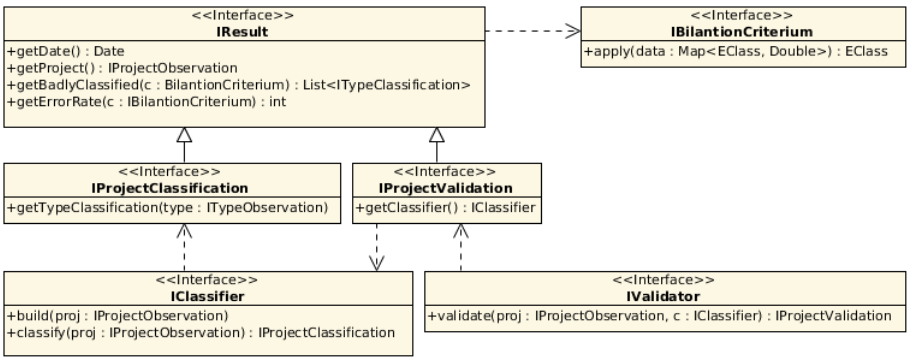


Fig. 3. A class diagram of classification and validation interfaces.

The classifiers are implemented as layer over Neuroph [3] and Java-ML [1] libraries. Neuroph library is used for ANN classifiers and supports several common ANN architectures: perceptron, multi-layer perceptron, rbf network, neuro fuzzy perceptron, Hebbian network, and others. The Java-ML library provides implementation of Support Vector Machine classifier, k-NN (k nearest neigh-

bours) classifier and Naive Bayes classifier. Other classifiers as LDA and QDA are to be implemented in the near future.

The validators are implemented as classes that realize interface IValidator (Figure 3) and they allow to perform validation of a given project observation using a given classifier. Results from both classification and validation processes are represented by a common interface IResult, which, among other, allows calculating error rate of performed process. Value of error rate depends on how the best classified class is selected. This selection is performed using balance criterions, represented by interface IBilantionCriterium.

At present, the tool has a text user interface (TUI), provided by Apache Commons CLI library. This allows user to select a required classifier as well as other options through command line parameters. User settings are subsequently passed to the launcher. In the future graphical interface is planned to be implemented.

5 Application example

This section gives a brief explanation of how the designed tool works.

At first project for classification has to be selected and passed to the tool in a form of directory. Then features are collected for each data type in a given directory. Currently, there are 42 types of features collected by the tool; these features are divided into four major groups: expression features, statement features, member features and relation features. Expression and statement features are connected with expressions and statements in the project code, typical expression feature is for instance a number of instantiations within a definition of a one data type weighted by total number of expressions in the same data type. Typical member feature is for example number of public, non-static setters and getters in a selected data type weighted by total number of methods in the data type. Relation features depict a relationship of a data type with its surroundings; this kind of feature is for instance a logical value, which is set to true, if data type uses his direct parent type as an attribute.

Example: Number of class instantiations within a definition of a data type

```
for $type in .//type
let $norm := count($type//expression[@expression-type =
    'class-instance-creation'])
let $res :=
  count(
    $type//method[@constructor =
    'false']/body//expression[@expression-type =
    'class-instance-creation']
  ) div (if ($norm > 0) then $norm else 1)
return flib:result($featureId, $type, $res)
```

Example: Number of getters/setters

```
for $type in .//type
let $res :=
  count(
    $type//method[@constructor = 'false' and @public = 'true'
    and @static = 'false'
    and (astlib:is-method-pure-getter($type, .)
    or astlib:is-method-pure-setter($type, .))]
  ) div flib:method-norm($type)
return flib:result($featureId, $type, $res)
```

Example: Direct parent as an attribute

```
for $type in .//type
let $superTypes := ($type//supertype/name/text(),
  $type//super-interfaces/interface/name/text())
let $res :=
  if (
    count(
      $type//field[not(astlib:is-type-primitive(./variable-type)
    ) and $superTypes = ./variable-type/name/text()]) > 0
  ) then 1 else 0
return flib:result($featureId, $type, $res)
```

After features are collected, they are passed to the trained classifier, where all project's data types are processed and it is estimated, which class is the most suitable one for a given data type. At the moment, there are ten classes recognized by the developed tool; each class refers to a one special kind of data type. These classes are: Adapter (adapts one interface to the other one), Bean (encapsulates data and provides access to it), Builder (creates various configurations of a particular type), Composite (represents a recursive type composed of attributes of the same type), Constant (contains declarations of constants), Decorator (adds additional functionality to a given type), Factory (creates new instances of a given abstract type), Proxy (changes implementation of a given type), Utility (provides static utility methods), Worker (performs operations with instances of other types).

If a user wants to train a classifier, he has to provide the tool with an additional configuration file containing user's classifications of data types in the input project.

6 Conclusion

In this paper, we have stated, that there is no general purpose tool for the statistical classification over source code and have proposed a specification for such a tool after formulation of functional and nonfunctional requirements. Design

of the tool was described and some implementation details and considerations were given. The tool was created within the research of application of statistical classification over source code. The intention was to simplify and speed up the process of feature testing and thus support overall reduction of the feature space as mentioned in the previous section.

7 Acknowledgment

This paper was supported by grants SGS11/167/OHK4/3T/14 and LA08015.

References

1. Abeel, T., de Peer, Y.V., Saeys Y.: Java-ML A Machine Learning Library. In: 'Journal of Machine Learning Research', (2009), vol. 10, pp. 931-934.
2. Aho, A.V., Lam, M.S., Sethi R., Ullman J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, (2006), 2nd edition.
3. Neuroph project, `http://neuroph.sourceforge.net/`
4. Pecinovský, R.: *Návrhové vzory*. Computer Press, (2007).
5. Smolka, J.: Refactoring tool for Java programs. Master's thesis, Czech Technical University, (2010).
6. Virius, M.: Java pro zelenáče. Neocortex, (2005).
7. W3C XQuery, `http://www.w3.org/TR/xquery/`
8. Walmsley, P.: XQuery: Search Across a Variety of XML Data. O'Reilly Media, (2007).

Lessons learned from a case study of scrum adoption at complex system integration project

Jakub Balada

Department of Information Technologies, Prague University of Economics

Abstract. Scrum has become the most used agile methodology, mostly for standard information systems development projects. In general, agile methodologies are considered as an alternative to standard rigorous methodologies, which are still recommended for complex system development. We have adopted scrum during implementation of complex system integration project of the Identity and Access management for Telefonica O2 Czech Republic. Lessons learned from this case study are the main purpose of this article.

1. Introduction

Agile principles represent a modern way of information system development, which brings more effective process of IS development. The basis of agile approach to the IS development is a maximal effort to satisfy customer requirements, which are usually changed during the project realization. In other words, it is the endless effort in helping a customer to increase its competitiveness through the in-time delivery of the service, which he really needs at a given time.

Agile methodologies have already established themselves in cases of smaller projects with short implementation time. According to the world wide survey [5], the most used agile methodology is Scrum. It is used by 52% of respondents, while another 14% uses a combination of Scrum and XP. In general, the agile development is used approximately in one half of all projects. One of main barriers according to this survey is project complexity which was mentioned by 30% of respondents.

In general, agile methodologies are considered as an alternative to standard rigorous methodologies which are still recommended for complex system development. The article Limitations of agile software processes [4] states the following conditions and limitations for adoption of agile methodologies:

- limited support for distributed development environments
- limited support for subcontracting
- limited support for building reusable artifacts
- limited support for development involving large teams
- limited support for developing safety-critical software
- limited support for developing large, complex software

In the conclusion of the [4] it is summarized that companies developing long-time complex projects would not be able to use agile principles in their current form.

1.1. Goals of this paper and methods of their achievement

The main goal of this paper is to describe benefits of the agile methodology Scrum in comparison with rigorous methodologies in the development of complex information systems. Advantages of this methodology are known enough in cases of certain project types. However, the goal of this paper is to prove benefits even in the case of complex IS development projects. The case study of the Scrum methodology adoption during the development of a complex information system is used for this purpose. Concretely, it is the project of the Identity and Access management setting for the Telefónica O2 Czech Republic, which controls accesses of more than 18000 employees to tens of various systems. The project was divided into two main phases, where the first one was managed in the rigorous way and the second one with the help of the Scrum methodology. Therefore, we are able to compare main characteristics of the project in particular phases and to prove success of the Scum methodology adoption.

2. Scrum

K. Schwaber and J. Sutherland are considered to be the founders of the Scrum methodology. They both, independently from each other, used principles of this methodology even in the 90s of the last century. They were also among authors of the Manifesto for Agile Software Development [1], in which 4 basic values and 12 principles of agile development are defined.

Based on these values, they developed the Scrum, which is a typical agile methodology. The main goal is functional software, which is delivered in regular short intervals and is ready to be deployed to the customer side. The cooperation with the customer is the key point - Scrum is based on the win-win model where requests for change are not a problem but are, in fact welcomed and will help the customer in his competitive environment.

2.1. Development process according to Scrum methodology

The development process itself is based on an iterative lifecycle model. Particular iterations are called Sprints and have always the same duration, typically one month. The result of each Sprint is tested and deployable increment. Requirements (user stories) which are defined at the beginning of the Sprint in the so called Sprint Backlog, are implemented within each Sprint. Sprint Backlog is the subset of the Product Backlog which represents the complete list of all requirements for the software being developed. The priorities of Product Backlog items are highly important. According to these priorities the tasks for the next sprint are chosen. This way the in-time delivery of the most needed services, is ensured.

3. Case study introduction

At first it is necessary to state the main parameters of the project in which the benefits of Scrum methodology are presented. It is the project of Identity and Access Management implementation for the Telefónica O2 Czech Republic, developed by the Siemens IT Solutions and Services (SIS). Author of this paper was the solution architect and after transition to Scrum also the Scrum Master responsible for the Scrum methodology adoption in this project.

The project consisted of the implementation and customization of the set of DirX products, which were developed by the mother company in Germany. The goal of the solution was a control of all the company employees (c. 18000) and their accounts in most of company's information systems (the solution serves about 260 000 accounts in 900 servers, 200 databases and 200 applications with 12 500 permissions). The final solution should have reduced the time needed to create employee's account, the work of administrators because of automatic creation and removal of all accounts in the most important IS, centralized control of users and their access rights, and last but not least should have increased the security including successful SOX audit.

On the supplier side, there were three subjects in the project. SIS – the prime provider (8 people), SIS Germany – the main supplier of basic products (support team), and the sub-provider, responsible for the graphical user interface integrated into the customer's intranet – Orchitech Solutions (6 people).

The sponsor of the project on the customer's side was the IT Security Department. However, the main portion of requirements arose from the IT Production Department, which is the main user of the system, and last but not least from the Business Department. As it was discovered during the project, requirements of these three sides were quite different, sometimes even opposing. The detail specification of requirements, which was the amendment of the contract, contained 130 functional and non-functional requirements.

The project was divided into two main phases. The first one started in September 2008 and should have lasted 12 months. Its aim was to achieve the first version of the solution, which should have been set into the production environment. The second phase should have lasted 5 months, and its aim was to connect other IS of the company, as required in the specification. So, the project should have been finished in February 2010 with the successful production of the second phase.

Since the beginning of the project the first phase has been managed by a standard rigorous methodology based on waterfall model. Analysis meetings, resulting in a large document describing the detail design of the first phase and briefer design of the second phase were held during the first 3 months. After that the implementation with 2 milestones presenting key parts of the solution followed. According to the plan – one month before the end of the first phase – the solution was passed to the acceptance testing. However, these tests discovered a big amount of misunderstandings in the specification, even when the detailed design was approved before. These problems resulted in prolongation of acceptance testing and consequent stabilizing operation. The first phase was officially accepted at the end of 2009 with a four-month delay followed by two sets of change requests.

After this experience, the steering committee asked the project team for analyzing the reasons for delays and the increasing costs for change requests and asked for suggestions of measures which would result in a successful realization of the second phase of the project. Based on the provider's recommendation, the change of the development methodology was approved for this purpose, and the Scrum methodology was chosen for the second phase of the project.

The main argument, thanks to which the steering committee approved the Scrum methodology adoption, was a big amount of change requests after realization of a first phase. Other arguments were not meeting deadlines in the first phase, better understanding of the project state, possibility to set new functionalities in month cycles, and not-used functionalities implemented in the first phase. This last point is a common problem, which could be eliminated with the help of Scrum. According to the Chaos Report 2009, issued by the Standish Group, 50% of implemented functionalities have never been used. It is due to the traditional approach, when a customer must specify all requirements, including the so-called nice to have, at the beginning of the project.

Based on problems during the first phase of a case study project, these main metrics were chosen to prove benefits of scrum adoption in the second phase:

1. Phase delay [%] = delay / fixed time x 100
2. Possibility to change req. [%] = new req. / final functionality x 100
3. Number of deployable releases per month = number of releases / phase duration
4. Customer visibility into project state per month = number of demo meetings / phase duration

4. Scrum adoption process

The case study, described above, has envinced most of limitations of Scrum adoption described in [7]. In this chapter, the process of transition from the rigorous way of project management in the first project phase to the Scrum methodology in the second project phase and consequences of this step are described.

The first phase of the project was finished by acceptance of two lists of change requests deployed into the production environment of the customer in April 2010 which meant 7 months after the planned date. The second project phase, where requirements were a subset of the original detailed project specification, should have followed. Because the project was developed with the help of rigorous methodology till that time, the rough solution design, which should have been implemented in the second project phase, had been already made. By the contract, the project was agreed as the so called fix-time, fix-price from the very beginning, So all tasks of the second project phase were already in the offer, later in analysis and design, evaluated in man-days. The steering committee agreed with an agreement, thanks to which the content of the second project phase could have been changed according to the actual needs of the customer. Dates and price were kept the same.

The first step of the transition from a rigorous methodology to Scrum was an initial Product Backlog filling. In our case, we were talking about the set of requirements from the original specification of the second phase, which were transformed into user stories. The initial velocity of one sprint was calculated by dividing the time estimates of the whole project phase by four month sprints. Next, the initial backlog had to be prioritized in order to choose tasks for the first sprint. This step was very complicated since it was a problem to define the product owner on the customer's side. But the Scrum Master strictly insisted on the nomination of one person on the customer's side. Finally, an external employee of the customer, who had been the project manager and had experience with development of such types of IS, was nominated. The product owner built a function counting priorities of particular user stories based on their benefits to the company goals according to the concrete departments (business, security, operations). With the help of this function he prioritized all the newly incoming requirements for the whole development phase.

Each sprint lasted one month, while the last week in the Sprint was reserved for customer acceptance. So, on average 16 days left for the development. During acceptance tests, the team was already preparing the next sprint. Namely, analytical meetings took place in order to define the design and estimates for the new user stories. Verification meetings focusing on the tasks in a sprint were held in the first two days of a sprint. After this, the design of the whole sprint was closed. After this verification, the customer strictly could not change the definition of tasks in this sprint. If this happened, the required change was added to the product backlog and if it had sufficient priority it was put into the next sprint. The first day of the acceptance testing, the presentation of new functionalities (sprint review) was

made and tests according to testing scenarios, which have been defined at the beginning of verification meetings, then officially started.

This approach is based on a Scrum Type B defined in [3], which uses overlapping iterations.

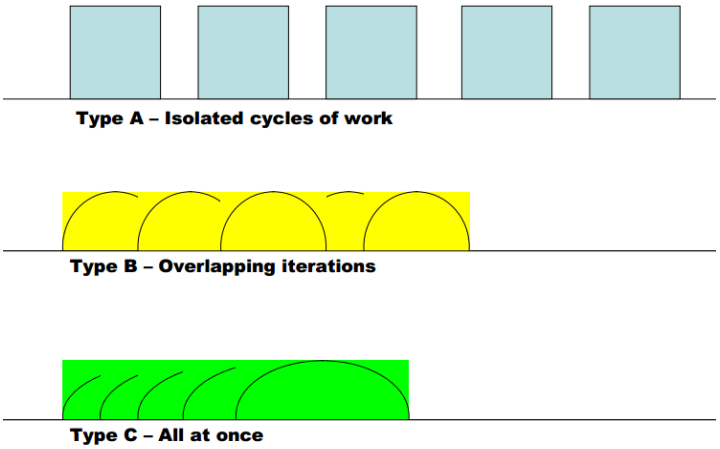


Figure 1: Types of SCRUM [3]

Only once within all 4 sprints, the increment was not deployed during the first regular system break after one week of acceptance testing. The reason was an enormous number of reported bugs, which led to one-week delay of the new version deployment. However, the next sprint ran in parallel within the defined terms. Because concurrently with the second phase development the system was already in use in the production environment a small relevant portion of minor bugs, discovered in tests, was solved by a standard problem management of the project, as there was no obstacle to put the new version into the production environment.

The last fourth sprint was accepted in October 2010 with eight-day delay, and after testing and acceptance of the whole second phase, the project was officially accepted and finished in November of the same year. Despite time delays of deadlines in the first phase, the project was marked successful with a next evolution at present.

5. Benefits from scrum adoption

With no doubts, the greatest benefit of the Scrum adoption in this project was a possibility to define the content of month sprints before their starts. After four sprints, there were implemented less than 45% of requirements of the original specification of the second phase. The reminding 55% was replaced by requirements that have newly arisen during the development process. At the beginning of

the second phase, user stories replacement process was agreed. In the case of a new requirement, another requirement with the same estimate and the lowest priority was omitted. As time went on, the customer already realized that the second phase can finish the product backlog with some user stories not done, and simply added newly defined required functionalities. The basis was the agreed velocity of particular sprints counted according to the original agreement. At the end customer added requirements with higher priority with estimation of 2.4 months (56.47 % of whole second phase). In a first phase, customer could change requirements only after acceptance of initially required functionality in two series of change requests, which took 3 months (15.79 % of 19 months of complete first phase).

Unofficially, the fix-time, fix-price contract was changed into fix-time, fix-price contracts of particular increments, defined in their starts. The indispensable provider's benefit of this principle is the monthly invoicing with positive influence for his cash flow. Mutual trust between a provider and a customer is the necessary condition for such process, at least because of acceptance of suggested estimates of new user stories.

Table 1 Comparison of phases by defined metrics

Metric	Phase 1	Phase 2
Phase delay	7 / 12 x 100 = 58.3 %	0.25 / 4 x 100 = 6.25 %
Possibility to change requirements	3 / 19 x 100 = 15.79 %	2.4 / 4.25 x 100 = 56.47 %
Number of deployable releases	3 / 19 = 0.16 per month	4 / 4.25 = 0.94 per month
Customer visibility into project state	5 / 19 = 0.26 per month	4 / 4.25 = 0.94 per month

The next already mentioned benefit was meeting the deadline of the second phase of the project. Only eight-day delay means 6.25% against 58.3% in a first phase (7 months delay after 12 months fix-time phase). Incremental acceptance and deployment of particular sprints into the production environment immediately after their implementations helped the deadline to be met a lot. There were only 3 releases in the first phase with change request series (that means 0.16 per month) against 4 in the second phase (0.94 per month).

A presentation date of new functionalities within a sprint was never postponed. Only once, all tasks of the backlog sprint were not caught up with the implementation deadline and 2 tasks with the lowest priority were moved into the next sprint and implemented without invoicing. That means 4 demo meetings in the second phase (0.94 per month) against 5 (2 development milestones, first release and 2 change request series) in the first one (0.26 per month).

It is difficult to compare the quality of the delivered IS of the both phases. The number of reported bugs from acceptance tests is comparable. Nevertheless, Scrum helped in identifying of bugs far sooner than in the case of a classic development and so the bug-fixing was not so time consuming with regard to the integration with other parts of the solution.

On the other hand the dependency between phases has to be taken into consideration. Development teams had some domain experience from the first phase and more demanding requirements were in a first phase.

6. Conclusion and future work

Information systems projects still fail to a great extent. Agile methodologies reached higher level of success in specific projects developed by small teams. However most of IT companies still refuse to use this approach at large-scale projects within complex environments.

Goal of this paper was to prove a benefit of Scrum adoption at large-scale projects. For this purpose project case study of Identity and Access implementation was described. This project is characterized by limitations discussed above which should avoid Scrum adoption. Nevertheless project was marked successful primary due to the change of methodology from plan-driven to Scrum using described practices solving those limitations at our project. This change helped to meet all the dates of the project, enhanced visibility into project realization, enabled deployment of new functionalities on a monthly basis and mainly helped to deliver functionalities which bring higher value to a business. All these benefits are proven by evaluation of defined metrics.

As a future work I would like to extend described practices for scrum adoption at large-scale projects within complex environment.

7. References

1. Beck, K. et al.: *Manifesto for Agile Software Development* [online]. 2001. Cited 2012-04-15. <http://agilemanifesto.org/>.
2. Schwaber, K., Beedle, M.: *Agile Software Development with SCRUM.* Prentice Hall, 2001.
3. Sutherland, J.: *Future of Scrum: Parallel Pipelining of Sprints in Complex. Projects.* AGILE 2005 Conference, Denver, CO, IEEE.
4. Turk, D., France, R., Rumpe B.: *Limitations of agile software processes*. In Proc. of the 3th International Conference on eXtreme Programming and Agile Processes in Software Engineering. 2002.
5. VersionOne: *The State of Agile Development Survey* [online]. 2011 Cited 2012-04-15. <http://www.versionone.com/pdf/2011_state_of_agile_development_survey_results.pdf>.

Functional Programming Constructs and Their Integration into Lessons of Object Oriented Architecture

Rudolf Pecinovský

University of Economics, Prague, 4 Winston Churchill sq., 130 67 Prague 3
rudolf@pecinovsky.cz

Abstract. In contemporary programming the significance of parallelizability of wide range of human activities is increasing. The development of various teaching tools is so quick that it is difficult to even watch it. Concurrently a natural endeavor is arising to install such programming constructs which would enable to transfer the main burden to the used libraries and frameworks and leave only a care for the best programming of the required business logic to programmers. A significant place among these constructs belongs to functional programming, above all the lambda expressions and data streams. This paper presents their basic characteristics and how these constructs can be included into the lessons on the object oriented architecture.

1 Introduction

The performance efficiency of computers increases at present above all due to the increased number of cores as well as the number of processors in the system. Of course, this efficiency is still increasing according to the well-known Moor's Law, but the efficiency of running programs does not follow the Moor's curve for a long time. With regards to the above mentioned growth of computer effectiveness the effectiveness of programs is subjected to the Amdahl's Law which says that the *T(n)* time needed for carrying out of the algorithm at *n* processors can be derived from the *T(1)* time needed for using one processor with the help of the following formula (1, 3):

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right)$$

where B presents the proportional representation of that part of the program that cannot be parallelized. If B=0, i.e. when the program cannot be parallelized at all, it would run with the same speed any time independently of how many processors would be at its disposal. If B=1, i.e. when the program will be parallelized by 100 per cent, the program would run quicker as many times as many processors would be at its disposal.

To provide the highest parallelizability of the program is usually not simple and mostly it requires a very good knowledge of not only processed issues but also of the programming constructs through which the parallel running is carried out. That´s why

we can register a strengthening pressure to transfer as much of these abilities and skills into the functionality of compilers, frameworks and libraries and to set free the programmers who then would be able to better concentrate for proposing optimal business logic. Therefore various algorithms as well as data structures are developed throughout the world which enable such transfer and make it easier (2, 3, 4, 5).

The side effect of these established algorithms and data structures is finding areas, the proposal of which can be significantly improved with the help of these algorithms and data structures, despite the need of its possible future parallelizability is not considered.

Several such constructs have been brought by the functional programming. The lambda expressions could be included into them as well as closures and data streams, which are gradually incorporated into programming languages proposed originally for different paradigms. Even the eighth version of Java language which is at present the most wide-spread used language not only in programming practice, but also in teaching, includes these constructs into its portfolio. Let's have a look how to include these novelties into teaching with the highest efficiency.

2 Extended concept of the `interface` construct

The new version of Java language comes with enhancing the interface usability with the possibility to define an implicit definition of certain methods together with the definition of static methods. This seemingly tiny extension has a huge impact at possibilities of future extending of the functionality of classes implementing the given interfaces.

In previous versions of the language, when you wanted to extend the possibilities of classes implementing certain interface without modifying all classes which implement the given interface, mostly you had to define the new interface, which should be implemented by those classes that would offer this extended functionality.

In case this extension would not influence the current code, it would be more advantageous to define the new class designed most often according to the design pattern *Utility Class* or *Servant*, the methods of which had instances of the extended classes among their parameters and defined the needed extension for them. However, in the newly designed interfaces it is sufficient to add methods with implicit definitions only, and the instances of all classes implementing the given interface will automatically be enriched by the functionality defined like that.

The newly added possibilities can be used also for modifying the definitions of current interfaces and for ensuing simplification of definitions of those classes which will implement them in future. For example in 8 the `Position` crate representing the position in double dimensioned space is defined. The `IMovable` interface characterizing objects that know both to get and to set their positions is then defined as follows:

```
public interface IMovable extends IPaintable
{
    public Position getPosition();
```

```
    public void setPosition(int x, int y);

    public default void setPosition(Position position)
    {
        setPosition(position.x, position.y);
    }
}
```

When the definition is set like that, the classes implementing the `IMovable` interface do not have to define all three methods because the implicit definition of `setPosition(Position)` method will suit to prevailing majority of them, and that's why they will simply take it over.

However, this making the work easier is only a side effect of the above mentioned extension of the `interface` data construct possibilities. The main advantage is the above mentioned possibility to extend easily the functionality which does not need any interventions into classes that already implement the modified interface. In the new Java version a significant part of interfaces gained the new methods with implicit definitions, whereas many of these interfaces have more implicitly defined methods than the original ones.

3 Functional Interfaces

Such interface is called functional that requires a definition of the only method from the implementing class. If it declares further methods it has to offer their implicit implementation. No other requirements are placed at functional interface. No parameters of this method are determined, neither the type of its return value.

Therefore the functional interfaces are often defined as generic types and the specification of parameter types and the type of their return value of their method is usually entered through type parameters.

There is a number of functional interfaces in the Java standard library. The older ones are mostly one-purpose – they were proposed for one specific usage (e.g. the interface `java.lang.Comparable<T>` defines the method serving for the comparison of the given object with the other one), but in the Java 8 a number of universal functional interfaces have been added. They have precisely defined only signatures of their methods, but their contract (i.e. what they will be used for) is relatively free. The main purpose of their introducing is the signature of the method declared in them according to which these interfaces mostly received their names – e.g. as follows (the overview shows full names of interfaces at the left side and the signature of declared methods in the right side):

```
java.util.function.Consumer<T>          void accept(T t)
java.util.function.BiConsumer<T, U>     void accept(T t, U u)
java.util.function.Suplier<T>           T get()
java.util.function.Function<T, R>       R apply(T t)
java.util.function.BiFunction<T, U, R>  R apply(T t, U u)
java.util.function.UnaryOperator<T>     T apply(T operand)
java.util.function.BinaryOperator<T>    T apply(T left, T right)
```

```
java.util.function.Predicate<T>         boolean test(T t)
java.util.function.BiPredicate<T, U>    boolean test(T t, U u)
```

4 Lambda expressions

The Lambda expressions represent the way how to define particular code part, which we would like to use in another part of the program, as an object. These code parts are defined by the compiler as instances of certain functional interface. The lambda expression value is written down in the following form:

```
parametr -> výraz
(parametry) -> výraz
parametr -> { příkazy }
(parametry) -> { příkazy }
```

On the left from the arrow there is always a list of parameters (it may be empty) and on the right there is an action which should be carried out. The following rules are valid:

- If there is the only parameter on the left, you do not have to put it into parentheses.
- If the compiler is able to derive the type of parameter, you do not have to quote it.
- If there is no parameter on the left, you have to quote the empty parentheses.
- If you evaluate certain expression on the right, you do not have to put it into braces.

The Lambda expressions behave as instances of functional interfaces, the method of which has the corresponding parameters and returns the value of corresponding type (the compiler arranges the relevant casting). If we need to remember them, we save them as values of the given functional interface. And when we cannot save them into the variable, we can pass them as the parameters of the methods.

Using of lambda expression can be demonstrated to students in a number of examples, the most often of which are presentations of sorting using the user-defined comparators. In 8 their application is demonstrated for using instances of the Repeater class which are able to repeat the entered code part as many times as is required. After finishing the required set of repeating the defined code is called which announces to the applicant that the required repeating has been finished. The signature of the discussed method is as follows:

```
public void repeat(final int times, Runnable action, Runnable finished)
```

5 Data streams

In computer technology the data streams are generally understood as sequences of data elements made available over time. They may acquire different, more specific meanings in various areas. Let's precise firstly in which meaning this term will be used in this paper.

In programming we mostly name by this term the objects mediating the transfer of data from the source to the target – we are speaking about the input, output and input-output streams. This paper does not deal with streams comprehended like that; it is focused on objects which are named by this term in the type theory and in functional programming. The term stream is used there in the sense of potentially endless list which (contrary to the classic list) does not save data, but only „knows about them".

Mostly those data are included into these streams at which we suppose their lazy evaluation. Therefore we independently define which data will be included into the stream (how the stream obtains these data), and how these data will be processed. Then, in a suitable moment, we ask the stream to apply the entered operations with these data.

These data streams can be introduced to students as an equivalent of a conveyor used during the assembly line production.

- At the beginning of the stream there is an input. In case of assembly line production it is a stock of parts, in case of a stream it will be a source of data - mostly some source container.
- Along to the assembly line there are workplaces where the trained workers carry out individual operations. In case of the stream we replace the workers by methods (more precisely by lambda expressions) that will carry out the required operation with the object. The processed object continues along the stream to the next workplace.
- At the end of the assembly line the accomplished product drops out, at the end of the stream we receive (at least we hope to receive ☺) the required result.

The streams differ from the collections and the arrays, as the main representatives of containers, in several important issues:

- They do not block any memory for processing data. The data are „flowing", the stream processes them and sends them further ahead.
- In most cases they do not influence the source data, which is very important during multiple processing of data by several processors, because then everybody call rely on the expected properties of input data.
- The planned operations behave similarly as the workers at the assembly line. They do not flock to the stock (container) to process all entered data but they wait until the processed object „flows" to them with the stream and then they look after them.
- The streams are not interested in the data input, if it is the final (classic container), or endless (data continuously flowing from certain external source).

The operations we order at the stream (stand of the assembly line) can be divided into two groups:

- **The intermediate** operations takeover the object, process it and send it further along the stream. Therefore the output value of current operations is again the data stream. Thanks to it we can concatenate these operations simply so that further operation will be applied at the result of the first one, similarly as we do it e.g. when further text is added into instances of the `StreamBuilder` class.

  When using concurrent operations we have to have on mind that some of them return their stream, but the others return the newly created stream. Therefore, generally, we cannot call the current method without remembering the stream which will return it. It is better to learn not to delete the returned streams and immediately call some of their methods, or at least save them.
- **The terminal** operations complete the stream activity and give over the received result to the surrounding program. The output value of these operations may be a collection, an individual object or void – this comes in case that the result is the required processing of an object.

The basic rule of the stream work is that during calling the methods incorporating current actions the stream only remembers what should be done at the given stop. But nothing is done at that moment, i.e. during incorporating the action. The activity will be started only at the moment of incorporating the finishing action. Only this will start the imaginary conveyor that transports the source objects from one workplace to the other, so that we could receive the required result at the end.

6 Internal iterators

Teaching with classical conception of processing the set of objects saved in a collection is based on using external processing of saved objects delivered through sequence iterators – the program asks the collection for an iterator which then delivers one saved object after another one and the program processes the given object. This processing is running quite under the charge of that program and the programmer who processes the program decides if he/she will use the possibility of parallel processing of more objects to achieve increased efficiency.

When using the streams the internal, batch processing of given objects is preferred. It means that the surrounding program does not require any iterator, but only provides the code (optimally the lambda-expression) describing how the elements in the stream should be processed. However, the organization of this processing is in charge of the stream and it is defined by the library authors. The program that asks for this processing informs the addressed stream if the processing of individual objects can be considered as sufficiently independent, so that they could be realized parallelly. The stream then decides how to divide processing to individual accessible processors.

The advantage of this access is multiple. As the most significant I would like to quote two of them:

- The auxiliary code that was responsible for controlling of iterations disappears from the program and the description of actions for processing the objects is far clear.
- In case some more sophisticated techniques of parallelization of executed activities will appear in future, it is not necessary to modify the program's solving application logics, but it is sufficient only to improve the stream library and thus automatically the program processing, which these streams use, will be made more efficient.

7 The new constructs and methodology *Architecture First*

The above analyzed techniques play into the hands of the *Architecture First* methodology (9, 10, 11, 12) which teaches in initial phases only the architectonic view on solving the problem and entrusts the creating of the necessary code to certain code generator. When using the above described constructs, employing of classic algorithmic constructs (as conditioned statements and loops) is considerably decreased because the area of tasks, for solving of which we need only the statement sequence, is significantly enlarged. Thus, at the same time, the area of tasks, for programming of which relatively simple code generators are sufficient, is also extended. The students can concentrate for studying and absorbing the key architectonic principles for a longer time, not being distracted by the necessity to get at the code level and define the operations by their own, because the used code generator does not suffice for their definition.

8 Conclusion

This paper introduced the new programming constructs that are included into the most used programming languages in the last time. It demonstrated their basic properties based on the Java language and showed how these constructs can be presented to students of introductory courses of programming. It also showed the impact of using these constructs on the architecture of the system and briefly outlined their most important advantages.

At the conclusion it also showed how including of these constructions according to the methodology *Architecture First* can influence the teaching and outlined its possible advantages.

References

1. AMDAHL, G.: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIPS Conference Proceedings 1967.
2. *Developing Parallel Programs – A Discussion of Popular Models*. An Oracle White Paper September 2010. Available at http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-parallel-programs-170709.pdf.
3. GOETZ B., PEIERLS T., BLOCH J., BOWBEER J., HOLMES D., LEA D.: *Java Concurrency in Practice*. Addison-Wesley Professional 2006. ISBN 978-0-32-134960-6
4. *Implicit parallelism.* Available at http://en.wikipedia.org/wiki/Implicit_parallelism.
5. NIKHIL, R. Arvind: *Implicit Parallel Programming in pH*. ISBN 1-55860-644-0
6. PECINOVSKÝ Rudolf: *Myslíme objektově v jazyku Java – kompletní učebnice pro začátečníky, 2. aktualizované a rozšířené vydání.* Grada Publishing, 2008. ISBN 978-80-247-2653-3.
7. PECINOVSKÝ Rudolf: *OOP – příručka pro naprosté začátečníky*. Computer Press, 2009, ISBN 978-80-251-2126-9.
8. PECINOVSKÝ Rudolf: *Java 8 – učebnice objektové architektury pro mírně pokročilé*. Grada, 2013, ISBN 978-80-247-4638-8.
9. PECINOVSKÝ, R, KOFRÁNEK, J. How to improve understanding of OOP constructs. Wroclaw 09.09.2012 – 12.09.2012. In: *Science Education Research Conference.* [online] Wroclaw : PTI, 2012, s. 19–24. URL: http://fedcsis.org/proceedings/fedcsis2012/pliks/136.pdf
10. PECINOVSKÝ R.: Principles of the Methodology Architecture First. *Objekty 2012 – Proceedings of the 17th international conference on object-oriented technologies*, Praha. ISBN 978-80-86847-63-4.
11. PECINOVSKÝ R.: Methodology Architecture First. *Proceedings of the international conference DidactIG 2013*, Liberec. http://jtie.upol.cz/clanky_1_2013/JTIE-1-2013.pdf
12. PECINOVSKÝ, R., KOFRÁNEK, J.: *The Experience with After-School Teaching of Programming for Parents and Their Children*. Las Vegas 22.07.2013 – 25.07.2013. In FECS'13 -- The 2013 International Conference on Frontiers in Education: Computer Science and Computer Engineering.

Author index