Paweł BASZURO*

BUILDING AN INTERPRETER FOR THE REY PROGRAMMING LANGUAGE USING DESIGN PATTERNS

Abstract

Rey is an educational programming language. It is designed for students as their first programming language, so it contains a simple grammar and a user friendly environment. For ease of usage, an interpreter was chosen as the execution model. Most of the well-known industry interpreters are created with performance as a key point. Building an educational tool requires a different approach. Rather than performance, extensibility is the main goal of the Rey implementation. Unlike many others, the Rey interpreter is implemented in a high level programming language using design patterns. This paper describes the process of building a language interpreter using Interpreter, Factory Method and Visitor design patterns.

1. INTRODUCTION

Teaching programming might be a very complicated process, especially when there are not many dedicated tools. Also choosing a language is very important. The description and primary purpose of the Rey language is defined in Rey an Educational Programming Language [1]. Rey can be considered an easier version of the C language, with key features:

- Syntax based on ECMAScript,
- Procedural model (only functions and variables),
- Weakly typed,
- Localized keyword set,
- Some of more advanced instructions are intentionally removed.

A language is only one side of the coin. The other side, maybe even more important is the environment. Rey is strongly connected to its environment (also named Rey), built with extensibility as a main characteristic. Integration with existing technologies, creation of new API functions or even definition of new language instructions is highlighted from early releases.

* Faculty of Computing Science and Management, Poznan University of Technology, e–mail: pbaszuro@wp.pl

Rey language

The language was primary designed during work on Paweł Baszuro's bachelor thesis (promoter Jakub Swacha). Next versions (still versioned 1.0) are published at http://rey-lang.ovh.org. The most actual are binaries with Polish localized keyword set.

A Rey program consists of only one source file. The end-user (student programmer) defines a program with one entry point (equivalent to the "main" function or static method in languages derived from C). The main function is parameterless and parenthesisless. In many cases, it is the only routine that user wants to execute. Inside "main" (same as function body) there are instructions. Out of the box instructions set contains:

- Variable definition – assignment of variable,
- Function call,
- Condition instruction (if instruction),
- Iterations (for and while instructions).

Independently of the definition of main point, user can define: functions, global variables and constants. To gain access to other system resources, the user might define a usage section, where a set of used modules is declared. It teaches habits and prepares for programming in other languages like C, C++, Java or C# (equivalent of include, import or using instructions).

The Rey language can be seen as a domain specific language. Checking conditions defined by Marjan Mernik, Jan Heering and Anthony M. Sloane [2]:

- Appropriate or established domain specific notations – Rey is an open framework for new instructions, to better adjust to concrete CS courses.
- Appropriate domain-specific constructs and abstractions – this point is connected with the previous one. For example definition of an array in Rey is a mixture of notation and best practices in C and C++.
- Use of a DSL offers possibilities for analysis, verification, optimization, parallelization, and transformation – Rey strongly differs from well known industry languages. It is an educational language so attributes of optimization, parallelization, and transformation are not considered as major elements.
- DSLs need not be executable – Rey programs, written with a localized keyword set, can be analyzed and evaluated on a blackboard, without the need for a computer.

According to Martin Fowler division of DSLs [3], Rey can be also considered an external DSL, where there is a division between the interpreted program, the parser, and the semantic model.

2. DESIGN PATTERNS COLLABORATION

Parsing the source code

Typical language analysis consists three phases [4]:

- Linear analysis – where text is split into tokens,
- Hierarchal analysis – where tokens are grouped into nested collections,
- Semantic analysis – tests if tokens have correct syntax.

One of the major changes compared to other interpreters, Rey encapsulates all elements of analysis in classes. Each token is converted to its object representation (e.g. while token

becomes an object with the type of WhileWordToken). Token objects have also a text representation field, which is the same as position in the source code file.

Any other analysis is based on the type of Token and its neighbors. Token object is passed to Parser to build an abstract syntax tree. A string representation is passed to an instance of TokenManager, which chooses a concrete IProducer (token producer) and creates a token instance. Those tokens are sent to the parser, via IScannerManager parameter object. Parser selection is made by an InstructionParser. It gets an object from a dictionary collection, where the token type is a key and an object (implementing IParserProducer) is the value. IParserProducer is a factory for instances of IParser objects. IParser with a Parse method builds an abstract syntax tree based on the type of the first token – in fact being an example of the Factory Method design pattern.

Having one factory gives a point of extension to register new instructions. Also delegating responsibilities to many interfaces makes the interpreter more modular. Parsers produce sets of IExpression interface instances, which "ride" user program interpretation to the Interpreter pattern.

Instruction execution

For given language, the Interpreter design pattern defines a representation for its grammar along with an interpreter that uses the representation to interpret sentences of the language [5]. Rey might be a typical implementation of the interpreter pattern. Each instruction has its own class, an instance of which represents each appearance of this instruction in the program's source code.

Main evaluation of instruction is being done by the Evaluate method, defined in the IExpression interface. It contains the method Evaluate which has a Context object as a parameter and returns an instance of IVariable. Evaluation of most instructions are linear, the cases where control goes to a certain point in a code file are a little more complicated. It is possible to distinguish three cases:

- Iterations – where control flow can be interrupted by continue (going to the beginning of the iteration) and break (exiting from the iteration) instructions. It is implemented by regular iteration instructions and exception handling. The body of the iteration contains dedicated try … catch blocks where those special situations are maintained via catching special exceptions,
- Function call – control is passed to a syntax tree stored in Context,
- Return instruction – similar to continue/break instructions, where an exception is being thrown with or without the return value of routine.

Debugging feature

The everyday programming process requires many program analyses. This can be split into many forms of software analysis, with characteristics of: correctness, usability, efficiency, reliability, integrity, adaptability, accuracy, robustness [6]. Also it can be expressed in code review, looking for code vulnerabilities or even checking code readability. Of course, after the analysis of used algorithms, it is possible to predict memory and CPU cycles consumption, and program correctness – using computational complexity theory. But still it is only static source code and algorithm analysis, without knowledge of program behavior at runtime. Also with a little knowledge of programming language, forecasting the implementation correctness might

be very hard. Programmers involve a debugging process to identify the root cause of an error and correcting it [6].

Begel Andrew and Beth Simon spot the importance of debugging in the transition from novice to expert software developer [7]. So teaching it from early beginning of programming lessons might make good habits for future programmers.

A good programming environment should support debugging – program runtime analysis. A debugger can obtain memory used by a program and show what instruction is being executed. Also it is possible to add a breakpoint – a dedicated place in source code, where program execution stops.

When using our own language interpreter with a complex object structure, it is hard to observe changes using standard tools. In addition the end-user wants to see only his or her own variables, the state and changes; not the objects used by the interpreter to supply that functionality.

One of the main differences between Rey 1.0 and 2.0 is the debugging feature. At any point of execution (main program or function call) a new context object is created. *Context* contains collection of all user and module variables and constants, which state end-user is interested in.

To support this functionality *IExpression* interface should be changed. Another operation is added by using Visitor pattern.

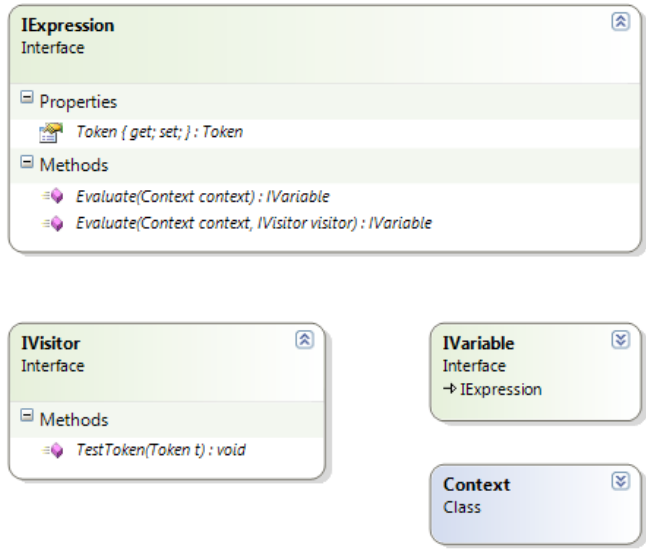

Fig. 1. Snapshot of interfaces and classes used the debugging in Rey

Fig. 1. shows the modified interface of *IExpression*, supplied with a new method overload, used by the visitor object. Each instruction during evaluation, calls the *TestToken* method on a visitor object, the executed token as a parameter. A class implementing the *IVisitor* interface should do all level analysis (e.g. checking the source code line or putting execution in wait state for some user action). When control is moved to a visitor object, it can access the static collection *Kernel.Contexts*. In this collection the implementation of visitor can access a context object related to each Rey function call (one context in the collection represents one function

call, with one additional context for the main program). Also a visitor object should be bypassed to inner *IExpression* objects.

In some cases a new method might only be a proxy to an existing implementation – for example when a new implementation is not ready, and there is a need to have library compatibility with a new version of Rey.

The debugging feature is very important for programming. Using easy to use tools, allows utilizing Rey on very early stage of CS courses. Also using the Visitor pattern, gives almost unlimited possibilities in controlling the interpreting process.

Extensibility

One of the biggest challenges during designing of the Rey interpreter is extensibility. Study of many educational tools used in programming, gives one result: most of them are hard to integrate with other technologies. Also it is not easy to build anything else on them (except open source software, where license allows changes). Rey is designed with client development as a main principle.

The client programmer can extend the set of functions, global variables, constants and instructions. Instruction implementation boils down to the creation of classes, which implement certain interfaces. Also the interpreter should have good mechanisms to inject external code.

Rey extensions are strongly connected with the environment in which it was built – Common Language Infrastructure (CLI) defined in ECMA-335 and ISO/IEC 23271. In general, code is compiled to an intermediate language, what in conjunction with the Common Type System allows building software in many programming languages independently of the operating system. Also one of the advantages of CLI is that, the code can be built to multiple independent binaries – executables (.EXE) and libraries (.DLL). Rey interpreter is a DLL library (for easy incorporation with other software), same as Rey modules. A module is defined in a class, with the ReyModule(module name) attribute. Static methods with the ReyFunction(function name) attribute are seen from Rey source code as a function. There are two additional attributes to methods:

- ReyInit – method executed when user explicitly defines usage of this module. In this method registration of new constants or global variables is being done.
- ReyPreInit – method executed as first, where instruction and token registration should be done.

All extension's discovery is being done by reflection, traversing all classes. Extension libraries should be placed in the lib directory. Sample code is provided with the interpreter (directory samples/developer).

3. PROTOTYPE SYSTEM

Nowadays we see a lot of new features in integrated development environments (IDEs). New features make them more capable to market needs, but at the same time more complex to learn by beginners. Many teachers experience the problem of that kind of difficulty. The BlueJ and Greeenfoot respond to the need of software for easy to use tool for teaching Object Oriented Programming [8, 9].

Unlike those projects the Rey2 reference environment is built for teaching procedural programming. In some applications, the procedural way of thinking is more vital than others – for example, when the target technology is embedded environment with C as a primary

programming language. So still, there is a need for easy to use environment concentrated on the procedural way of thinking.

The IDE is built around the Rey2 interpreter with similar features to BlueJ, like editor, visualization (debugging and flowchart) and testing. Also the Rey2 contains Wiki-like mechanism for text material publishing in view of Web 2.0 and E-learning 2.0. Users can publish materials formatted with a WikiMedia tags for describing the layout. But also new tags are developed to describe tests. Analogous to JUnit used in BlueJ [10], a simpler method is developed. JUnit is standalone testing environment well known in the industry. JUnit tests are Java code files, with methods corresponding to tests. The teaching with Rey2 reduces the necessity of writing testing code by a teacher. It is extremely important for simple programs, where teacher wants to test only the input and output data of the program. The structure of the test in an Extended Backus-Naur Form is presented on a formula (1).

$$
\begin{aligned}
&< input >::=="== input",< NEWLINE >,\\
&\{(< stringvalue >|< \text{int}\, value >|< doublevalue >),\\
&< NEWLINE >\};\\
&< comparator >::== (('INT',< \text{int}\, value >) |\\
&('DOUBLE',< doublevalue >) | ('STRING',< stringvalue >) |\\
&('\text{STRING\_CASE\_INSENSITIVE}',< \text{stringvalue} >)),\\
&< \text{NEWLINE} >;\\
&< output >::=='== output',< NEWLINE >,< comparator >,\\
&\{< comparator >\};\\
&< test >::==< input >< output >;
\end{aligned}
\tag{1}
$$

Where <intvalue> is valid textual form of an integer type, <doublevalue> of double type, <stringvalue> text within double quotation marks. Each input line corresponds to one input routine call, same as an output line for comparison. Also it is very important how to compare. Integer is the easiest type to compare; double must be compared with some margin for rounding. Also software must distinguish text comparison with and without case sensitivity.

After testing its own program, user can read results of each test. The test has ‘passed’ status when all comparator conditions are satisfied, otherwise ‘fail’. When the result is fail, then it is possible to change the source code and do the tests once again. When still user does not have exact result, he or she can debug the code. User can stop execution at certain point (see **Błąd! Nie można odnaleźć źródła odwołania.**) and see the execution context with a content of a memory (see **Błąd! Nie można odnaleźć źródła odwołania.**).

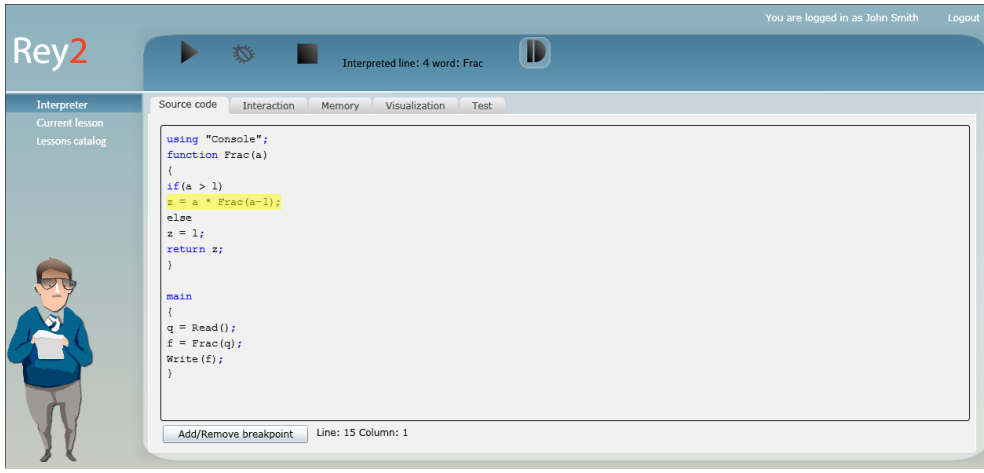

Fig. 2. The Rey2 system with editor under debugging shown, where current line is highlighted

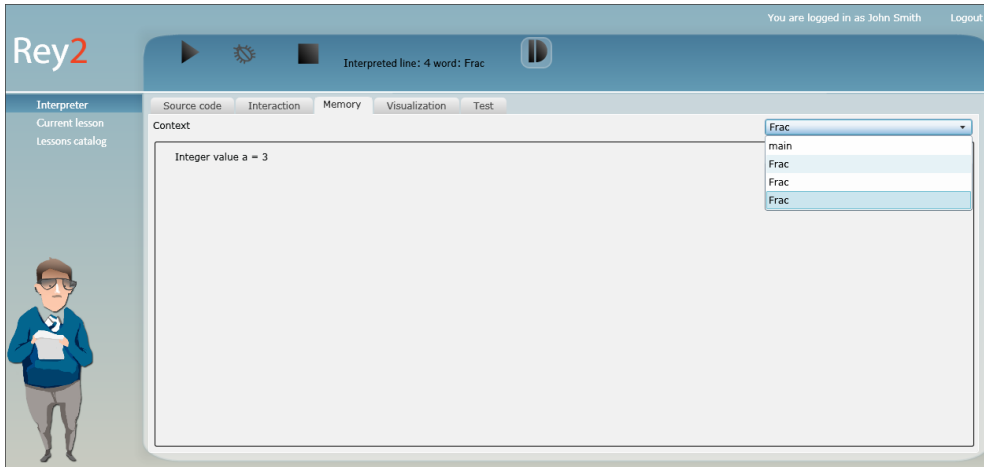

Fig. 3. The Rey2 system with shown a snapshot of the memory under execution

Current implementation is made in Microsoft Silverlight 2.0 – Rich Internet Application framework. Due to previous implementation (Microsoft .NET 2.0) and code compatibility Silverlight is chosen. What is important, the Rey2 application can be run on multiple systems from many popular internet browsers (Microsoft Windows Internet Explorer, Mozilla Firefox and Apple Safari). Having good abstraction level (described in chapter 2) it is possible to visualize different Rey2 features. Additional Rey2 environment is equipped with a console and graphics libraries (shown on Fig. 4), and algorithm visualization module (shown on Fig. 5). Screen assistant (called 'Bill', bottom left on figures) was introduced to show code errors and hints in a more user friendly form.

The Rey2 project will available in near future through official website: http://rey2.studentlive.pl .

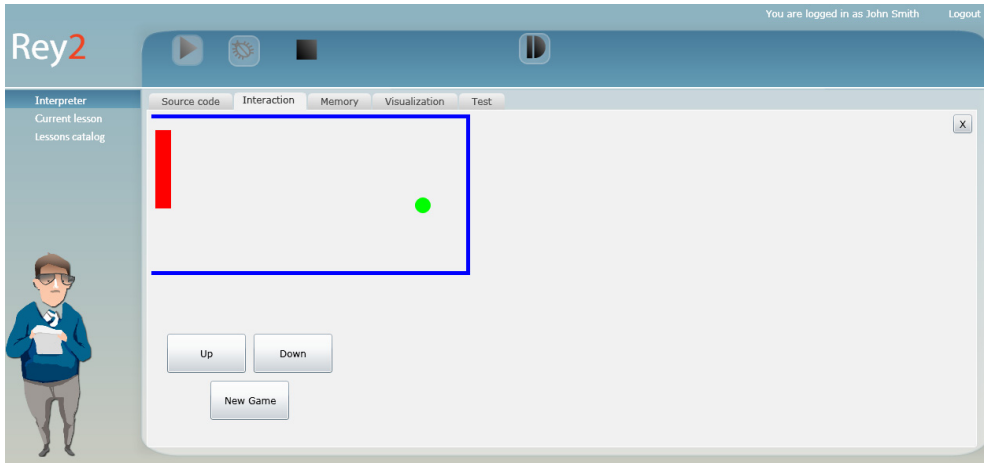

Fig. 4. Graphics interaction in the Rey2: simple PingPong game

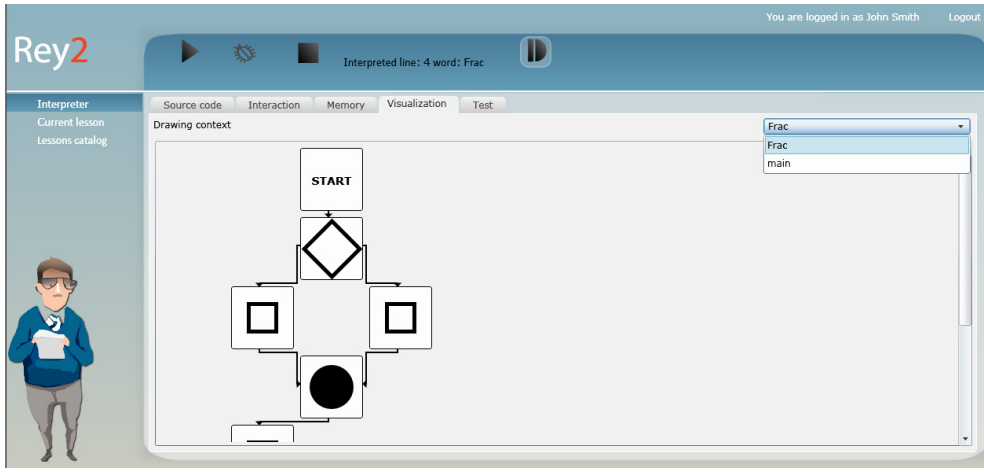

Fig. 5. Visualization of an algorithm in Rey2

4. SUMMARY

Common conduct of CS education using Rey environment might contain usage on very first classes with programming, and then any other classes that incorporate Rey and whose teacher supplies accurate libraries like for example multimedia. But Rey environment can be also used on Object Oriented Programming (OOP) classes, where students can learn more about concepts of OOP like abstraction (e.g. token object contains many information, not only the text content), encapsulation (e.g. usage of getters/setters), inheritance (e.g. creating new tokens

which inherit Rey system's tokens for building new instructions set), polymorphism (implementation abstract classes and interfaces – e.g. writing user own debugger system). As research of Ariel Ortiz on S-expression Interpreter Framework [11] shows, also teaching Programming Languages this way might give positive results. Students can learn OOP, design patterns, and programming language construction at the same time. Extensibility and design patterns play a major role in the Rey interpreter, which allows usage of Rey on multiple classes. One educational environment can cover the needs on many classes.

5. ACKNOWLEDGMENTS

I would like to thank Dr. Jakub Swacha, University of Szczecin for giving the idea of an educational programming language with a Polish keyword set. Also I am very grateful to Dr. Wojciech Sysło, dean of department of Information Technology Vocational College in Gorzów Wielkopolski (currently State School of Higher Vocational Education in Gorzów Wielkopolski) for all support given to the Rey project.

I am very thankful to Iwona Kalitan, student of Academy of Fine Arts in Poznań for creating graphics for the Rey2 system.

References

[1] BASZURO P., SWACHA J.: *Rey: An Educational Programming Language*, Informatics Education Contributing Across The Curriculum, Nicolaus Copernicus University, Toruń, Poland, 2008.

[2] MERNIK M., HEERING J., SLOANE A. M.: *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys, Volume 37 , Issue 4, ACM, New York, USA, 2005, Pages: 316 – 344.

[3] FOWLER M.: *Language Workbenches: The Killer-App for Domain Specific Languages?* [Online] June 2005. http://martinfowler.com/articles/languageWorkbench.html [access: 2008].

[4] AHO A.V., SETHI R. AND ULLMAN J. D.: *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Reading, MA, USA, 2001.

[5] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design Patterns*, Addison-Wesley, New York, USA, 1995.

[6] MCCONNEL S.: *Code Complete Second Edition, A practical handbook of software construction*, Microsoft Press, Redmond, USA, 2004.

[7] BEGEL A., SIMON B.: *Novice software developers, all over again, International Computing Education Research Workshop*, Proceeding of the fourth international workshop on Computing education research, ACM, Sydney, Australia, 2008, Pages 3-14.

[8] KÖLLING M., QUIG B., PATTERSON A., ROSENBERG J.: *The BlueJ system and its pedagogy*, of Computer Science Education, Special issue on Learning and Teaching Object Technology, Vol 13, No 4, Dec 2003.

[9] KÖLLING M., ROSENBERG J.: *An object-oriented program development environment for the first programming course*, Proceedings of 27th SIGCSE Technical Symposium on Computer Science Education, ACM, Philadelphia, Pennsylvania, 83-87, March 1996.

[10] PATTERSON A., KÖLLING M., ROSENBERG J.: *Introducing Unit Testing With BlueJ*, Proceedings of the 8th conference on Information Technology in Computer Science Education (ITiCSE 2003), Thessaloniki, 2003.

[11] ORTIZ A.: *Language Design and Implementation using Ruby and the Interpreter Pattern*, Technical Symposium on Computer Science Education, Proceedings of the 39th SIGCSE technical symposium on Computer science education, ACM, Portland, OR, USA, 2008, Pages 48-52.