Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Customizing a functional programming language for web development

Saša N. Malkov [*]

University of Belgrade, Faculty of Mathematics, Studentski Trg 16, 11000 Belgrade, Serbia

A R T I C L E I N F O

Article history:
Received 19 August 2008
Received in revised form
2 April 2010
Accepted 21 April 2010

Keywords:
Functional programming
Web development
Programming languages design
Wafl

A B S T R A C T

The characteristics of functional programming languages recommend them for web development. We checked that in practice, by customizing the functional programming language Wafl for web development. Customizations include support for web document types, web data access functions, textual templates, web dialogs and extension of transactional mechanisms to web session data. Prototype implementations of Wafl are developed. Our experience shows that application of Wafl in the web development domain is fully justified and that web development may benefit from functional programming languages in both education and production.

1. Introduction

The application of general domain programming languages in web development has some specific complexities. The complexity of language semantics reduces the efficiency of small teams of web developers, often built of web designers and HTML coders with moderate programming skills. It is often necessary to engage both programmers and web designers to achieve good results.

We analyzed the applicability of functional programming languages in web development. Different web development solutions and web development projects were analyzed. Many functional programming languages and approaches to web development were covered.

1.1. Web development tools

Different technologies and tools are used in web development. Their semantics and approaches to the problem vary widely. Some are based on general domain programming languages, while others are based on specifically designed environments and tools. Shklar [1] divides the technologies in four categories by their approach to web development. He recognizes programmatic approach, template approach, hybrid approach and frameworks. In this paper, we consider only the language design aspects of the problem. We recognize that languages are oriented towards four different approaches to web development: programmatic, template, hybrid and session approach.

The programmatic approach assumes that dynamic web content is constructed by programs, which are developed in a similar way as general domain programs. Both compiled and interpreted programming languages are used, including C, C++, Perl, Python and many others. Page construction programs use multiple string outputs or string concatenations to build

[*] Tel.: +381 11 2027 812; fax: +381 11 2630 151.
E-mail address: smalkov@matf.bg.ac.rs

1477-8424/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2010.04.001

web content as a string value. This approach produces large and weakly readable programs. That introduces unnecessary program complexity, even for relatively simple pages. Developers need to be well educated not only in HTML, XML and general programming topics, but also in web server architecture and basic web technologies like HTTP [2] and CGI [3].

The template approach is designed to simplify the development of pages with relatively simple dynamic elements. A page template describes a page in an extended HTML. The HTML extension includes some additional constructs for describing dynamic page elements. Static page elements are developed in pure HTML. Dynamic page elements are described using additional tags, including iterative tags, database query tags, tags for setting variable values and many others. The extensions are processed at the server side, during the page construction process. Some template-based solutions include a scripting language, primarily intended for additional customization of dynamic tags. A representative page templates solution is ColdFusion, which consists of ColdFusion Markup Language (CFML) and CFScript scripting language [4].

The template approach is based on the observation that, usually, only a small part of a dynamic page has to be dynamic, and most of the dynamic elements are often very simple. The principal quality of such solutions is their basic simplicity. However, with greater content complexity, the development complexity significantly increases.

The hybrid approach is a composition of the programmatic approach and the template approach. The basic principle is the same as for page templates—pages are described using an extended HTML. While template solutions use many additional markup tags, the hybrid solutions introduce only a few tags (often one or two) used to embed program segments that construct dynamic elements. The hybrid approach is the most commonly used one. Some of the significant representatives are PHP [5], JSP (Java Server Pages [6]) and ASP.NET [7].

It is widely accepted that the user driven sessions represent one of the web's strongest features. Nevertheless, the users' freedom results in a low level of application directed session flow control. The session approach is designed to provide a higher level of explicit session control. Representatives of the session approach are Mawl [8] and $<$bigwig$>$ [9]. Web sites implemented in Mawl consist of service logic (written in the programming language Mawl, which is similar to C++), and page templates (described in MHTML, which is an extended HTML). Additional MHTML tags are used for coupling of page templates and corresponding elements of service logic. A Mawl application is a service with one or more sessions. A session defines the data exchange between the user and application. The direction of session flow is fully determined by the service logic. Each user interaction has a single input point and a single output point. A weakness of programs in Mawl and $<$bigwig$>$ is the complexity of interfaces that have multiple input or output points.

1.2. Functional programming languages and web development

Functional programming (FP) languages are *declarative* languages. They perform the computation entirely by evaluation of expressions. Functional programs define *what* to evaluate, and not *how* to perform the evaluation [10–12]. Some of the most important FP languages are Lisp [13], ML [14] and Haskell [15]. It is often claimed that functional languages are not used in commercial applications. However, even though they are not used as often as imperative languages, many complex projects have been implemented successfully in FP languages [16].

Among the most common reasons given for the avoidance of FP languages in commercial software development are the questionable weakness of FP languages in the domains of dynamic systems and user interfaces and their restricted capacity to work with persistent data. The declarative semantics of communication mechanisms, including interactivity and file operations, may indeed become relatively complex to understand, to use, or both. On the other side, each persistent data update represents a side effect, which is strongly opposed to FP principles. Persistent data reading introduces a kind of non-determinism and prevents referential transparency. It is only possible partially to guard individual functional programs from these problems, by demanding the highest possible transaction isolation and delaying the realization of persistent data updates until the end of the program evaluation. However, the complete application may not be guarded.

Analyzing the properties of FP and web development, we concluded that the weaknesses of FP languages are not very significant in web development domain. On the other side, some FP features may deliver significant advantages:

- web content construction and general automatic document construction share an important property: in most cases, all necessary data are available at the beginning of the construction, and there is no need to explicitly communicate with a user. Web interactivity is either localized at client side or based on the concept of the session. The majority of web programs strictly construct some dynamic content, without any interactivity.
- Web applications are very often based on intensive usage of persistent data. However, there is a significant disproportion between read and update operations. Web content construction is usually based on persistent data, and most of the programs do read some data. On the other side, only a minority of the programs update persistent data. This provides some room for balancing.
- One FP feature is the ability for implicit parallelization of pure functional expressions. The first level of parallelization is that many programs may run in parallel. The second level, which is particularly well addressed by FP, is execution of different parts of the same program in parallel. With some care, implicit parallel evaluation of expressions with read-only database access may also be considered.

Many web development technologies based on FP languages have been introduced in recent years. Most of them are based on either a programmatic or template approach.

Haskell Server Pages and WASH/CGI are two of the solutions based on Haskell. Haskell Server Pages employ the template approach. HTML fragments are treated as ordinary Haskell expressions. Code fragments are escaped using additional tags. HSP supports XML pattern matching [17]. On the other hand, WASH/CGI is a Haskell library for web development using a programmatic approach. It is implemented in Haskell in a purely functional manner, using monads to support session states [18]. One significant achievement is the development of a web server in Concurrent Haskell. It resulted in an acceptable performance [19].

SMLserver is a web server platform based on Standard ML. It features strong typing, usage of quotations for HTML embedding and a kind of dialog mechanism [20]. It represents a relatively complete web development solution.

Lisp Abstracted Markup Language (LAML) abstracts major markup languages using a set of Scheme functions [21]. Dynamic documents are written as Scheme programs. The main achievement is strong typing of represented XML (or HTML). However, the LAML document itself (i.e. source program) is not typed, because of weak typing in Scheme.

The programming language Curl is defined as a platform for both programming and content presentation [22]. It is an object oriented language, with many features designed after C++ and Java. At the same time, it features many common FP techniques, including automatic garbage collection, higher order functions, static type checking and many others.

The PLT Scheme Web Server is based on the programmatic approach, but features strong support for session control [23]. Its implementation uses continuations to support send/suspend and send/suspend/dispatch execution models.

2. Programming language Wafl

Wafl is a functional programming language customized for web development [24]. Wafl supports all different approaches to web development. In the following sections, we present both the general language properties and web customizations. Because of the scope of the paper, the general properties are covered briefly.

2.1. General properties

Wafl is generally designed to be domain independent and to provide general applicability, with many conceptual similarities to contemporary functional programming languages, like Haskell and ML. Wafl programs are syntactically equivalent to expressions. The basic expression syntax is similar to ML and C/C++.

Wafl is a strongly typed language with implicit static type checking. The type system is based on generic polymorphism, with no explicit type specifications in source code. Each defined construction can be used in any context and with arguments of any type for which its definition represents a valid expression. Static type checking with automatic type inference is entitled to infer and validate all expression types before the program evaluation. Higher order functions and partial function application are fully supported. Partial function application in Wafl is not limited by an argument order.

In addition to function application syntax with parenthesis and comma-separated arguments, Wafl introduces new "arrow syntax". Arrow syntax enhances the readability of multiple function application nesting, which is very frequent in FP programs. It is similar to method usage syntax in OO languages: a function can be applied as a "method" of its first argument, with other arguments specified in parenthesis:

$f(a_1, a_2, \ldots, a_n) = a_1\text{->}\, f(a_2, \ldots, a_n)$

Wafl syntax defines an interface to persistent data. The interface is directed primarily to relational databases. It is based on the SQL standard [25]. Read-only access to persistent data is provided by "query" functions. Persistent data update is handled by actions and transactions. Although actions and transactions represent general language features, they are presented in the following sections, because of their significant role in web development.

Side effects in Wafl are localized in special transactional functions. While this approach does not eliminate the side effects, the clear separation of transactional code segments from pure functional code segments allows the easier recognition, validation, verification and implicit parallel evaluation of pure functional program segments. As a consequence, Wafl partially supports pure functional programming in programs that use persistent data. In addition, application of the highest transaction isolation level may prevent non-determinism in single transactions, or even in single programs.

A more comprehensive review of the general properties of the Wafl programming language is available on WWW [26].

2.2. Wafl web customization

Wafl is customized to support each of the web development approaches. This support is attained by introduction of MIME resources, web data access functions, textual templates and dialogs. Furthermore, Wafl transactions are extended to cover web session data. All web customizations are fully orthogonal to general Wafl features and pure functional programming.

A Wafl web application represents *a service*. A service consists of many page constructing programs and corresponding libraries. The service user is called *a visitor*. Visitors can explicitly enter data into visual forms, or implicitly send it using links. In either case, the data sent to a service by single *request* represent *a form*. The targeted web server and its Wafl processing module map each *request* to an appropriate Wafl program. The program evaluates a HTML page (or another resource), which is sent to the visitor as *a response*. A request–response pair represents a *visit*. A set of all visits to a single service by one visitor, in a preconfigured time interval, represents a *session*. Sessions are usually structured as ordered sequences of visits; however, if a visitor opens links in separate browser windows, the session can be structured as a tree of visits.

2.2.1. *Constructing web content*

Web content is usually specified using MIME types [27]. Wafl supports MIME types by the `MimeResource` data type. A `MimeResource` value consists of *resource type name* and *resource content*. The function `MimeResource(t,c)` evaluates a `MimeResource` value of type `t` and content `c`.

Each Wafl program, belonging to a web service, should evaluate a `MimeResource`. To provide developers with some flexibility, Wafl supports implicit conversion of program results to string type, and then to a `MimeResource` of default type (usually "text/html").

2.2.2. *Web data access functions*

Three new data collections are accessible in Wafl web programs: service data, session data and form data. Each of these data collections is implemented as a map of string keys to string values.

Service data are read-only, shared among all visits in all sessions for a service. These data represent service configuration parameters. The configuration setting is handled by service administrators using service configuration files, which is not covered here. The functions `Service()` and `ServiceValue(x)`, respectively, evaluate a map of service parameters and a value of service parameter with key `x`.

Session data represent the session state. Session data are shared among all visits in a single session. The functions `Session()` and `SessionValue(x)` read session data. Transaction functions may update session data, which is covered in the following sections.

Form data are read-only, embedded in a single visitor request. The functions `Form()` and `FormValue(x)` read form data.

Other request data are evaluated by additional functions: `httpHost()` (the full name of the host running the service), `httpScript()` (the name of the currently evaluating program and its logical path), `httpPathInfo()` (the path information of HTTP request) and others.

2.2.3. *Textual templates*

The construction of HTML and XML web content is based on multiple string concatenations. Some of these strings are literals, and others are constructed during the program execution. *Textual template* is a simplified syntactical construction for multiple string concatenations. Its base purpose is to allow developers to easily integrate dynamic elements in designed static HTML or XML content.

The textual template syntax is based on tags, and thus customized to HTML and XML. A template begins with keywords `html template` and ends with termination tag `<#>`. The template body consists of static HTML or XML text and expression tags. Expression tags have the form: `<# exp #>`, where `exp` is any Wafl expression of string type. Templates evaluate strings, which are not considered to be template definitions. Textual templates are semantically equivalent to concatenations of static template blocks and the results of embedded expressions. For example, the following functions are equivalent to each other:

```
greeting1( name)=htmltemplate
   Hello <# name-> strUpperCase()#> ! <br/>
   <#> ;
greeting2( name ) = "Hello " + strUpperCase(name) +"! <br/> ";
```

In addition to basic Wafl modules (programs, function modules and libraries), page template modules are introduced. A page template module is a program module, which has a textual template as a body. The termination tag is not required at the end of the page module.

2.2.4. *Dialogs*

Session flow is usually, implicitly directed by the visitor. However, sometimes a service must communicate with visitors in a strict order to construct complete responses. A session segment driven by service represents *a dialog*. A dialog consists of an ordered sequence of *driven interactions*. While usual visits consist of visitor's requests followed by the service's responses, a driven interaction consists of a service's *question* followed by a visitor's *answer*.

The dialogs represent segments of the response evaluation process. Each driven interaction is realized by the `ask` function. It has a single argument—a string representing a HTML page, which plays the question role. The function sends the given question to the visitor as an intermediate response, thus initiating a driven interaction and *asking* the visitor for an answer. The evaluation of `ask` is then temporarily suspended. The suspension lasts either until an answer is received, or until a timeout period expires. (The timeout period is configured on service level.) When the visitor submits a form or clicks a link contained in the question, the corresponding HTTP request is transferred to the suspended program as *an answer*. The suspended program resumes and the suspended function `ask` returns the answer as a result. The answer is returned as a map with string keys and string values, in the same form as regular requests data are returned by the function `Form()`. After the dialog completion, the answer is further processed and the program computes *a final response* and sends it to the visitor.

A single program may contain multiple usages of the ask function. There are no limits on dialog length or complexity. Questions may form a sequential dialog, or be embedded in other question evaluations.

From a visitor's point, the question is no different from any other web page. However, a page representing a question contains specifically encoded links and form actions. These encoded links and actions contain additional information, which is used to inform the service that the appropriate visitor's action represents an answer to a specific question, instead of an ordinary request. Each link and form action URI in a question page has to be encoded by the `answerAction` function. It has no arguments and evaluates a string that uniquely identifies the currently evaluated request and the evaluation phase. Two evaluations of the `answerAction` during a single response evaluation, not separated in time by evaluation of the `ask` function, return the same value. In any other case, two evaluations of `answerAction` return different values.

The following example asks a visitor for his/her name, and evaluates a corresponding greeting page as the final response:

```
question()                  // same as: greeting(ask(question()))
  -> ask()
  -> greeting()
where {
  question() = html template
    < html >< body > Hello, User!
    < form action = " <# answerAction() #> " method = "post">
     Please enter your name:
      < input type = "text" name = "name" />
      < input type = "submit" value = "OK" />
    </form > </body > </html > <#> ;
  greeting(answer) = html template
    < html > < body > Instead of "Hello, User!", now we can say
    < b > Hello, <# answer["name"] #> ! </b >
    </body > </html > <#> ;
}
```

The semantics of Wafl dialogs may be defined and implemented based on the continuation-passing style [28]. The continuation semantics is fully enclosed in the function `ask`. It is similar to send/suspend model of PLT Scheme Web server [23]. The dispatcher logic for more complex send/suspend/dispatch model is not embedded in Wafl definition. However, it is relatively easy to implement and use it in program code, based on analysis of results returned by `ask`.

The limitations of the presented dialog subsystem are not significantly different from the limitations of other session controlling solutions: it is not possible to freely browse back to pages that represent dialog questions. It is necessary to restart the dialog from the beginning, instead.

2.2.5. *Transactions*

Transactions are one of the basic concepts in databases [29]. Wafl extends transactions to cover not only database data, but also web session data. Transaction handling in Wafl is realized by *actions* and *transactional functions*.

2.2.5.1. Actions. Actions are special Wafl functions that may perform side effects. Actions may not be used as freely as other Wafl functions. Each usage of an action has to be (directly or indirectly) enclosed in a transaction.

An action body is a sequence of logical expressions. The expressions may include any Wafl expression of `bool` type, Wafl statements and SQL statements. Wafl statements and SQL statements are exclusive sources of side effects in Wafl. They may appear in programs exclusively as atomic elements of action bodies. Thus, all side effects are localized in actions.

Action syntax is customized to its imperative behavior. The semantics is defined as a conjunction of corresponding logical expressions. An action is evaluated by evaluating the contained expressions in the order of specification until an expression evaluates "false", or all expressions are evaluated. The result is "true" if all expressions evaluate "true"; or "false" otherwise.

SQL statements are executed against a database. They evaluate "true" if the execution is successful or "false" otherwise. If a read-only statement is executed, it returns "true" if the execution is successful and the result is non-empty; or "false" otherwise. Read-only statements in actions and transactions are used only to check if some condition stands. Query functions are used to read data from the database, which is not covered here.

Wafl statements perform side effects against web session data. Successful statement execution evaluates as "true". There are two Wafl statements: `set` and `reset`. The statement `set` updates a session value for the given key, or adds a new value if there is no session value with the given key. The statement `reset` removes a session value with the given key.

2.2.5.2. Transactional functions. Transactional functions are a kind of action. A transactional function evaluation is wrapped by transactional behavior. The transactional function evaluation consists of a new transaction *initiation*, the *transaction body evaluation* and the transaction *completion*.

The transaction body evaluation is the same as in the case of the actions.

The transactional function initiation opens a new transaction. If a transactional function evaluation is enclosed in an another transactional function evaluation (either directly on indirectly), then only the evaluation of the most outer transactional function incorporates the transactional behavior, and includes both the initiation and the completion. The inner transactional functions evaluate in exactly the same way as actions, without initiation and completion.

Transactions are completed by commit or rollback. If the transaction body evaluates "true", the transaction commits and evaluates "true". Otherwise, the transaction is rolled back and the result is "false". The transactional behavior includes both the database data and the session data, as a single complex transaction environment.

The usage of actions and transactional functions is illustrated by the following example, which updates the citation counts for papers in 2008, and sets session value `updated` to "`All`":

```
papers( 2008 )-> updateCitations()
where {
    papers( year ) = typed query {
        select paperid from paper
        where year = :year
        };
    updateCitations( paperNoList ) = transaction {
        paperNoList-> forall(
            \r: updateSingleCitation(r.paperid)
            );
    set updated = 'All';
    };
   updateSinglePaperCitation( paperid ) = action {
    update paper
    set cit_count = ( select count(*) from citation
                    where citedpaper = :paperid)
    where paperid = :paperid;
    };
};
```

3. Discussion and conclusion

The functional programming language Wafl is customized for web development. The customizations cover different aspects of web development and support each of the known web development approaches: programmatic, template, hybrid and session approach. The programmatic approach is supported by the general properties of the language. Textual templates allow template-based web development. Together, templates and general programming features fully support the hybrid approach. Support for session control is introduced with Wafl dialogs.

The web customization of Wafl is designed to be orthogonal to general language features, including language capabilities for pure functional development. The Web contributions introduced in this paper do not reduce the general applicability of the language, but extend its functionality and usability in the specific domain.

Web development and automatic content construction are very often strongly dependent on persistent data. Wafl features a simple database query interface, using embedded SQL for database access. The standardized database interface prevents future incompatibilities in interfaces to different database systems.

The transaction subsystem supports transactions on a unified data store, consisting of both database and internal session data. Data sharing is implicit. The environment itself (i.e. the language implementation) takes care of data sharing and concurrency problems. Side effects are fully localized in transactional functions, which are clearly syntactically emphasized. Principles of pure FP are applicable for code segments that neither contain nor use transactions.

Prototype implementations of the Wafl programming language have been developed for Apache (for both Linux and Windows) and Microsoft IIS web servers. A command line version and a standalone Wafl web server have also been developed. Many web sites have been developed using Wafl. While some of them are designed primarily for testing purposes, some complex web applications have been developed on independent projects.

Our experience shows that web development may benefit from the FP paradigm in many different ways. Important correlations of properties of FP languages and web development emphasize the qualities of the FP paradigm and, at the same time, marginalize the most important weaknesses. The benefits are notable in both education and production.

Among the benefits, we emphasize the simplicity of the development of the web content and the relatively steep learning curve. Wafl was used with several generations of students as one of the tools for practical work in functional programming and web development subjects. After only few introductory lectures, students were capable to develop web applications in Wafl. An interesting example developed by students is an online interactive book, Analytical Geometry [30].

Our current and future plans include improvements to existing Wafl features and design and introduction of some new customizations. Possible improvements in the general language concepts and syntax will also be analyzed. Among the research directions are analysis of prospects for multi-tier development and SOA, and type checking of resulting HTML/XML code. The prototype implementation improvements will primarily target certain advanced Wafl features and performance optimization.

References

[1] Shklar L, Rosen R. Web application architecture—principles protocols and practices. John Willey and Sons; 2003.

[2] Fielding R, Gettys J, Mogul J, Frystyk H, Berners Lee T, Hypertext transfer protocol—HTTP/1.1, *world wide web consortium,* 1997. ⟨http://www.w3.org/Protocols/rfc2068/rfc2068⟩.

[3] ISO/IEC 9636:1991 Information technology – computer graphics – interfacing techniques for dialogues with graphical devices (CGI)—functional specification, ISO/IEC, 1999.

[4] Developing Web Applications with ColdFusion, Allaire Corporation, 1999.

[5] Stig Sather Bakken, et al. PHP manual. PHP Documentation Group; 2000.

[6] JavaServer Pages Specification, Version 2.0, Sun Microsystems, 2003.

[7] Dino Esposito. Introducing microsoft ASP.NET 2.0. Microsoft Press; 2004.

[8] Mawl 2.1 Tutorial. Mawl software distribution. Bell Laboratories, Lucent Technologies; 1998.

[9] Brabrand Claus, Moller Anders, Schwartzbach Michael I. The <bigwig> project. ACM Trans Internet Technol 2002;2(2):79–114.

[10] Hudak Paul. Conception, evolution, and application of functional programming languages. ACM Comput Surv 1989;21(No. 3).

[11] Hughes John. Why functional programming matters. In: Turner uD, editor. Research topics in functional programming. Addison Wesley; 1990.

[12] Henderson P. Functional programming: application and implementation. Prentice Hall International; 1980.

[13] McCarthy J. History of Lisp, preprints of proceedings of ACM SIGPLAN history of programming languages conference. SIGPLAN Not 1978;13.

[14] Milner R, Tofte M, Harper R, MacQueen David. The definition of standard ML—revised. 0-262-63181-4.

[15] Paul Hudak, Simon Peyton Jones, Philip Wadler, editors. Report on the programming language Haskell, ACM SIGPLAN Notices 1992; vol. 27 (No. 5).

[16] Wadler Philip. An angry half-dozen. ACM SIGPLAN Not 1998;33(2):2530.

[17] Meijer E, van Velzen D, Haskell server pages—functional programming and the battle for the middle tier, in electronic notes in theoretical computer science 41(1), Elsevier Science B.V., 2001.

[18] Thiemann Peter. WASH/CGI: server-side web scripting with sessions and typed, compositional forms. PADL 2002 2002:192–208.

[19] Marlow S, Writing high-performance server applications in Haskell, In: Hutton G, editor, Haskell Workshop, 2000.

[20] Elsman Martin, Hallenberg Niels. Web programming with SMLserver. PADL 2003 2003:74–91.

[21] Nørmark Kurt. Web programming in scheme with LAML. J Funct Program 2005;15(1):53–65.

[22] Ward S, Hostetter M. Curl: a language for web content. Int J Web Eng Technol 2003;1(No. 1):41–62.

[23] Krishnamurthi S, Hopkins PW, McCarthy J, Graunke PT, Pettyjohn G, Felleisen M, Implementation and use of the PLT scheme web server, higher-order and symbolic computation (2007).

[24] Malkov S, WAFL—functional programming language for development of web applications, Master Thesis, University of Belgrade, Faculty of Mathematics 2002.

[25] ISO/IEC 9075:1999 Information technology—database language SQL, ANSI/ISO/IEC 1999.

[26] Malkov S, Wafl Tutorial, available at: ⟨http://codd.matf.bg.ac.rs/wafl⟩.

[27] Freed N, Borenstein N, Multipurpose internet mail extensions (MIME) part two: media types, Technical Report 2046, Draft Standard, World Wide Web Consortium 1996.

[28] Appel AW. Compiling with continuations. Cambridge University Press; 1992.

[29] Date CJ. An introduction to database systems, 8th ed. Addison-Wesley; 2004.

[30] Mitić N et al., Analytical geometry, (online book in Serbian; English translation is in progress. Available at ⟨http://codd.matf.bg.ac.rs/angeom/pocetak.wafl⟩.

Saša Malkov has PhD in Computer Science from University of Belgrade. Fields of research include functional programming, relational databases, software engineering and bioinformatics. Professional experiences include software development on different positions, from programming to information systems modeling and development management.