Component-based Engineering of Web User Interface Designs for Evolutionary Optimization

Maxim Bakaev
Automated Control Systems department
Novosibirsk State Technical University
Novosibirsk, Russia
bakaev@corp.nstu.ru

Vladimir Khvorostov
Automated Control Systems department
Novosibirsk State Technical University
Novosibirsk, Russia
xvorostov@corp.nstu.ru

***Abstract*—Component-based approach has proved its effectiveness in modern web engineering, where programmers increasingly rely on web development frameworks. In front-end web development parts of available solutions cannot be reused directly, due to technical and legal obstacles, so we propose a specific process to generate web user interfaces (WUI) designs from configurable components. The similarity between the generated designs and the exemplary high-quality solutions retrieved from case base is then optimized within the evolutionary algorithm. In our current paper we designate the structure of the components and justify the employment of Drupal as the organizational framework. We also describe the implementation details, including the two supplemental software tools that we developed: 1) web intelligence miner that collects website-related data from linked open data sources and 2) web UI screenshot analyzer that converts it into semantic-spatial representation in JSON format. Finally, we specify the new solutions generation algorithm, per the three WUI dimensions: functionality, layout and visual appearance.**

Keywords—software components, user interface design, web engineering, case-based reasoning, evolutionary algorithms

I. INTRODUCTION

Reuse of existing code and solutions, which started as early as in the 1950s, is now widely named as the top productivity-boosting technique in software engineering [1]. Conventional websites are nowadays rarely created from scratch, but are built using web design (front-end) and web development frameworks that aim to automate the common activities involved in the process. Case-based reasoning (CBR) approach, which is well-developed in AI theory and successfully applied in practical expert systems, can be a solid formal base for facilitating the web user interface designs reuse support. CBR implies the following stages, classically identified as Retrieve, Reuse, Revise, and Retain [2]:

- describing a new problem and finding similar problems with known solutions in a case base (CB);
- adapting the solutions of the retrieved problems to the current problem, in an assumption that similar problems have similar solutions;
- evaluating the new solution and possibly repeating the previous stages;
- storing the results as a new case.

The reported study was funded by RFBR according to the research project No. 16-37-60060 mol_a_dk.

Currently, there are reports of successful CBR use in software engineering [3], web services composition [4], web interaction personalization [5], and so on. However, in web design domain, there are technical (lack of access to the website's back-office and server-side code) and legal (copyright protection) impediments to adapting new solution from an existing one, as the "classical" CBR would prescribe. To the extent of our knowledge, there is lack of established approaches for generating new web user interface (WUI) designs similar to the retrieved (model) solutions, which is a major obstacle for wider CBR application in the domain [6]. Web design frameworks or design mining systems [7] provide rich libraries of interface elements (icons, buttons, form fields, etc.), but they generally lack capabilities to generate or even recommend designs appropriate to a particular design problem and to aid in their refinement.

We would argue that web UI generation from individual elements can hardly be effective, since 1) they can play too many roles in different contexts of use and 2) the resulting layout and interaction workflow are very sensitive to even minor shortcomings and "99% close to optimal" can still be too far in terms of being comprehensive to the users. Thus, we suggest relying on the intermediate reusable and configurable blocks of logically interrelated elements solving some common tasks in a specific context. Although such concept is utilized under somehow different names in different areas of Software Engineering, in our work we are going to use the term "component". Akin reusable sub-routines date from as early as the 1950s (IBM's Share library with mostly math functions), but even much later reuse of design remains rather situational [1]. Reported obstacles to its wider applicability include domain-dependence, increased effort involved in development of reusable solutions (about 3 times compared to specialized one), and issues in structuring and self-organization [8]. For resolving the latter, several components-related standards and specifications were introduced in Web Engineering, such as Custom Elements, Shadow DOM, templates and HTML-imports, etc. The numerous examples of de-facto components repositories involve such technologies as JavaBeans, EJB, CORBA, ActiveX, DCOM, .Net – as organization and population of the components base is crucial in facilitating their application.

In our paper we outline the approach for engineering new website designs from components, while referring them to existing cases, which promises significant boost in development

978-1-5386-5889-5/18/$31.00 ©2018 IEEE
SNPD 2018, June 27-29, 2018, Busan, Korea

IEEE computer society

of conventional websites. Since immediate adaptation of retained solutions is rather problematic in the considered domain, we propose to generate new designs and only then check if they are similar to the relevant and high-quality solutions (based on the CBR assumption "similar problems have similar solutions"). So, in the generation we need to vary parameters of the new designs to optimize the goal function measuring similarity between the new and the exemplary solutions, per the identified set of influential features. An effective iterative approach for such generation can be based on evolutionary algorithms (EAs), as we described in [9], where candidate solutions are composed from available configurable components, with cross-overs and certain probability of mutations.

This paper is built upon several our previously published works (we provide references where appropriate) and is dedicated to the actual generation process and its facilitation by some related software that we developed. The structure of the paper is the following. In Section 2, we designate the structure of the components and the requirements towards their organizational framework. We also outline the features for cases in web design domain and describe the two supplemental software tools that we developed. In Section 3, we present some implementation details: the web UI knowledgebase portal, including the components' repository, and the new solutions generation algorithm, as part of the evolutionary optimization process. In Conclusions, we summarize our findings and outline prospects for further work.

II. Component-Based Approach to Web User Interface Designs Engineering

A. Interaction Design Components and Frameworks

Although widely used in practice, the concept of component still has certain ambiguity, as sometimes "common sense" understanding of the term gets in the way. A rather classical definition for component w.r.t. architecture is "abstract element in the system's structure, explicitly distinguished in the environment, solving certain sub-tasks within the system's goals, and interacting with the environment through interface(s)". For component as a structural unit, the interfaces specifications (indeed, there can be several of them, since one component can play several roles in the system) have to be very rigorous, delimiting all the component's potential interactions with the environment. The primary points of such "interface contract" are:

- Constraints for the input data (pre-condition);
- Properties of the outputs/results (post-conditions);
- Dependencies with other components.

Without alternative, the components work within a framework – platform that defines the structure of the system and facilitates its development from the components (e.g. Zend Framework, Django, .NET, etc.). The framework user's programming code is infused into the framework code and is managed by the latter (unlike in libraries, where the situation is the opposite). Another important function of framework is ensuring consistency of the system's end user interface, particularly through using design patterns. Some popular categories of interaction design (ID) patterns – established solutions for UI usability and accessibility – are data input through web forms, navigation hierarchy and menus, etc. The main points in an ID pattern structure are the following [10]:

- Problem and applicability – a situation (in terms of the tasks, the users and the context of use) that the pattern is relevant for;
- Solution – a proven solution to the problem, usually allowing freedom of implementation (configuration of certain parameters);
- Justification and examples – ergonomic principles a pattern is based on, what particular interaction quality attributes could be enhanced, rationale why and how the pattern works to solve the problem, and how the pattern has been successfully applied in a real life system.

Obviously, this structure quite resembles the case structure in the case-based approach. Further we'd like to note that the relationship between the scale of problems in ID patterns and in web design cases implies that ID patterns correspond to components – the abstract (configurable) structural elements performing sub-tasks within the whole system. Indeed, web design-solutions are assembled from ID patterns-solutions, but this is by and large done by human designers, unlike in more advanced fields (e.g. Web Engineering with its auto-organizing components [8]). Today's pattern management tools do make some use of relationships and semantics [11], but for the ID patterns to truly allow self-organization, they must possess components' features and be organized in a corresponding framework.

To construct these ID components, let us perform a sort of semantic union operation on ID patterns {Problem, Solution, Justification} and components {Pre-conditions, Post-conditions, Dependencies}. Firstly, it would mean that users, tasks and context of use must be specified rather formally, as the input data in a component's "interface contract". Second, there must be full specification of different configurations for solutions produced from the pre-conditions. Third, ID components need to contain estimations of quality attributes for various pre-conditions, possibly with justifications and references to the underlying principles. Finally, the dependencies with other components must embrace both technical compatibility and configuration compatibility, to keep the resulting UI consistent (e.g. prevent different interface elements from having different visual styles). The above considerations transform into the following requirements towards a framework:

- Allow custom attributes for components (functional applicability, interaction-related pre-conditions, quality attributes). Support procedures for obtaining the attributes' values. Allow search and filtering through the components.
- Maintain the organized components' repository supporting their reuse. Allow updates for the components, control compatibility and dependencies.

- Provide capabilities for rapid programmable composition of new solutions with support of genetic operators (cross-over, mutation).
- Support easy integration with external software tools: to populate the case base, obtain the feature values for the cases, assess the similarity between the solutions, etc.

B. *WUI Design Features and the Supplemental Software Tools*

As there seems to be no agreed structure of features in the web design domain, we founded ourselves upon the model-based approach to web UI development, which generally identifies three groups of models: (1) per se interface models – Abstract UI, Concrete UI, and Final UI, (2) functionality-oriented models – Tasks and Domain, and (3) context of use models – User, Platform, and Environment. Of these, we consider the Domain, Tasks, and User of higher relevance to web design reuse, while Platform and Environment models rather relate to website's back-office. Also, not all existing website designs are equally good (in contrast to e.g. re-usable programming code), so quality aspects must be reflected in the feature set. More detailed description of our informal feature engineering for solutions' retrieval and similarity assessment can be found in [6].

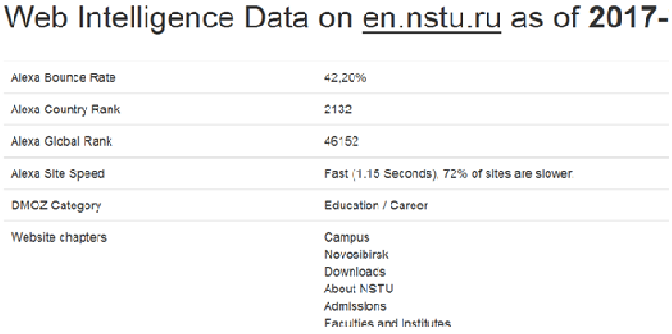
In most cases collecting websites from the Web (HTML, CSS and JS code, or even screenshots of their visual appearance in browser) involve little difficulties with the modern web scraping technologies. However, to properly organize the CB, the structured data describing both problems and solutions must be somehow obtained – preferably on a regular basis, as thousands of new web projects are launched every day, while the relative quality of existing websites can fluctuate. Based on the features we identified, it implies attaining the knowledge about the target users, their tasks, the platform, etc., as well as about the performance of the solution respective to the project goals. Obviously, the corresponding specifications and performance indicators are virtually never made openly available, so reverse engineering (RE) techniques have to be applied to existing websites.

RE in software engineering implies moving "backwards" in the development cycle – from program code to design or requirements. The expected outcome of RE very much depends of the goal, and common objectives are [12]: 1) understanding an existing product – especially, in software maintenance, 2) re-using effective and verified design solutions – sometimes the ones made by competitors, 3) facilitating the "forward engineering" process when migrating software to a different platform or re-designing a website. To support software source code RE, a wide range of techniques and tools exist, including disassemblers and decompilers, browsers for code analysis and annotation, and of course numerous frameworks performing (with varying degree of success) automation of the process – e.g., generating UML diagrams as the output. With the increasing recognition of the model-based web UI development paradigm, inferring design models from existing code became an attractive prospect, as explicitly specifying model for a complex interaction may be harder than actually coding an interface. Tasks extraction from website code is arguably the most common focus for existing web RE tools (a quite successful example being e.g. [12]), while user characteristics can be inferred from access/interaction logs [13]. To facilitate the CBR- and components-based composition of WUI designs, we developed two supplementary software tools:

1. "web intelligence miner", collecting and storing website quality and meta-data from linked open data sources (see details in [14]),
2. website screenshot analyzer, capable of transforming the web page visual representation into semantic-spatial representation of the web UI in JSON format and calculation of several quality-related metrics (see details in [15]).

Web Intelligence miner. The mining of website code, in the absence of access to the website specifications and use statistics, can be supplemented with extraction and analysis of website-related data from external sources (Linked Open Data). Virtually any operational website is regularly explored by crawlers, robots, spiders, etc. of numerous global web services, which can provide reasonably good estimations of the website performance, visitors, and of some other features' values. Openly accessible web catalogues, search/indexing systems, code and design validators can supply information on the website domain, location, target audience, platform, and so on. So, previously we proposed to apply the term Web Intelligence (WI) to the process of website-related data gathering from LOD sources and their accuracy cross-checking [14]. Our WI miner software is capable of extracting data related to the webpage at the specified URI (see also Fig. 1):

- **Domain:** website category names (collected from DMOZ catalogue and SimilarWeb service).
- **Tasks:** website chapter names (extracted by the WI miner from the webpage code).
- **Users' demographics:** gender, education (collected from Alexa.com service).
- **Quality:** (1) technical quality: Flesch-Kincaid Grade Level (from https://readability-score.com), page load speed (Alexa), number of errors and warnings (https://validator.w3.org), and (2) quality-in-use: number of visits popularity ranks (Alexa and SimilarWeb), bounce rates (SimilarWeb), etc.


Web Intelligence Data on en.nstu.ru as of 2017-

Alexa Bounce Rate	42,20%
Alexa Country Rank	2132
Alexa Global Rank	46152
Alexa Site Speed	Fast (1.15 Seconds). 72% of sites are slower
DMOZ Category	Education / Career
Website chapters	Campus Novosibirsk Downloads About NSTU Admissions Faculties and Institutes

Fig. 1. The example of the WI miner output, with auto-extracted website chapters.

Screenshot Analyzer. This software makes screenshot of the webpage at the specified URI or takes an existing screenshot (e.g. of a generated solution for a new problem) and outputs semantic-spatial representation of the web UI in JSON format (see Fig. 2). From the JSON representation, it also calculates several quality metrics, currently the ones related to visual complexity of the page and the ratios of areas (e.g. whitespace to text). In its work, the analyzer employs specially developed algorithm for detecting rectangular shapes, which uses OpenCV and Dlib libraries, and performs text recognition based on Tesseract. We plan to supplement the analyzer with the solution's code mining and reverse engineering ID patterns from it, so that the EA trying to generate the solution similar to the ones retrieved from the CB could "know parts of the answer" and thus achieve better convergence.

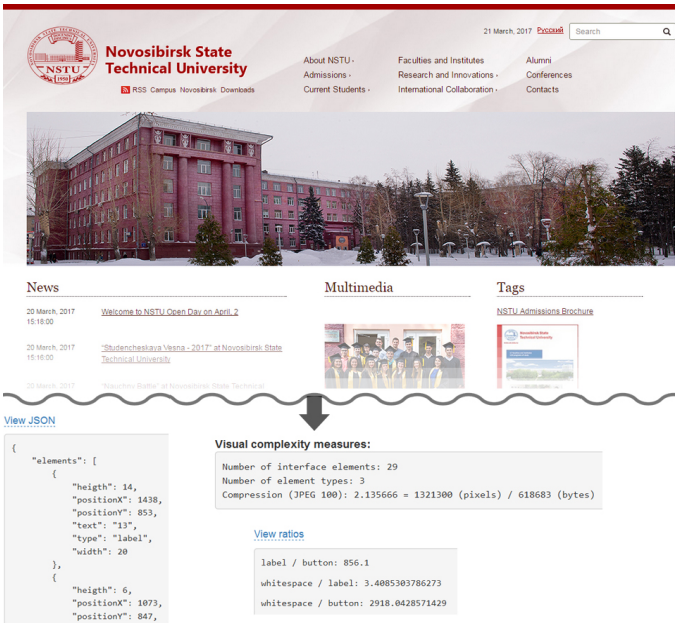


Fig. 2. Illustration of the screenshot analyzer input and outputs.

III. Implementation of the Web User Interface Designs Generation Algorithm

In this section we present some highlights of the proposed approach that we implemented (some of the prototype tools are available at our dedicated portal http://wuikb.tech). We used Drupal web content management framework as the platform for the implementation, motivated by the following:

- Drupal is known for its robust architecture that allows handling high number of components in the website.
- Drupal has lots of components (modules) ready for reuse, they are well-organized and centralized in the single repository with API access (http://drupal.org), they have quality auto-maintained quality attributes (# of downloads and actual installs, # of open bugs, etc.).
- Drupal has programmable (via command line, API, etc.) support for installment of websites, the layout of interface elements on webpage, handling web forms, menus, content items, adjustment of visual appearance styles, and so on.

A. The Case Base and Specification of Existing Solutions

In the portal, the case base is repository of projects each of which correspond to a case and can be either automatically scrapped from the web or specially created. Further we describe the constituents of Project in the CB.

Problem. The problem description are items that are either directly specified when a new project is created, mined by the CB populating tools, or provided by human annotators. The detailed structure of the Problem is the following.

Specification of the Tasks model. Tasks model for a problem is represented as the "cloud" of keywords that are website chapters. For existing projects they are automatically extracted from the HTML code (labels in the navigation) of the solutions by our WI miner (Fig. 1). Domain-specific ontology or a lexical database such as WordNet can be used as controlled vocabulary with the standard chapter names.

Specification of the User model. The user model is compliant with WebML and FOAF specifications. As estimations by the automation tools were so far found inaccurate, we have to rely on human annotators in assessing the user attributes for the target user group: gender, age, education and income levels, etc.

Meta information. Extra information that characterizes the project can be automatically derived from the problem description or annotated and includes the following.

Domain. For a new project domain can be selected from the pre-defined values that are based on DMOZ categories: Business, News/Media, Shopping, Society, etc. For existing solutions, domain is mined from a DMOZ mirror (see Fig. 1). Similarity can be calculated automatically, as DMOZ allows downloading the hierarchical structure of its categories in RDF format.

Textual description. The text can be mined from web catalogues and meta-tags in the solution's homepage.

Context tags. The context of the project can be described with the cloud of weighted tags, available from domain ontologies.

Design guidelines. The list of guidelines and design patterns that are relevant to the project, with weights denoting the degree of applicability in the solutions.

Solutions. There can be two types of solutions in the case:

1. Prototypes, uploaded to the portal or composed by the EA-based generation process that we describe in the following subsection.
2. Website versions, each of which has the version number, the corresponding period of activity, URI of the homepage, and screenshots. Each screenshot is characterized by resolution, compression parameters (default is JPEG 100), chapter name, and associations to the corresponding Tasks the chapter is engaged in.

The solution quality. The solution quality can be associated with prototype, website or screenshot. It contains an expandable list of quality attributes and their values, which are either provided by users or experts, or obtained automatically by running the WI miner or the screenshot analyzer, in the course of the CB population.

B. *The Components' Repository and the Evolutionary Optimization Process*

The second major part in the portal is meta-repository of components that is needed to facilitate the work of the EA generator. The main storage of the components is the framework's repository (modules and themes at drupal.org), but accessing it in every step of the EA would make the generation too slow, hence the necessity of the local copy. Its synchronization with the main repository is performed via API at the portal administrator's request. The meta-information about the components collected from the repository (see Fig. 3) include the component's quality attributes (number of downloads, maintenance statues, etc.) and the taxonomy categories that can be related to the Tasks model of the project.

Fig. 3. A component's page in the meta-repository.

The new solution (web UI) generation stages correspond to the EA we described in [9], with the particulars introduced by the CBR process and the portal implementation:

1. The new project is created in the portal's CB, with Tasks, Users and other input information specified.
2. Similar projects (cases) with solutions having high values for quality attributes are retrieved from the CB.
3. A new generation of intermediate solutions is composed from the components (detailed description is provided in the next sub-section).
4. Fitness function for the new solutions is evaluated based on their similarity with the reference solutions as assessed by the target Users subjective impressions model and the screenshot analyzer (see [6] on the similarity assessment).
5. Until EA's finishing conditions are met, genetic operators (cross-over and mutation) are applied to create new generations of solutions.
6. If the new solution is acceptable, retain it in the CB as prototype. If the project later goes live, start collecting the quality attributes for the website.

C. *WUI generation*

Our approach to WUI generation is based on three distinct dimensions of a website: functionality, layout (page structure) and visual appearance. For each of the dimensions, the website is configured through the means provided by the framework, but without relying on its GUI. Since each generated website is essentially an intermediate or ultimate solution in the EA, genetic operators of cross-over and mutation are supported.

Functionality. For the website functionality generation, we devised the software tool consisting of the two distinct parts:

Component picker. The input data are Domain and specification of the Tasks model. The output is a list of the functional components – Drupal modules. The component picker can either use pre-defined configurations or match the keywords from the Tasks model to the modules' categories. The quality attributes of the modules (such as version status and recency, # of downloads/installs, # of open bugs, etc.) is considered by the picker when choosing between several modules appropriate for a required functionality. This choice needs certain degree of randomness, to implement the genetic operators.

Website installer. The input data for the installer are the list of the functional components and the reference to the components' repository location (drupal.org or the local repository). The output is the installed website and the link to its homepage, as */projects/$EAG>*, where *$EAG* is the projects' name supplemented by the generation number in the EA. The installer controls the availability and compatibility of the modules, installs the dependency modules (that are not in the list, but required by the others) and logs the generation elapsed time. It was implemented through the use of *Drush* project and sh-scripts (*make.yml* is the file with the list of the functional components):

```
./drush.sh make make.yml projects/$EAG;
```

```
./drush.sh si standard -r projects/$EAG
--db-url=mysqldb --db-prefix=$EAG --site-
name=$EAG --locale=ru;
```

Page structure (layout). In Web Engineering practice, each functionality item implemented for website's front office has one or several related user interface elements. Correspondingly, in Drupal most front-office modules have *blocks* (groups of UI elements) placed in one of webpage's *regions* and ordered within a region by their *weight*. The currently default list of regions in Drupal (Twig template engine) has 10 items: *page.header*, *page.primary_menu*, *page.content*, etc., which are sufficient for many conventional websites. The placement and order of blocks then can be programmatically set via *Drush extras* commands, such as:

```
drush block-configure --module=block --
delta=block_delta   --region=page.header
--weight=10
```

Genetic operators are then applied to *weight* and *region*. For more advanced solutions that need to go beyond the 10 default regions, customized layouts can be created as Drupal's *Panels*. They also have added benefits of being able to use groups of pre-made styles (*Classy Panel Styles*) allowing flexible adjustment of visual representation in whole regions or panels.

Visual appearance. Most of Drupal's themes have adjustable parameters (stored in JSON format) shaping their visual presentation, such as colors, font sizes and families, etc. E.g. the parameters in the *Palette* group affect the coloring:

```
palette:
  top: '#dada4a'
  background: '#ffffff'
  bottom: '#161617'
```

The theme's parameters constitute a significant part in the website's "genome" and can be affected by the genetic operators. In Fig. 4 we present an example of "mutation" for the two of the above Palette's parameters (Bartik theme)

```
top: '#daba4a'
bottom: '#361617'
```

in the subsequent generation of solutions composed in the EA.

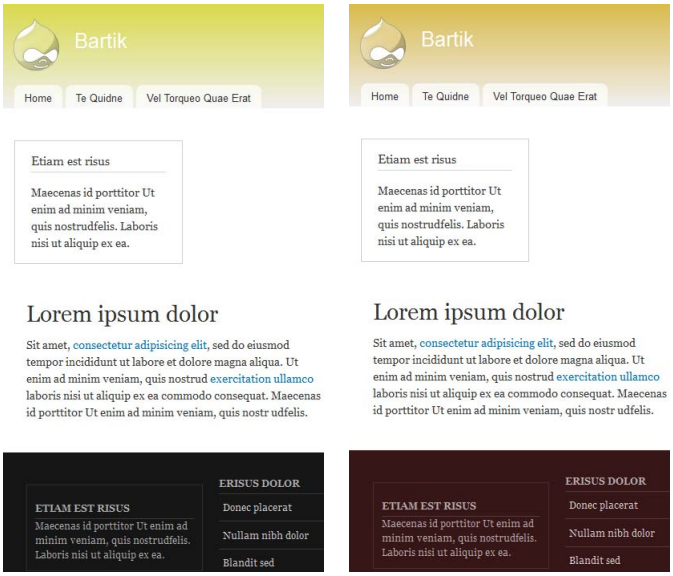


Fig. 4. Theme's mutated colors in generation N (left) vs. N+1 (right).

The website composition process starts with a basic theme and applies the genetic operators to create a sub-theme of each new solution to base on. The basic theme needs to have the number of parameters appropriate for the problem (more parameters mean more flexibility, but EA's convergence may take much longer time). Adaptiveness and support of Drupal's Panels Everywhere functionality (to facilitate use of the panels' restyling) are also desirable. We implemented the dedicated Drush module to adjust the themes' parameters from command line (see https://github.com/vkhvorostov/subtheme_color).

IV. Conclusions

The CBR application for reusing WUI designs has certain particulars, since parts of available solutions cannot be reused directly, due to technical and legal obstacles. We outlined approach for engineering new website designs from components while still referring them to exemplar solutions retrieved from a case base. The major ingredients of the process are:

- the well-organized case base with mechanisms for auto-populating and maintaining it, as well as for retrieving and retaining the cases;
- the components' repository and the organizing framework, providing the means for managing the components and facilitating the WUI composition;
- the evolutionary optimization algorithm with fitness function reflecting similarity between the generated new WUI designs and the exemplary solutions (the function is based on trained user behavior and perception models [6]);
- the algorithm for composing the new WUI designs from the components, using the means provided by the framework to apply the genetic operators to the websites' functionality, layout and visual appearance.

Some of these points are still being implemented, and in the current paper we focused on the composition and described implementation of the case base, the supplemental software, the components' repository, and the generation process. Our plans for further work include running the EA with various genome sizes (corresponding to organisms of different complexity) and assessing the quality of the resulting solutions with real users.

Acknowledgment

The reported study was funded by RFBR according to the research project No. 16-37-60060 mol_a_dk. We thank S. Heil and M. Keller who took part in the WUI analyzer development.

References

[1] R.L. Glass. Facts and fallacies of software engineering. Addison-Wesley Professional, 2002.

[2] R.L. Mantaras et al., "Retrieval, reuse, revision and retention in case-based reasoning". The Knowledge Engineering Review, 20(3), pp.215-240, 2005.

[3] R.G. Rocha, R.R. Azevedo, Y.C. Sousa, E.D.A. Tavares, and S. Meira, "A case-based reasoning system to support the global software development". Procedia Computer Science, 35, pp.194-202, 2014.

[4] A. De Renzis, M. Garriga, A. Flores, A. Cechich, and A. Zunino, "Case-based reasoning for web service discovery and selection". Electronic Notes in Theoretical Computer Science, 321, pp.89-112, 2016.

[5] F. Marir, "Case-based reasoning for an adaptive web user interface". In The Int Conf on Computing, Networking and Digital Technologies (ICCNDT2012), pp. 306-315, 2012.

[6] M. Bakaev, "Assessing similarity for case-based web user interface design". In Proc. Digital Transformations & Global Society (DTGS 2018). – in print.

[7] R. Kumar et al., "Webzeitgeist: design mining the web". In Proc. of the SIGCHI Conference on Human Factors in Computing Systems, pp. 3083-3092. ACM, April 2013.

[8] M. Gaedke and J. Rehse, "Supporting compositional reuse in component-based Web engineering". In: ACM symposium on Applied computing, vol. 2, pp. 927–933, 2000.

[9] M. Bakaev and M. Gaedke, "Application of evolutionary algorithms in interaction design: from requirements and ontology to optimized web interface". In Proc. IEEE ConRusNW, pp. 129-134, 2016.

[10] E. Folmer, "41. Interaction Design Patterns. The Glossary of Human Computer Interaction". The Encyclopedia of HCI, 2012.

[11] C. Kruschitz and M. Hitz, "Human-computer interaction design patterns: structure, methods, and tools". Int. J. Adv. Software, 3(1), 2010.

[12] L. Paganelli and F. Paterno, "A tool for creating design models from web site code". Int Journal of Software Engineering and Knowledge Engineering, 13(02), pp.169-189, 2003.

[13] C.R. Varnagar, N.N. Madhak, T.M. Kodinariya, and J.N. Rathod, "Web usage mining: a review on process, methods and techniques". In Proc. Int Conf Inf Communic and Embedded Syst (ICICES), pp. 40-46 (2013).

[14] M. Bakaev, V. Khvorostov, S. Heil, and M. Gaedke, "Web Intelligence Linked Open Data for Website Design Reuse". In Proc. International Conference on Web Engineering, pp. 370-377. Springer, Cham, 2017.

[15] M. Bakaev, S. Heil, V. Khvorostov, and M. Gaedke, "HCI Vision for automated analysis and mining of web user interfaces". In Proc. International Conference on Web Engineering (ICWE 2018). – in print.