Software architecture components of an abstract framework for assessment in e-learning

Milen Y. Petrov, and Vladimir A. Vlaykov

Abstract—These days, software engineering skills are under-evaluated. Software projects, software product lines and vast amounts of open-source code and frameworks are pushing software engineers up to their limits in order to manage software tasks. The current paper puts an accent on the implementation of the client and repository architecture components from an abstract event-driven interoperability framework for assessment in e-learning. Reference architecture and implementation enforce development and evaluation of the abstract framework. The proposed solution is reviewed and discussed and some advantages and disadvantages are pointed out.

Keywords—e-learning, assessment, event-driven framework (EDF), software engineering skill assessment

I. INTRODUCTION

These days, software engineering skills are under-evaluated. Software projects, software product lines and vast amounts of open-source code and frameworks are pushing software engineers up to their limits in order to manage software tasks. The Eclipse foundation estimates 24 million lines of code (LOC) reported in years 2009 and 2010 [1, 2]. Acquiring deeper technical knowledge and skills is important for every programming practitioner [3]. As new technologies literally appear every day and programming languages grow in size and functionality, it is hard to evaluate the current best practices together with new language features [4], [5]. Different approaches for evaluation of student results are proposed, such as service-oriented peer assessment architectures [6], stand-alone assessment tools [7], and traditional e-learning systems [8] among others. We propose an architecture framework for detailed evaluation of tasks performed by university course students in software engineering and more precisely in courses on programming fundamentals (based on C/C++ programming languages), object-oriented programming fundamentals (based on Java programming language) and data-structures (based on Java). The current paper describes details on a reference implementation of components of an event-driven framework (described below in part II) for student assessment. The paper continues with a short explanation of an abstract event-driven framework for interoperability in e-learning systems. In the third part, we describe the methodology of research – from abstract concepts to software implementation. In part IV of the current paper, a specific architecture and the underlying components are presented. In the next part, the interaction sequence between the repository components is described. In part VI, we have the relevant use-cases for testing and evaluation of the architecture components. Finally, in part VII, we have listed some conclusions and an analysis of the results.

II. EVENT-DRIVEN FRAMEWORK FOR INTEROPERABILITY IN E-LEARNING SYSTEMS

In previous research papers [9], [10], an abstract event-driven framework has been proposed to support data-exchange and interoperability (I14Y) in e-learning systems. In short, that framework defines the following main components: client, repository, monitored repository and validators. The interaction between components is described on Fig.1 below.

A. Client

Client – the event-driven architecture client is an actor. It could be a learning materials author, an instructor and/or a system or tool which is handling design and implementation of interoperable learning content. Designing interoperable content or content conforming to some of the learning specifications, i.e. one managed by IMS Consortium [11], [12], is not an easy task. Interoperability is described with the so-called conformance statements – a table with informal descriptions of different aspects of the specifications. These statements tend to be verified by hand and completing such an operation is not so easy. Problems come from the complexity of specifications and the lack of user-friendly tools to support authoring activity.
In the area of e-learning, repository means some kind of data warehouse. There is nothing special in that part of the framework, except that the repository can contain both ready-to-use learning objects (conforming to some of the specifications or standards, such as the IEEE LOM standard [13]), or just decoupled resource assets (like images, documents, pages). It also must collaborate with a special type of object – a monitored repository (described below), which must be able to handle client requests for state changing.

C. Monitored Repository

The monitored repository is a concept of a special kind of repository (similar to the one described above). What is new here – the two types of repositories – one for hosting materials and assets, and one for editing, are totally decoupled. Such differentiation of repositories is the major advantage of the presented framework. With repository separation in two parts, the task of design and development is decomposed.

D. Validators (V)

The one or more interoperability validators are decision functions returning results (i.e. true or false, or something else) which depend on input data and its internal state. Validators can be: a) with internal state, b) without internal state – or the so-called stateless validators – depending only on data, passed as parameters.

III. RESEARCH METHODOLOGY

The methodology which we are using in the current paper is an adoption of the event-driven abstract framework (described in part II above) towards working components from a software system. More precisely, we are presenting only two components from the abstract framework architecture here – client and repository. Starting from an abstract framework and going through the implementation of each of the architecture components, the whole framework can be evaluated. The current paper focuses on the architecture and working sequence of the presented components. Afterwards, an evaluation of the results is presented and some conclusions and future work is pointed out. How this paper’s research work fits in the whole architecture is depicted on Fig.2 below.

The mid-term evaluation presented in this paper is done after the implementation of the repository component and one sample use-case scenario.

IV. ARCHITECTURE AND IMPLEMENTATION OF CLIENT AND REPOSITORY COMPONENTS

A. Introduction

One of the requirements of the reference implementation is to use mainly open-source software components and servers. Repository components and deployment diagram is shown on Fig. 3. It contains one client layer and three application layers - database layer, application server layer and SMTP server middleware.

B. Overview of architecture components

The client can be either a thin developed web client or a fat client – any standard mail client. The database layer of the repository is implemented and tested with Oracle and PostgreSQL servers. The SMTP layer is a software component which is responsible for sending and receiving e-mail messages. The application layer is an advanced architecture layer, which consist of following sub-components: database interface, web interface developed on Struts technology [14], incorporating CKEditor [15], deployed on Apache Tomcat container [16], JavaMail component [17], SubEthaSMTP component [18], James Mime4j [19] and CyberNeko HTML Parser [20]. The repository component is developed as an SMTP listener which accepts messages composed as e-mails.
C. SubEthaSMTP

SubEthaSMTP is an open-source Java library that provides an SMTP server component for receiving mail in the abstract Java interface. The SubEthaSMTP component can be used from almost any application for processing e-mail. SubEthaSMTP library is extended with custom code and used as part of the solution, its function being to behave as a listener which runs in the background and listens for incoming messages. The SubEthaSMTP-based component in the solution is not concerned with storing queues of messages like a full-fledged SMTP server. An incoming message is either directly accepted or refused.

D. James Mime4j

Mime4j is a part of the Apache James project. It provides a parser (MimeStreamParser) for electronic communications in pure RFC822 and MIME format. It uses a callback mechanism to report events in the processing of an e-mail message, triggered by the header, the beginning of the message body, etc. Mime4j interface is very similar to the SAX XML parser. There are two types of APIs through which the message can be analyzed – one is iterative, using class MimeTokenStream, and the other is based on events, using class ContentHandler. In the solution, a custom code-extended Mime4j component performs the important function of parsing e-mail messages that the SubEthaSMTP-based component determines need to be received by users. Mime4j is on the level immediately after SubEthaSMTP in the structure of the Mail Delivery Agent (MDA) part of the repository in our solution.

E. NekoHTML

NekoHTML is an HTML scanner and tag balancer which allows parsing HTML documents and access to the information within using standard XML interfaces. The parser can scan HTML files and correct common errors made in the construction of HTML documents. NekoHTML adds missing parent elements, automatically closes elements with optional closing tags, and can cope with poorly balanced tags of several items listed on one line. NekoHTML is written using the Xerces Native Interface (XNI), which is the basis of the implementation of the XML parser Apache Xerces2. This allows use of the NekoHTML parser with existing XNI tools without modification or rewriting code. In our solution, NekoHTML is used for scanning the bodies of HTML messages and the removal of some unwanted tags and JavaScript attributes, which increases user security, as well as the stability of displaying HTML messages within the boundaries of the solution.

F. JavaMail

JavaMail is a Java API which is used for reading and creating, receiving and sending e-mail messages through use of protocols SMTP, POP3 and IMAP. JavaMail is relatively simple and easy to use, which makes it a preferred choice in case detailed processing of e-mail messages is not required. Since March 2009, JavaMail is open-source software. JavaMail API is used solely for sending e-mail messages in our solution. We do not use it for receiving since SubEthaSMTP, in combination with Mime4j and NekoHTML, offers a more detailed means to process incoming messages. JavaMail’s role as part of the repository is to compose an e-mail message based on user input in a web client and then pass it to the configured SMTP server responsible for sending e-mail.

G. CKEditor

CKEditor is a WYSIWYG rich text editor for use in web applications. It provides text editing functionality similar to what is seen in the likes of Microsoft Word and OpenOffice. In the current solution, users can use CKEditor to compose e-mail messages in a web client. The resulting messages are in HTML format.

V. INTERACTION SEQUENCE BETWEEN ARCHITECTURE COMPONENTS OF THE REPOSITORY

The interactions between different components of the architecture are depicted on the sequence diagram on Fig.4. Since the diagram describes receiving a message, the only component which is not mentioned here is the one using JavaMail to pass an outgoing message to the Mail Transport Agent (MTA) which delivers e-mail messages. Upon receiving a message, the SMTP listener using SubEthaSMTP checks the recipients and, if they are approved, the message is passed to the parser using Mime4j. There, the message is processed and passed to the HTML parser for cleanup. Finally, a processed and formatted form of the initial message is returned to the...
listener and then passed to the manager/database interface, which subsequently stores it into the database.

VI. USE-CASES FOR TESTING AND EVALUATION OF THE ARCHITECTURE COMPONENTS

A. Use-case 1

The use-case diagram describes the basic usage of the described solution. The user checks the existing messages (internal for the system or external, e-mail-based ones) and can then read one of them and/or download attached resources), or compose and send a new message (attaching resources, if necessary).

B. Use-case 2: Software engineering courses - fundamentals

Algorithm for automatic evaluation of predefined software programming tasks is described in source code on Fig. 6 below:

line 0: for-each skill-in-development (SkillLevel)
line 1: foreach-student S
assign Task T
line 2: S performs tasks and submits back to the system
line 3: perform evaluation on several different steps
- check for formal criteria (lines-of-code, using comments) for each artifact required (can be number of lines of the requirement (or number of pages if task is in several pages)) – usually above 5
- check internal database for plagiarism
- check asset-specific evaluators
- assessment indicator statistics and self-improvement
Fig.6 Transformation from abstract framework to software component artifacts

VII. CONCLUSION AND FUTURE WORK

Design and development of the entire system are still unfinished and the conclusions made above cannot be applied to the results which can be achieved with a fully working system. Nevertheless, it is an important step of building software components to evaluate them as soon as they are designed and developed, in order to find and remove any possible defects – especially in the requirements, high-level architecture (abstract framework), low-level architecture and software design of the system. As a result of the analysis made, here are the main advantages of the architecture and the underlying abstract framework:

- Events can be fired as common e-mails
- The framework and architecture are content independent (any kind of learning objects format can be used, as long as it is wrapped as an e-mail
• Easy to use clients – any user with basic internet skills can handle a browser and/or e-mail client
• Easy to extend
• Based on open-source tools – can be easily replaced with better tools (either open-source or commercial)
• Final components have been tested with the scenario from use-case 1 (described above) and show very good performance and stability

Here are also some disadvantages:
• In order to evaluate the whole abstract event-driven interoperability framework for assessments, all components should be implemented
• Second use-case is not evaluated, as not all of the described architecture components are available yet

Nevertheless, the results of the evaluation and analysis are very promising for future implementation of the framework.

REFERENCES