AN APPROACH OF BEHAVIORAL SOFTWARE MODELS COMPARISON BY MEANS OF THEIR FORMAL REPRESENTATION ANALYSIS

Olena Chebanyuk, Mykyta Krainii

Abstract: *Paper presents in approach of behavioral software model comparison by means of their analytical representation. This idea is not new. Known researches are concentrated on analysis of software model structure (Chebanyuk, 2018), (ECL, 2015). Advantages of the presented approach is that it allows to analyze both structure and semantics of software model. Structural analysis is based on software model graph representation. Semantic analysis is based on analysis of text signatures of software models.*

Keywords: *Theory of Categories, UML, Graph Theory, Requirement Engineering, Software Repository, Software Reuse.*

DOI: *https://doi.org/10.54521/ijita28-01-p03*

Introduction

The study of storage and retrieval methods of software assets in software libraries gives rise to a number of paradoxes: while this subject has been under investigation for nearly two decades, it still remains an active area of research in software reuse and software engineering; this can be explained by the observation that new technologies, keep opening new opportunities for better asset packaging, better library organizations, and larger scale libraries thereby posing new technical challenges (Lee, J. et. al., 2003), (Hummel, O. et. al., 2004). Also, while many sophisticated solutions have been proposed to this problem, the state of the practice in software reuse is characterized by the use of ad-hoc, low-tech methods; this can be explained by the observation that most

existing solutions are either too ineffective to be useful or too intractable to be usable (Tiwari, U. K., & Kumar, S., 2020).

Finally, while it is difficult to imagine a successful software reuse program without a sophisticated, well-tuned, systematic procedure for software component storage and retrieval, it seems many successful software reuse experiments rely on trivial methods of component storage and retrieval. This can be explained by the observation that, in the current state of the practice, software libraries are not the bottleneck of the software reuse process (D'silva V. et al, 2008).

This paper presents an approach for improvement of methods of software assets storage and retrieval in software libraries, by models comparison process, by means of their formal representation. These criteria afford us an exhaustive and uniform basis for assessing and comparing individual methods and classes of methods.

Area of using proposed approach of behavioral software models comparison

Activities aimed to design repository of software components are concentrated both in forward and reverse engineering activities.

Reverse engineering is gaining more popularity from the research community as an act of analyzing a software system, either in whole or in part, to extract design and implementation information that can be useful for several tasks such as software comprehension, documentation, maintenance, and re-engineering . Nowadays, reverse is used for various purposes other than software comprehension such as software testing, software reusability, updating user interfaces, migration and porting user interfaces to new platforms [19].

In terms of such a formal representation, the reverse software engineering consideration is needed also. Dealing with reverse software engineering, a system is analyzed and the goal is extraction some initial information, an example of which can be implementation, software requirements, design and

functionality of the product. There are some tools, which can help with extraction of this kind of information. For instance, Altova offers a line of products to deal with XMI, UML modeling and many other activities, which can help to achieve needed result. Retrieving such kind of data is performed in need for education, software reuse, software safety issues, to prevent harmful data extraction etc. Reverse software engineering can be implemented both for the complete system and for them partially The scheme of the process of reverse soft-ware engineering is shown further.

Working with the open source components in repository, it is rather simple task to retrieve the initial raw code and then by means of existing tools transform it into some kind of behavioral software model, for instance, Communication or Sequence diagram.

When data for this kind model is retrieved, the next step is provide user with the formal representation of it for further processing and comparison with other components, data of which can be retrieved also by means of the tools given above.

For the recommended approach, the ReqIF format for data analysis is used, particularly, ReqIF files which are created via RMF (ProR) plugin for Eclipse. Thus, in this work we deal with the analysis of specification, as well as with the formal representation of behavioral software models. Altova products allow user to generate UML models from the source code, therefore, having access to the source code, the process of reverse software engineering can be performed to retrieve information about the product specification, and then use in given tool to compare the structure and metadata.

The following steps should be performed to make a conclusion, whether selected component can be used in given project or not:

Structural comparison

- retrieve the source specification;
- obtain UML diagram graph representation from UML representation of requirements specification (Chebanuyk, 2018), (Chebanyuk et al., 2018).
- obtain UML diagram graph representation from software components ;

- comparing these diagrams with the aim of searching matching fragments of each diagram, which is represented as sub-paths of graphs.

Semantic comparison:

- make conclusion about the structural relevance of diagrams. If the structural relevance is more than given percent it is necessary to perform additional comparison because structures of different algorithms can occur practically the same;
- extract signatures from behavioral software models (for example for Use Case Diagram they are signees of precedents and comments) and compare with the keywords of software component problem domain (nouns and verbs);
- calculate the matching criteria;
- make a conclusion about further reuse of software component in considering software project.

The component from repository should contain the file with following additional attributes to be handled by a tool: its name, purpose, URI of com-ponent, URI of Software Model, license, structure, annotation, keywords nouns, and keywords verbs. The last ones can be stored in arrays, such as keywords [noun], keywords [verb]. This metadata is created by means of domain analysis and then should be handled to compare it with the goal model (Chebanyuk & Palahin, 2019).

First, a check for a component structure is performed. If the structure of component corresponds to the given specification, the keywords should be considered to make a conclusion. Thus, the keywords nouns are checked, and in case of coincidence of 60%, the keywords verbs are checked. If the re-sult higher than 60% is received, such component is recommended for usage in specified project.

The application, which implements the given approach, should take the set of components, and for each component the following actions should be prepared: generate sequence diagrams for component (Altova UModel tool can be used in

order to manage it); design graph representation of each component; save the graph representation; input the metadata stated above, for performing the semantic comparison; take into consideration the source specification; de-sign the sequence diagram of it; prepare analytical representation of such a di-agram; compare the received structures; calculate matching criteria as the final step.

Overview of requirement specification preparation techniques and tools

The Rule Interchange Format (RIF)

The World Wide Web Consortium charted the Rule Interchange Format (Rule Interchange Format) Working Group to create a standard for exchanging rules among rule system sin 2005, and specially, to the Web rules engines (RIF, 2005). Rule Interchange Format did not try to implement a single one rule language, it concentrated on the interchange, which contrasts to other different Semantic Web standards. An example of the last ones are RDF (RDF, 2018), OWL (OWL, 2018), and SPARQL (SPARQL, 2018). After its creation, it was at once clear, that a single language is not able to fulfil requirements of vast majority of popular paradigms for business modelling, particularly, in the knowledge representation. However, the rule exchange alone is considered a rather intimidating task. There exist three broad categories of the known rule systems, which are first-order, logic programming and action rules. The paradigms stated above, however, share few in the way of syntax and semantics. Additionally, large differences between systems exist even in terms of the same paradigm (RIF, 2005).

The Working Group took the following approach: designing a family of languages, which are known as dialects, where syntax and semantics are strictly specified. Thus, the family of dialects, referred to Rule Interchange Format are intended to be extensible and uniform. The uniformity of it means that it is expected that dialects should share existing semantic and syntactic apparatus as much as possible (Gordon, T. F. et. al., 2009). On the other hand, experts which are motivated enough should be able to define new Rule

Interchange Format dialect as an extension in terms of syntax to the existing Rule Interchange Format dialect, where new elements are appropriate to the desired additional functionality. It explains the principle of extensibility. The new Rule Interchange Format dialects can be not standard at their definition, but eventually can become the one.

The word format, which is called Rule Interchange Format, can be somewhat of an understatement, due to the accent on the strictness. It is far beyond being a simply format. The format's concept is essential to the intension for Rule Interchange Format usage, however. The exchange media between the rule systems is ultimately XML (XML, 2016), which is created as the exchange data format. According to the idea of rule exchange, which is used in Rule Interchange Format, various systems provide syntax mappings from the native languages to the dialect of Rule Interchange Format and vice versa. Preserving the semantics is the most valuable requirement for such mappings, therefore one system can provide the rule set to another by means of talking with a suitable dialect, which should be supported by both of systems.

The two kinds of dialect, basically, on which the Rule Interchange Format Working Group is concentrated, are logic-based and the ones for rules with action. Basically, languages employing some type of logic, like first-order logic (often restricted to Horn logic) or non-first-order logics underlying the various logic programming languages (e.g., logic programming under the well-founded or stable semantics). The rules-with-actions dialects include production rule systems, such as Jess, Drools and JRules, as well as reactive (or event-condition-action) rules, such as Reaction RuleML and XChange.

Due to the limited resources of the Rule Interchange Format Working Group, it defined only two logic dialects, the Basic Logic Dialect and a subset, the Rule Interchange Format Core Dialect -Production Rule Dialect. Other dialects are expected to be defined by the various user communities (RIF, 2005), (Bolye H., et. al., 2007).

Rule Interchange Format-Framework for Logic Dialects is a formalism in which both syntax and semantics are described through a number of mechanisms that are commonly used for various logic languages, but are rarely brought all

together. Amalgamation of several different mechanisms is required because the framework must be broad enough to accommodate several different types of logic languages and because various advanced mechanisms are needed to facilitate translation into a common framework. Rule Interchange Format-Framework for Logic Dialects gives precise definitions to these mechanisms, but allows well-defined aspects to vary. Therefore, for any Rule Interchange Format dialect to become a standard, its development should start as a specialization of Framework for Logic Dialects and extensions to (or, deviations from) Framework for Logic Dialects should be justified.

Dialects for action rules include production rule systems such as Jess, Drools, and JRules, and reaction rules (or conditional action rules) such as Reaction RuleML and XChange.

Due to the limited resources of the Rule Interchange Format working group, only two logical dialects have been defined, the basic logical dialect (Rule Interchange Format-Basic Logic Dialect) and a subset, the basic Rule Interchange Format dialect, which is shared with the Rule Interchange Format-Production Rule Dialect. The Manufacturing Rules Dialect (Rule Interchange Format-Production Rule Dialect) is the only rule and action dialect defined by the group. It is expected that other dialects will be defined by other user communities.

Rule Interchange Format-Framework for Logic Dialects has the following components:

- syntactic framework;
- semantic framework;
- XML serialization framework (W3C, 2013).

Syntactic framework defines the next types of Rule Interchange Format terms:

- constants and variables;
- datatypes;
- positional terms;

- terms with named arguments;
- lists;
- frames;
- classification;
- equality;
- formula terms;
- external;
- aggregation;
- restriction.
- signatures;
- remote (W3C, 2013).

Terms are used to define several types of Rule Interchange Format-Framework for Logic Dialects formulas. In addition, Rule Interchange Format-Framework for Logic Dialects introduces extension points, one of which allows the introduction of new kinds of terms. An extension point is a keyword that is not a syntactic construct, but a placeholder that is supposed to be replaced by specific syntactic constructs.

Depending on their needs, dialects can decide which symbols have which signatures.

Rule Interchange Format dialects are required to replace extension points with zero or more specific syntactic constructs of an appropriate kind. Note that in this way extension becomes part of specialization.

Semantic structures determine how the different symbols in the alphabet of a dialect are interpreted and how truth values are assigned to formulas.

Entailment is fundamental to logic-based dialects. Given a set of formulas (e.g., facts and rules) G, entailment determines which other formulas necessarily follow from G. Entailment is the main mechanism underlying query answering in Databases, Logic Programming, and the various reasoning tasks in Description Logics.

A specification of the XML syntax for Rule Interchange Format-Framework for Logic Dialects, including the associated XML Schema document.

A specification of a one-to-one mapping from the presentation syntax of Rule Interchange Format-Framework for Logic Dialects to its XML syntax. This mapping must map any well-formed formula of Rule Interchange Format-Framework for Logic Dialects to an XML instance document that is valid with respect to the aforesaid XML Schema document (RIF, 2005).

The Rule Interchange Format (ReqIF)

The Requirements Interchange Format (ReqIF) defines such an open, non-proprietary exchange format. Requirement information is exchanged by transferring XML documents that comply to the ReqIF format (ReqIF, 2013).

Most modern requirement authoring tools emulate word processors, but offer additional features. This allows authors of requirement specifications who have been using word processors to continue working in a similar manner, but enjoy the benefits of a tool specialized for authoring requirements (Bolye H., et. al., 2007), (ReqIF, 2013). Table 1 explains some differences between using word processors and requirement management tools. The full list of these differences is represented in (ReqIF, 2013).

Table 1. The word processing features of requirement authoring tools

Feature	**How requirement authoring tools handle it**	**Example**
Uniquely identify requirements	Requirement authoring tools allow to distinguish individual requirements and to automatically create a unique identifier for each	

Feature	How requirement authoring tools handle it	Example
	requirement.	
Associate attributes with the requirements	A user of a requirement authoring tool can define arbitrary attributes and attach them to requirements. Typically, a set of requirements shares the same attributes. However, these attributes may have different values for each requirement, and the values may have different underlying data types.	A user of a requirement authoring tool defines the attributes "id," "description," "priority," "status," and "department" as mandatory for a specification. The "priority" attribute has an integer data type, the "status" and "department" attributes have an enumeration data type, and the "description" attribute has a string data type. Each requirement may have a different value for each of these attributes.
Establish relations between requirements	A user of a requirement authoring tool can define relations between requirements.	Example purposes of relations: a) to establish traceability b) to connect non-functional to functional requirements.

Feature	How requirement authoring tools handle it	Example
Group relations	Some requirement authoring tools allow the user to define new types of relations and to group relations by their type.	A requirement authoring tool may allow its users to define the new type "contradicts" for relations between two requirements that contradict each other, and then allow the users
		to create a group of "contradicts" relations. Such a group of relations – together with the requirements that are related by it – may support the users when reviewing and consolidating specifications.

One of the basic ideas of ReqIF is to offer the opportunity to exchange information between different installations of the same requirements authoring tool with a standardized format and that the same format can be used to exchange information between different requirements authoring tools. This clause explains two exchange scenarios: • In the first exchange scenario ("one-way"), requirement specifications of one exchange partner are provided to a second exchange partner, for example to inform the second partner about the requested requirements. • In the second exchange scenario ("roundtrip"),

requirement specifications of one exchange partner are provided to a second exchange partner as well. After that, however, the second exchange partner makes modifications to the requirements, for example to comment them concerning the feasibility. The second exchange partner transmits the modified requirement specifications back to the first exchange partner.

Object Constraint Language

The Object Constraint Language (OCL) is a formal language used to describe expressions on UML models. These expressions typically specify conditions that must hold for the system being modelled or queries over objects described in a model. When the OCL expressions are evaluated, they do not have side effects (i.e., their evaluation cannot alter the state of the corresponding executing system). OCL expressions can be used to specify operations / actions that, when executed, do alter the state of the system. UML modelers can use OCL to specify application-specific constraints in their models. UML modelers can also use OCL to specify queries on the UML model, which are completely programming language independent [4].

The Object Constraint Language appeared as an effort to overcome the limitations of UML when it comes to precisely specifying detailed aspects of a system design. Now OCL has become a key component of any model-driven engineering (MDE) technique as the default language for expressing all kinds of metamodel query, manipulation and specification requirements. Among many other applications, OCL is frequently used to express model transformations (as part of the source and target patterns of transformation rules), well-formedness rules (as part of the definition of new domain-specific languages, or code generation templates (as a way to express the generation patterns and rules). To adapt the language to these new applications, several new subversions of the language have been released.

The disadvantage of traditional formal languages is that they are usable to persons with a strong mathematical background, but difficult for the average business or system modeler to use. OCL has been developed to fill this gap. It is a formal language that remains easy to read and write. It has been developed as a business modeling language within the IBM Insurance division, and has its

roots in the Syntropy method. When an OCL expression is evaluated, it simply returns a value. It cannot change anything in the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change (e.g., in a post-condition). Because OCL is a modeling language in the first place, OCL expressions are not by definition directly executable. OCL is a typed language. To be well formed, an OCL expression must conform to the type conformance rules of the language. For example, you cannot compare an Integer with a String. Each Classifier defined within a UML model represents a distinct OCL type. In addition, OCL includes a set of supplementary predefined types. OCL can be used for a number of different purposes:

- as a query language;
- to specify invariants on classes and types in the class model;
- to specify type invariant for Stereotypes;
- to describe pre- and post conditions on Operations and Methods;
- to describe Guards;
- to specify target (sets) for messages and actions;
- to specify constraints on operations;
- to specify derivation rules for attributes for any expression over a UML model.

OCL is a descendant of Syntropy, a second-generation object-oriented analysis and design method. OCL statements are constructed in four parts:

- context that defines the limited situation in which the statement is valid;
- a property that represents some characteristics of the context (e.g., if the context is a class, a property might be an attribute);
- an operation (e.g., arithmetic, set-oriented) that manipulates or qualifies a property;
- keywords (e.g., if, then, else, and, or, not, implies) that are used to specify conditional expressions.

OCL uses a Smalltalk-based "block" syntax to allow you to define some kinds of functions conveniently and inline, but it does not provide corresponding type rules for this based on generic types. For our discussion here, we treat blocks as first-class functions and use the syntax (T1 X T2 X T3 -> T) for such a function. This is purely a syntactic convenience; functions can be modeled as objects.

Overall, the **Object Constraint Language started as a complement of the UML notation with the goal to overcome the limitations of UML** (and in general, any graphical notation) in terms of precisely specifying detailed aspects of a system design [5]. Since then, **OCL has become a key component of any model-driven engineering (MDE) technique** as the default language for expressing all kinds of (meta)model query, manipulation, and specification requirements. Among many other applications, OCL is frequently used to express model transformations (as part of the source and target patterns of transformation rules), well-formedness rules (as part of the definition of new domain-specific languages), or code-generation templates (as a way to express the generation patterns and rules). Using OCL as a transformation language has a list of drawbacks, represented in the paper (Chebanyuk & Shestakov, 2017).

Requirements Modeling Framework

The "Requirements Modeling Framework" (RMF) project is a proposed open source project under the Model Development Tools Project [6].

This proposal is in the Project Proposal Phase (as defined in the Eclipse Development Process) and is written to declare its intent and scope. The vision is to have at least one clean-room implementation of the OMG ReqIF standard in form of an EMF model and some rudimentary tooling to edit these models. The idea is to implement the standard so that it is compatible with Eclipse technologies like GMF, Xpand, Acceleo, Sphinx, etc. and other key technologies like CDO.

The Eclipse ecosystem provides a number of projects to support software development and systems engineering. However, in the open source community, one important aspect of the engineering process is very much

neglected: requirements management, consisting of a number of sub-disciplines including requirements capturing, requirements engineering, requirements traceability, change management and product line engineering, to name just a few.

The goal of RMF is to provide the technical basis for open source projects that implement tools for requirements management. The conditions for the inception of such a project are perfect: Until now, all tools for requirements engineering suffered from the lack of a standard for the exchange and storage of requirements. Each tool provider invented his own method and meta-model for requirements, thereby limiting the common user base and the possibility for exchange between tools.

This open standard could have as much impact on requirements structuring as the UML had on modeling. The implementation of the ReqIF standard as an Eclipse project could similarly be as important for the requirements community as was the implementation of UML2 in Ecore for the modeling community by having the way for such tools as Topcased and Papyrus MDT.

Providing such a project under the Eclipse umbrella would offer a possibility for many projects that are involved in requirements management to find a common implementation of the standard. It would push Eclipse in to phases of the development process where it is currently under-represented.

The RMF project's focus is the creation of libraries and tools for working with ReqIF-based requirements. The objective is to provide the community with a solid implementation of the standard upon which various tools can be built. RMF will provide a means for data exchange between tools, an EMF-based data model, infrastructure tooling and a user interface [7].

RMF will not provide support for Requirements Management. Instead, it is expected that users will use specialized tools or work with the available Eclipse tooling (EMF Compare, version control integration, etc.). Generic or specific parts of the tooling can be hosted as part of the RMF project.

The figure 1 depicts the architecture of the current development and indicates which elements will be part of the initial contribution:

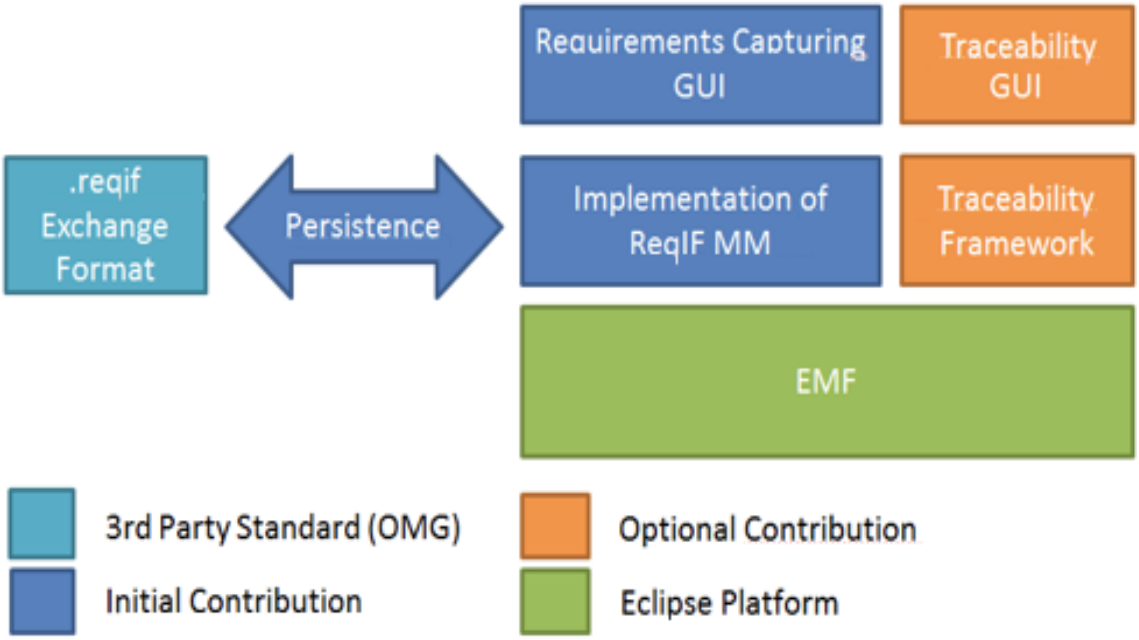


Figure 1. The Architecture of RMF figure is taken from [20]

The Eclipse ecosystem will benefit from an implementation of a requirements standard to cover more aspects of system development. Currently, modelling is covered well on the low level (EMF, TMF, CDT, etc.) and high level (UML, SysML, etc.). Adding the domain of requirements capturing would extend the coverage. To promote it, we need not only a standard, but also a common implementation that tools build upon.

The implementation of a standard benefits greatly from the participation of many parties (improvement of quality, reduction of cost). In addition, long-term support through the participation of many parties is essential for many domains.

The first initial contribution of the ReqIF and RIF metamodels as .ecore models, including special (de)serializers that map the EMF-models to a ReqIF conforming standard. The model and (de)serializers are already available at itemis and need only be provided.

The initial contribution has been internally tested. A ReqIF export from production data from the automotive domain from a commercial tool has been imported into Eclipse and exported again, keeping all the structural data, with the exception of ReqIFs XHTML extensions (see table 2).

They have been implemented according to the specification, but since ReqIF is a new standard, no extensive tests with ReqIF files coming from other sources / other tools have been made.

Figure 2 depicts implementation of ReqIF format by means of other tools.

RIF/ReqIF Version	EMF Metamodel	ProR Integration	XHTML Support	Tool extension support	Current industry support	Future relevance
RIF 1.1a	**Available**	Not available	**Available**	Not available	High	?
RIF 1.2	**Available**	**Available**	**Available**	**Available**	Low	?
ReqIF 1.0.1	**Available**	Not available	**Available**	**Available**	Low	High

Figure 2. Integration of RIF and ReqIF to different development environments figure is taken from [20]

The second contribution is about ReqIF data could be edited with the default EMF editor, this is not even remotely practical: a tree view of the requirements, with the details shown in the property view, doesn't allow users to efficiently navigate requirements or get an overview of what's there [7].

ReqIF Studio

For technical and organizational reasons, two companies in the manufacturing industry are rarely able to work on the same requirements repository and sometimes do not work with the same requirements authoring tools. A generic, non-proprietary format for requirements information is required to cross the chasm and to satisfy the urgent industry need for exchanging requirement

information between different companies without losing the advantage of requirements management at the organizations' borders [8].

The Requirements Interchange Format (ReqIF) described in this RFC defines such a tool-independent exchange format. Requirement information is exchanged by transferring XML documents that comply to the ReqIF format.

The ReqIF team expects that making the Requirements Interchange Format an OMG standard increases the number of interoperable exchange tool implementations on the market, fosters the trust of companies exchanging requirement information in the exchange format and provides safety of investments to tool vendors.

Epsilon Comparison Language

The aim of the Epsilon Comparison Language (ECL) is to enable users to specify comparison algorithms in a rule-based manner to identify pairs of matching elements between two models of potentially different metamodels and modelling technologies. In this section, the abstract and concrete syntax, as well as the execution semantics of the language, are discussed in detail (ECL, 2015).

In ECL, comparison specifications are organized in modules (EcLModule). As illustrated below, EclModule (indirectly) extends EolModule which means that it can contain user-defined operations and import other library modules and ECL modules. Apart from operations, an ECL module contains a set of match-rules (MatchRule) and a set of pre and post blocks than run before and after all comparisons, respectively (ECL, 2015).

MatchRules enable users to perform comparison of model elements at a high level of abstraction. Each match-rule declares a name, and two parameters (leftParameter and rightParameter) that specify the types of elements it can compare. It also optionally defines a number of rules it inherits (extends) and if it is abstract, lazy and/or greedy. The semantics of the latter are discussed shortly (ECL, 2015).

EMF Compare

EMF is an Eclipse project that implements model comparison for EMF 4 models. It avoids using static unique identifiers and relies more on similarity based-matching to add flexibility and usefulness in more contexts [11]. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

EMF (core) is a common standard for data models, many technologies and frameworks are based on. EMF consists of three fundamental pieces:

EMF – the core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.

EMF.Edit - The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.

EMF.Codegen – The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

Three levels of code generation are supported:

Model - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class.

Adapters - generates implementation classes (called ItemProviders) that adapt the model classes for editing and display.

Editor - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a starting point from which to start customizing.

In order to use EMF Compare for model transformation testing, it should be configured to ignore unique identifiers. For example, for every control flow or object flow reference missing when compared to its corresponding element, a reference addition of the same type appears. It is likely these differences are trivial and these references are to the same object: EMF Compare generates false positives, that is, elements that are identified as different but should not be. In addition, it is very difficult to pinpoint the differences.

It is clear that EMF Compare is not well suited for heterogeneous comparisons. The EMF Compare matching algorithm produces the same, relatively unhelpful information in that they fail to match what, semantically, we know should be a match near the top of the model hierarchy. Only straight-forward matches are discovered, such as those that have the same or similar names. For example, StateMachine, State, and PseudoState are present in both metamodels, but it is difficult to identify the differences of their children and other should-be matches at the same level are missed [12].

Mathematical foundation of software behavioral models formal representation

Set and Graph Theories for behavioural software models representation

Due to the specifics of task which if offered by an approach, set theory is the basis on which the whole formal representation of software models are held. Given the intrinsic nesting property of sets, it is natural to represent them as directed graphs: vertices will stand for sets, while the arc relation can show the

membership relation. Moreover, the main point of the whole approach for software models decomposition lies in the sphere where they are represented as directed graphs, particularly, in the structure of hierarchical tree. The switch of perspective is important: from a computational point of view, it led to many decidability results, while from a logical point of view, this allowed for natural extensions of the concept of set, such as that of hyperset. Interpreting a set as a directed graph gives rise to many combinatorial, structural and computational questions, having as unifying goal that of a transfer of results and techniques across the different areas [13].

In statement form of set representation, a well-defined description of the set elements is given [14].

Dealing with the formal representation of behavioral software models, it is important to refer to its mathematical foundation, which can be found in set and graph theory.

Development of a tool for behavioral software models comparison

The scope for the tool, which is designed to implement the stated above approach, needs to be define precisely. First, front-end of the application should allow user to upload the requirements specification, generated with the help of ReqIF format and related tools, such as ReqIF Studio or RMF plugin for Eclipse. Second, there is a need to work with graphical representation of components, with the purpose of their further decomposition and comparison. There are plenty of tools, which allow construct models from the source code, which can be Altova, or other tool, convenient for particular user.

An application should consist of front end and back end part, first should allow user to upload proper data and manipulate with it properly, second one performs the needed actions, such as analysis, comparison, and making conclusion. The files from either ReqIF Studio or RMF plugin can be stored in various formats, thus the HTML format was chosen as the most convenient for achieving needed goals. In addition, diagrams created by means of different

tools can vary not only in terms in presentation, but also in terms of data storage, so compatible formats should be defined before. The further development of a product can allow user to work with various datatypes, as well as with various formats of diagrams, created in different development environments. However, for this release, it is important to implement the approach stated above and show the solution of particular task by means of it.

For analytical decomposition of the software model, a parser should be created. It will allow retrieve the needed data from the model, stored in XMI, and process it to gain the result.

There is a need to fulfill the following requirements:

- obtain the source specification;
- receive graphical and analytical representation of the system;
- get access to repository;
- analyze uploaded system and its components with regard to their relevance;
- make a conclusion about the correspondent parts;
- offer a way for the improvement of current system.

Definition of the program for behavioral software models comparison scope

It is obvious, that the most prominent choice for implementing this kind of task is creating the web application due to it versatility, ability to effectively visualize given results and work with a particular repository for user needs. The tools and development environments for the application will be considered later on. One of the benefits of this approach is that clients are independent of the specific operating system of the user, so web applications are cross-platform ones. The approaches to the architectural design of web applications are used. The web application consists of the client and server parts, thereby implementing the client-server technology. The client part implements the user interface, generates requests to the server and processes responses from it. The server

side receives a request from the client, performs calculations, then generates a web page and sends it to the client over the network using the HTTP protocol.

Therefore, because of storage of data at the repository, design with web patterns and convenient interface for data transferring, the web application is chosen.

Functional requirements

The tool under development should satisfy the following requirements.

- F1 – the product should be able to upload files, compatible with .reqif format.
- F2 –the user should have ability to choose files from repository.
- F3 – the program should parse the data uploaded by user to get its analytical representation (Chebanyuk O. & Salem Abdel-Badeeh M., 2018).
- F4 – the data should be logically decomposed into the chains of elementary subgraphs.
- F5 – decomposed data should provide an option to make structural comparison of components.
- F6 – the program should be able to analyze data in.html format, generated via ReqIF compatible tools
- F7 – the parsed data should be analyzed to make conclusion about component structure.
- F8 – the product should provide semantic analysis by means of metadata, which is stored along with component.
- F9 – the program should present results of analysis to a user to make conclusion about the relevance of component.

The working environment of Microsoft Visual Studio 2019 was chosen for the development of a behavioral software model comparison tool. The software uses ASP.NET tools for design and implements the MVC architectural pattern.

MVC stands for the Model-View-Controller, by which the system is logically, separated into three various parts: correspondently, the model, the view and the controller. Each component is created to deal with specified development aspects of a product. Overall, it is one of the most widespread web development framework in industry standard, which allow developers to design scalable and extensible projects.

The diagram shown in the Figure 3 represents the main functionality and data flow during the work of application.

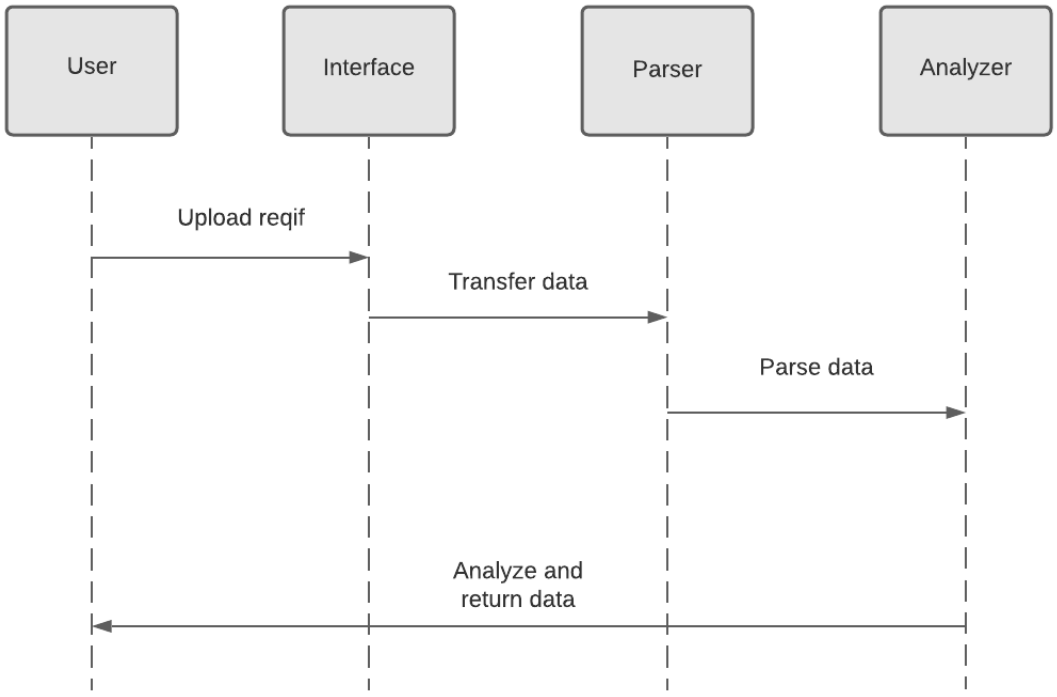


Figure 3. Sequence Diagram for the software models' comparison tool

User Interface Layer Designing

During the development of a Web application, it is important to take into consideration responsive design. Responsive Web Design is a design of web pages that ensures the correct display of the site on various devices connected to the Internet, and dynamically adjusts to the given dimensions of the browser window (BootStrap, 2019). The goal of responsive web design is the versatility of displaying website content across different devices. In order for the website to be conveniently viewed from devices of formats and with screens of different resolutions, using the technology of responsive web design, it is not necessary to create separate versions of the website for certain types of devices. One site can work on a smartphone, tablet, laptop and TV with Internet access, that is, on the entire range of devices. The main principles of responsive web design are: applying a responsive grid based layout, using flexible images, work with media queries, applying incremental improvement, smoothly rearranging blocks in responsive design when resizing the screen. The responsive web design is an obligatory point, because of widespread devices, which can use such Web applications in general.

Thus, to design the needed application, Bootstrap was chosen due to its implementation of responsive web design. Bootstrap is the free front-end framework for efficient web development. Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins. Boostrap is easy to use, due to its grounding on HTML and CSS (Bootstrap, 2019). It has responsive features, adjusting to phones, tablets and desktops. Bootstrap uses mobile-first approach, which lies in the field that mobile-first styles are part of the core framework. Moreover, it is compatible with all modern browsers without exclusions.

Business Logic Layer Designing

Below the detailed description of each class along with its functionality is given.

Req_data class is a class, which refers to the "Model" component, contains the basic functionality for data transfer between user interface and business logic layer. It processes the uploaded data, fills the data structures and allows or forbid to transfer it further.

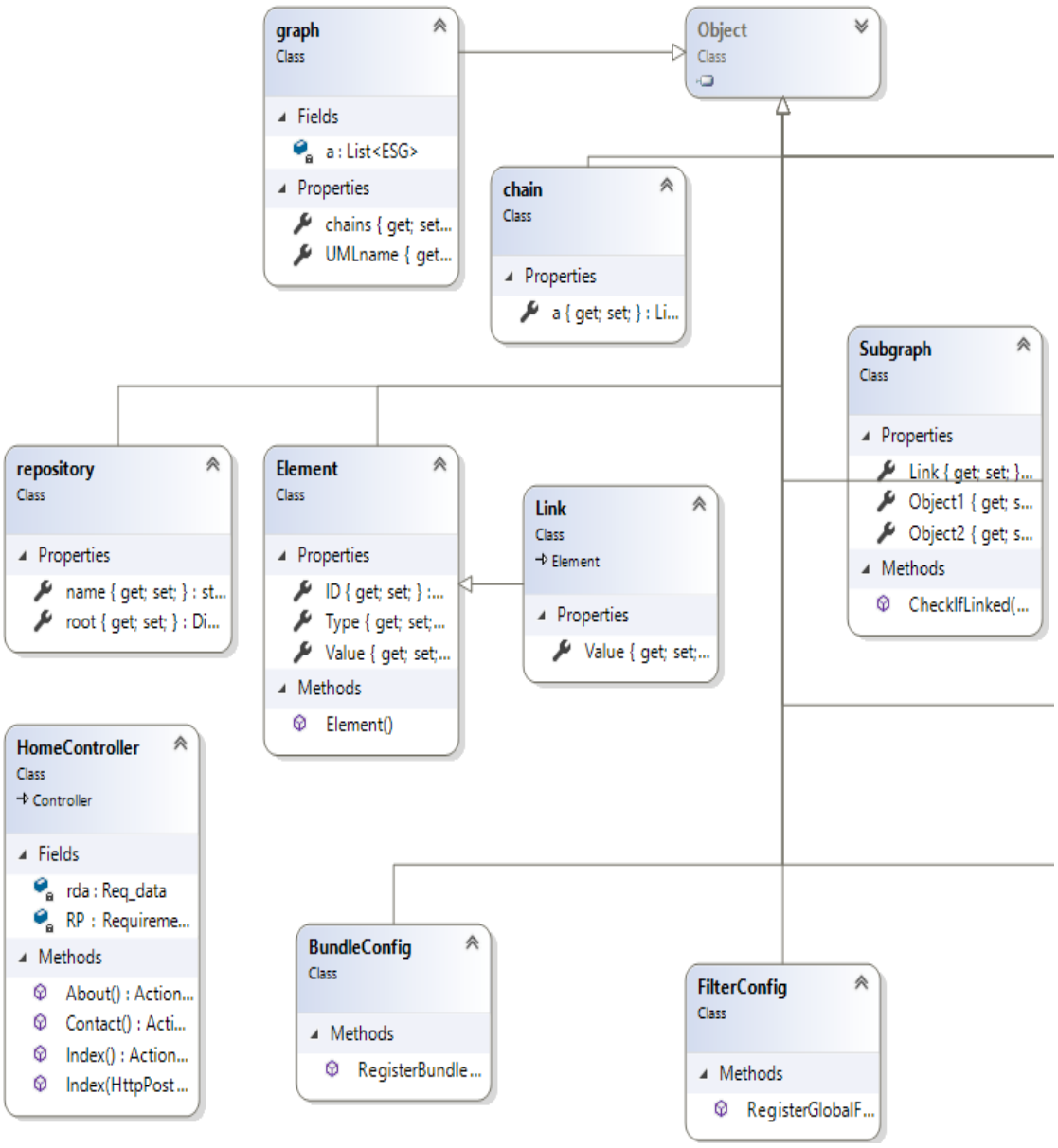


Figure 4 Class Diagram of Behavioural Software Model Comparison Tool (part.1)

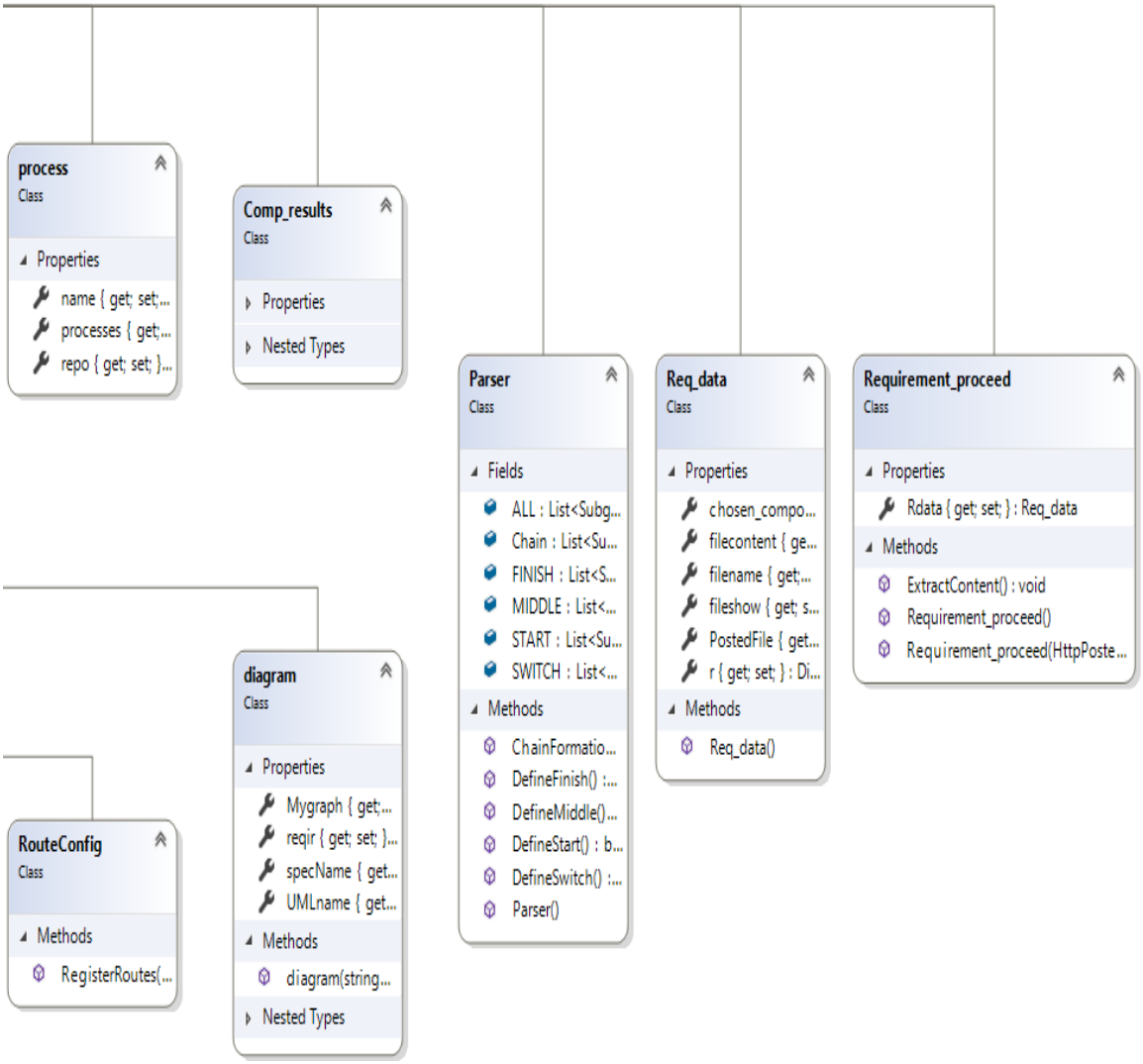


Fig. 5 Class Diagram of Behavioural Software Model Comparison Tool (part.2)

HomeController refers to a "Controller" component of a system, allowing the data transition from the Req_data to business logic layer, where the analysis and operations upon information are proceeded.

Requirement_proceed is a class of business logic, where manipulations and calculations of data take place. The processing of received data is performed at this layer, to satisfy the definition of three-layer architecture.

BundleConfig class registers CSS and JS so they can be bundled and minified.

FilterConfig class shows registered global filters. These filters are applied to all actions and controllers.

RouteConfig class shows registered routes.

Repository class contains the data and path to a current repository that is used in the session.

Comp_result, or computational resul class contains the data which is retrieved from business logic class after its processing. It stores information about structure, the diagram and other needed for user data.

Process class contains the data about processes and is also intended to work with repository.

Diagram class describes the structure of model and store data of it; it is bound with the source specification uploaded.

Element class stores the id, type and value of the object, which is received by means of LINQ query. It is the initial element, from which the complete logical chain will be formed.

Link class contains link to the following element, to show, whether the objects are linked and then they should be subdivided into correspondent groups.

Subgraph class consists of linked pair of elements and contains method to check if the elements are linked with each other.

Chain class consists of the elementary subgraphs, grouped by the same criteria to be stored separately from another one.

Graph is a class, which consists of the number of logical chains. It is the final step in analytical representation of behavioral software model.

Parser class is created to provide parsing of data by means of LINQ queries and then sort elements into groups, depending on whether they are linked or not, their position and type in chain. It defines, whether the element belongs to a

Starting, Middle, Switching or Finishing group and, moreover, is able to form logical chains from the elements.

The Figure 6 demonstrates the work of the application. For the example requirement specification, created via RMF plugin was chosen. The specification is uploaded and depicted in the window. Application accept for now only html format.

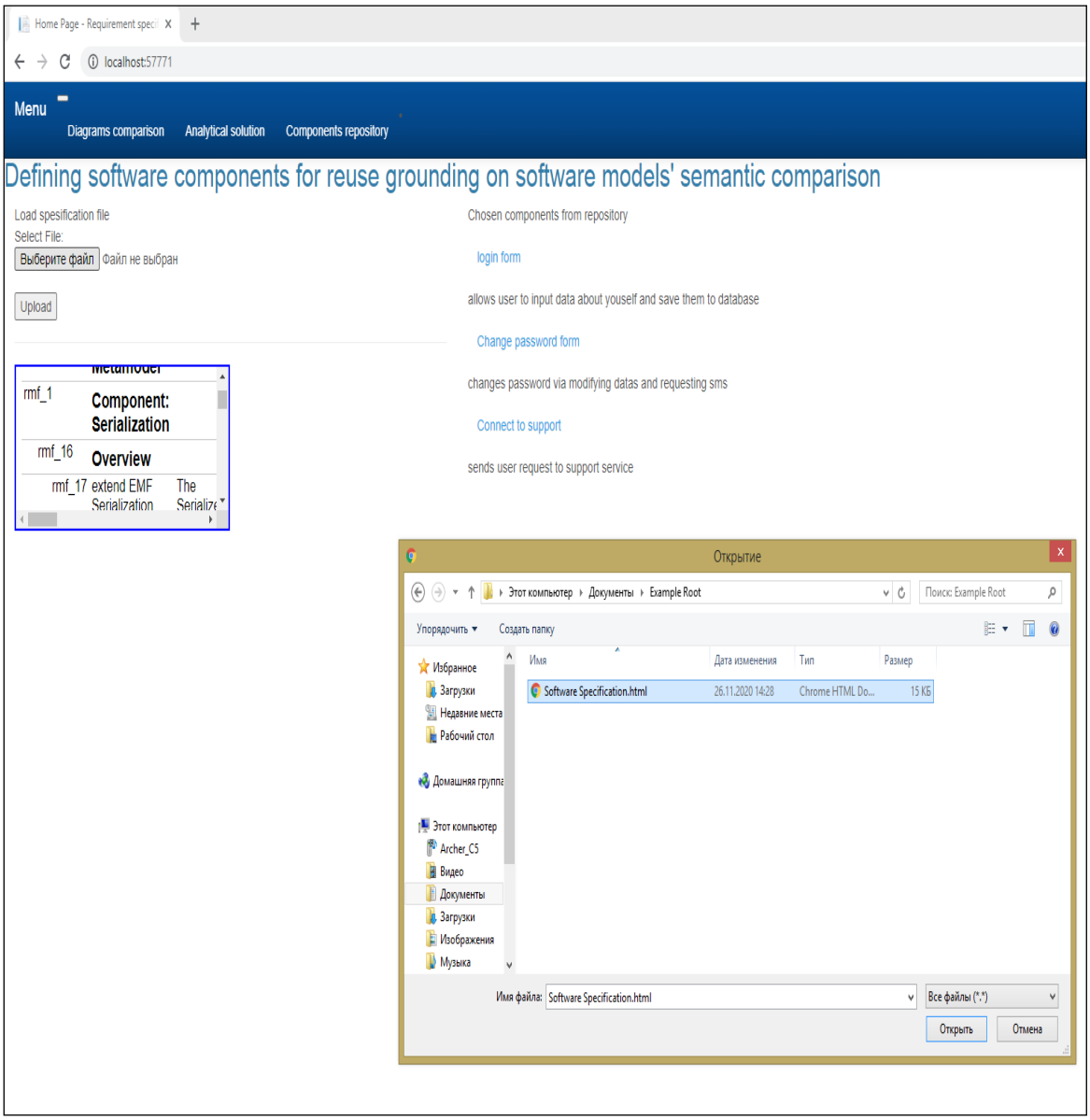


Figure 6 Demonstration of the tool work

The Components Repository shows the page of repository, from which certain components can be uploaded and compared with the initial scheme. In the current example, repository has its local host and is given as an instance. The implementation on a real example is given in the next section. The tools allows to perform connection to a repository by means of authorization and authentication, different paths can be used and edited by means of a software user.

Implementation of the approach and tool for solution of practical task

Requirement Analysis of Application

For the implementation of an approach and tool, an already existing application is used. It is the application created by means of Unity3D for the desktop and mobile usage. This application allows users to improve their real-life skills and control the progress in one or another activity which they perform. Overall, the application is made for people to measure their progress by means of real numbers, because there is always a problem of progress evaluation in real cases.

A user can choose, whether show results of previous work, or work upon another possible skill. The list of skills can be uploaded automatically and later on, the user can edit it. When a particular skill to improve is selected, the next window of working timer is opened. For a certain period of time, the application continues its work (it can also work minimized), and finally, when user decides to stop, the received result is written to memory as obtained experience in that subject. Next time, when the result page is invoked, the results achieved during the previous sessions are depicted as the progress bars, along with the numerical value, represented by them.

Functional system requirements for the sample project.

- F1 – user profile is loaded along with his/her name and list of possible activities;

- F2 – user should be able to choose activity for current session in main menu (Figure 7);
- F3 – user should be able to watch the experience received on the appropriate scene;
- F4 – user should be able to choose skills from various areas on the appropriate scene (Figure 7);
- F5 – choice of particular skill should lead to the scene with timer, where the time counting starts;
- F5.1 – user should be able to pause timer (Figure 8);
- F5.2 – user should be able to interrupt the work of a timer (Figure 8);
- F5.3 – user should be able to stop the work of the timer to see the results of work (Figure 8);
- F5.4 – user should be able to enter if some time was skipped during the work;
- F6 – the data of user profile should be serialized and stored by means of Json serialization;
- F7 – user profile should contain data about his/her name, possible areas of skills to improve and experience already gained during previous sessions;
- F8 – user should be able to update reference schemas to change the list of potential skills.

User interface prototypes

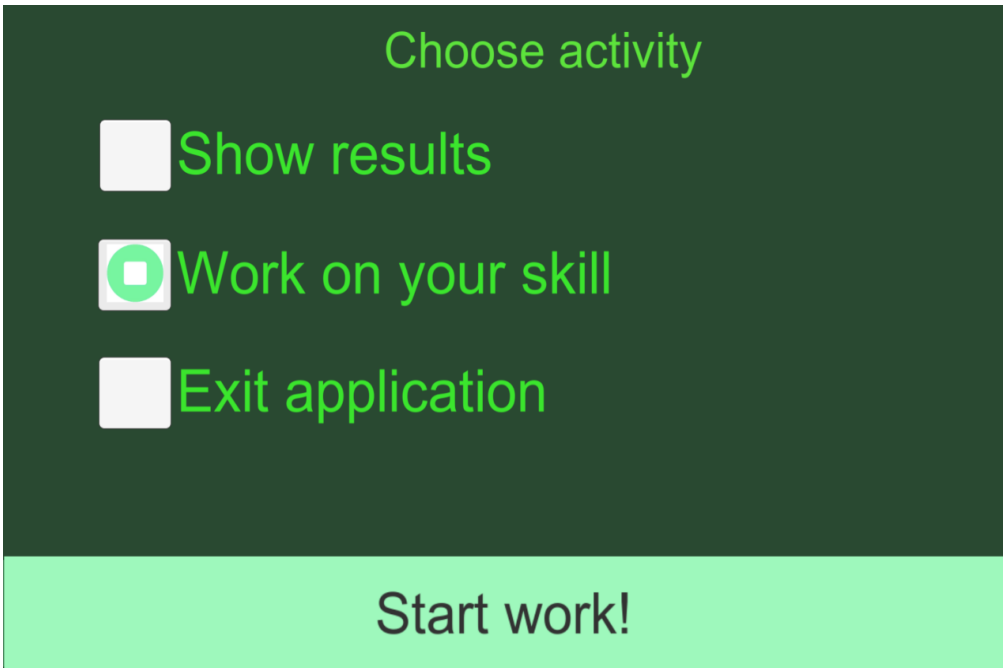


Figure 7 Starting screen of an application

Figure 8 Timer Game Scene

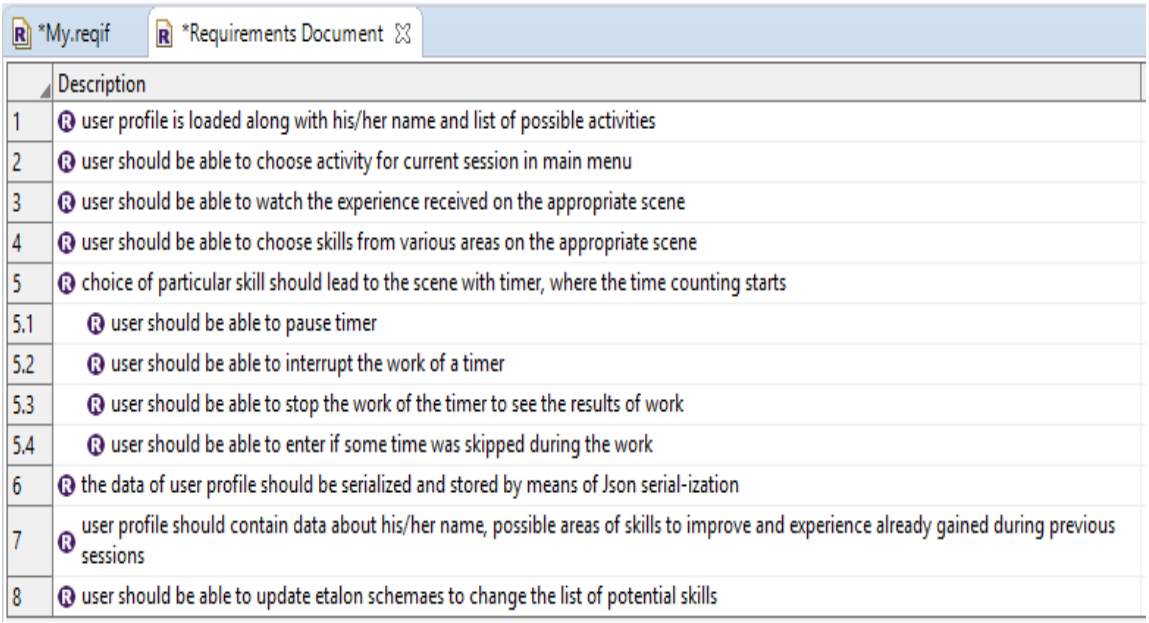
Development environment for the sample application is Unity3D. Unity3D is a modern cross-platform game and application development engine developed by Unity Technologies. Using this engine, the user can develop applications both for computers, mobile devices (for example, based on Android), game consoles and other devices. The engine is characterized by convenient testing possibilities, due to integration of game engine to the Unity development environment. It allows testing project during the development, not leaving the environment, so it is suitable for Test-Driven Development approach. In addition, Unity supports the import of a huge number of different formats, which allows the game developer to construct the models themselves in a more convenient application, and use Unity for its intended purpose – product development. Third, scripts are written in the most popular programming languages - C# and JavaScript.

Moreover, Unity 3D has a large and convenient component repository, which is called Unity Asset Store. It allows developers browse through the gygabytes of content to search for the asset, which is relevant to the topic of their projects. It contains both free to use and paid components, which can vary widely in terms of their use and their quality. The engine allows users to create their own assets and upload them to the store, increasing the quantity and quality of existing content.

Implementing the approach, which is described in this work, it is possible to deal with both types of components, but for example, the external parts are uploaded from the local repository. During the process of application design, it is important to divide the essential core parts of program from its component part, to be able to provide component-driven development, which is related to the use in Unity Engine. Component-Driven Development (CDD) is a development methodology that anchors the build process around components. It is a process that builds UIs from the "bottom up" by starting at the level of components and ending at the level of pages or screens.

In Unity3D, the application usually consists of scenes, which serve as the containers for various assets, which can be either downloaded, or created by a developer. Assets are some graphical parts, physics, materials, sounds and scripts. Scripts represent the functionality of project. They are attached to other objects in the scene; normally a container is present that contains a major amount of scripts. The sample application consists of 5 scenes and 15 scripts, which implement the different parts of its functionality.

The requirements above should be notated in ReqIF format, using either ReqIF Studio or RMF. The created tool is able to upload specification of this format to proceed to its analysis. Specifications of each component along with them themselves can be viewed at Repository window of an application. Figure 9 depicts the example of given requirements by means of RMF tool for Eclipse:


*My.reqif *Requirements Document

	Description
1	user profile is loaded along with his/her name and list of possible activities
2	user should be able to choose activity for current session in main menu
3	user should be able to watch the experience received on the appropriate scene
4	user should be able to choose skills from various areas on the appropriate scene
5	choice of particular skill should lead to the scene with timer, where the time counting starts
5.1	user should be able to pause timer
5.2	user should be able to interrupt the work of a timer
5.3	user should be able to stop the work of the timer to see the results of work
5.4	user should be able to enter if some time was skipped during the work
6	the data of user profile should be serialized and stored by means of Json serial-ization
7	user profile should contain data about his/her name, possible areas of skills to improve and experience already gained during previous sessions
8	user should be able to update etalon schemaes to change the list of potential skills

Figure 9. Requirements specification created by means of RMF

The sequence diagram for the sample project is viewed on the Figure 10:

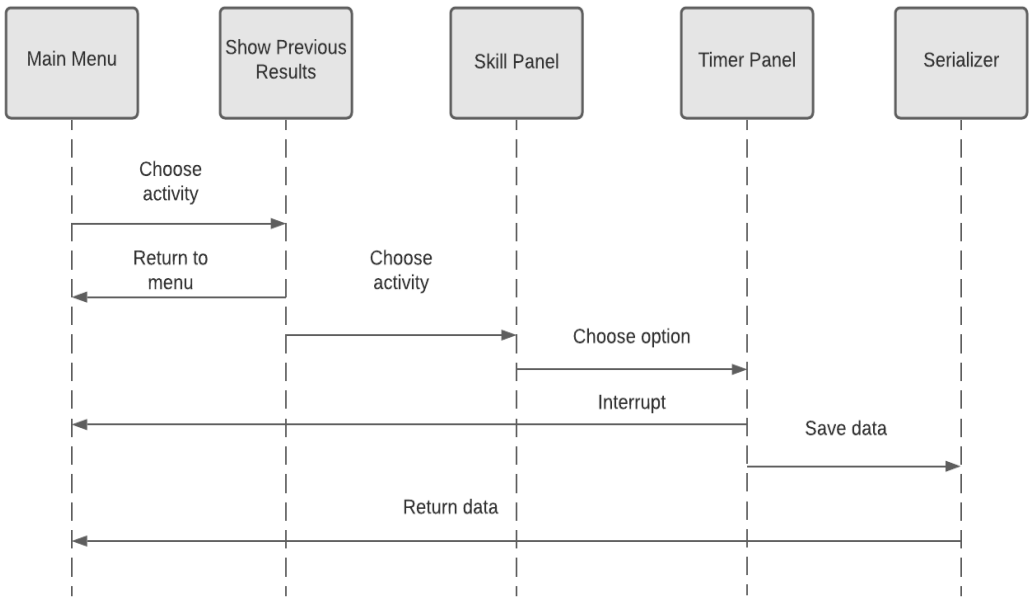


Figure 10. Sequence diagram of game scenario

Behavioural Software Models Analysis

Software specification and the behavioural diagram of current system is uploaded to the tool (Figure 9 and Figure 6.).

UML diagrams of project are decomposed into logical parts, which later on should be compared with the components formal representation. Thus, the structure of models are analysed. Specification is used for providing of semantic comparison of elements. In the current example, the scene with timer is considered (Figure 8). Data serialization should be provided, according to the functional requirements, so the data changes during the sessions should be stored to User profile and be available in the future sessions. Repository contains two components, which corresponds the needed structure, namely, they are Json and XMLSerializer.

Both components fulfil the need for saving data of the application, both can be recommended for the use in terms of concrete purpose (Requirement). According to metadata, which follows the mentioned components, keywords [noun] and keywords [verb] are checked manually to ensure the correspondence of elements. These keywords are compared to the requirements, which are given by specification, thus we see the relevance ~80% of both components to the functionality of the system.

However, requirements to the project specify serialization in Json format (F-6), so precisely the component of Json is recommended to be used for data save and load process in current situation. If the given requirement was not specified, both components would be recommended with ~80% relevance to subject

Figure 11 depicts the process of behavioral software model comparison tool, which allows us to understand the relevance of the project core to its component, which were either created locally, or downloaded externally. The data in rectangles represent objects; the numbers depict messages, which are sent by these objects to each other during the interaction.

In the Figure 12 communication diagram of developed project is represented. Different colors represent different parts of software, namely, different components, and the complete scheme shows interaction between them. For example, the scene with the timer is considered. Collaboration diagram is generated to view the flow of data between components of an application (in this case, the user scene of timer).

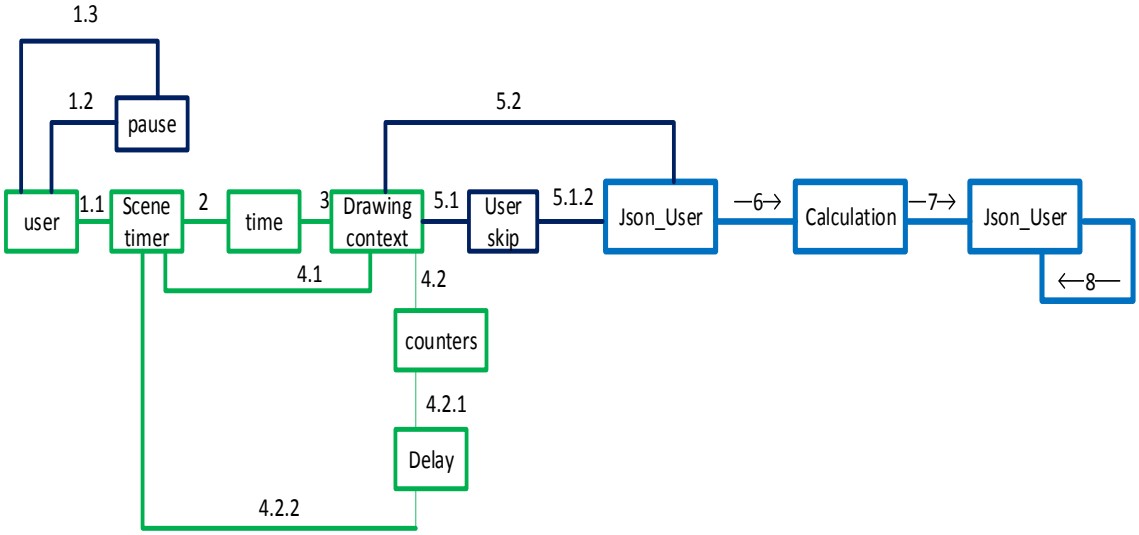


Fig. 11. Collaboration Diagram of the project in relevance to its components

The green part corresponds to the user/timer functionality, blue part represents serialization process, and darkblue parts represent the pause/resume mechanism.

The components were compared and the most relevant were chosen to provide more efficient functionality of the sample application. Structural analysis along with semantic comparison with metadata were performed to provide the most adequate result. It should be noted, that requirements specification for the core project and metadata, which is given with components should be correspondent to avoid semantic imprecision. The careful investigation of each component should be performed afterwards due to the possible need of its updating or rewriting some parts of functionality.

Table 2. Description of communication diagram elements (Figure 11)

Element	Description
User	The current user of the application
1.1	Proceed to Scene timer
1.2	Proceed to Pause
Pause	The application timer is paused
Scene timer	Starting the time flow
2	Time flows
Time	The timer is incremented
3	The message from timer increment is delivered to the drawing mechanism
Drawing context	Draws the increment of timer on the user screen
4.1	Returns to the initial scene to proceed the work of timer, if the panel is not filled
4.2	If the panel is filled, proceed to the counters
Counters	The number of shown time elapsed
4.2.1	The counters are reset
Delay	Provide the delay before forming of further graphic fill

4.2.2	Return to the initial timer scene to draw another timer
5.1	If some time was skipped, it allows user to subtract skipped time from the complete amount
5.2	If no time was skipped, proceed at opnce to the process of serialization
User skip	User enters the time that was skipped
5.1.2	Proceed to the process of serialization
Json_User	The data achieved in current session is serialized
6	Proceed to the Calculation process
Calculation	Contains results of the calculation
7	Proceed to the process of serialization
Json_User	The User profile is updated with serialized data.
8	Proceed to the process of serialization

The following chains can be formed from this part of behavioural software model (for instance):

ESG = {(user, 1.2, pause),(user, 1.1, Scenetimer),(Scenetimer, 2, time), (time, 3, Drawing context), (Drawing context, 4.2, counters), (Counters, 4.2.1, Delay), (Delay, 4.2.2, Scene timer), (Drawing context, 5.1, User skip), (Drawing context, 5.2, Json_User), (Json_User, 6, Calculation), (Calculation, 7, Json_User), (Json_User, 8, Json_User)}

Start = {(user, 1.2, pause), (user, 1.1, Scene timer)}

Finish = {(Json_User, 8, Json_User)}

Switch = {(time, 3, Drawing context)}

ESG = {(Scenetimer, 2, time), (Drawing context, 4.2, counters), (Counters, 4.2.1, Delay), (Delay, 4.2.2, Scene timer), (Drawing context, 5.1, User skip), (Drawing context, 5.2, Json_User), (Json_User, 6, Calculation), (Calculation, 7, Json_User)}

Conclusion

In this paper the approach for behavioural software models comparison by means of their formal representation was analysed and discussed. Several ways for storage of rule interchange formats were considered, the choice of the most effective method for data transition was grounded. The overview of environments, which can be used to work with software requiremtns, such as ReqIF Studio and RMF plugin for Eclipse is made.

The mathematical grounding of the software model comparison was given, and the way to decompose behavioural model to its formal representation was shown. Along with graph and set theory, the basics of reverse software engineering were examined, and the skills in working in the field of initial software product analysis were obtained.

The information received during the research led to the development of a software tool, which allows user to implement the given approach to formal representation of behavioural software models in relevance to the real projects and their component and thus, make conclusion about the correspondency of particular components to the core structure of project. The process is implemented by means of structural and semantic comparison of the models and software specifications.

During the work upon software tool, the acquaintance with Responsive Web Design, MVC architecture and ASP.NET web development features was got.

The choice of Bootstrap framework along with the approaches stated above was grounded and implemented. Due to the scalable and extensible structure of project, the further improvement and updating of a tool is possible, allowing to increase its abilities to work with various modelling environments and different data formats.

The considered approach and software tool can be used for purposes of software reuse, refinement and merging, decrease cost which is spent on the processes, allowing developers and analytics to estimate different assets and their relevance to the project structure beforehand. They may be used to increase efficiency of the analysis process during the planning stage in software development, and check for the various components' functionality and relevance by means of restoring them from the repository. It is also useful instrument to deal with potential risks during the software development life cycle.

Bibliography

(BootStrap, 2019) Bootstrap featureslist Access mode: https://www.sitesbay.com/bootstrap/bootstrap-features-of-bootstrap

(Chebanyuk, 2018) Chebanyuk O. An Approach of Text to Model Transformation of Software Models. In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018), 432-439

(Chebanyuk O. & Salem Abdel-Badeeh M., 2018) Chebanyuk O. & Salem Abdel-Badeeh M. (2018) Formal Foundation, Approach, and Smart Tool for Software Models' Comparison. Egyptian Computer Science Journal, Volume 42, Number 4. 89-102

(Chebanyuk O et. al., 2018) Chebanyuk O. & Dyshlevyy O. & Skalova V. An approach of obtaining initial information for behavioral software models processing. International journal "Informational Models and Analysis". Volume 7, Number 2, 2018, 25-41.

(Chebanyuk E & Shestakov K., 2017) Chebanyuk E. & Shestakov K. An Approach for Designing of Architectural Solutions Based on Software Model-To-Model Transformation. International journal "Informational Theories and Applications", Vol 24, Number 1, 2017, 60-84

(D'silva V. et al, 2008) D'silva, V., Kroening, D., & Weissenbacher, G. (2008). A survey of automated techniques for formal software verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(7), 1165-1178.

(Gordon, T. F. et. al., 2009) Gordon, T. F., Governatori, G., & Rotolo, A.. Rules and norms: Requirements for rule interchange languages in the legal domain. In International Workshop on Rules and Rule Markup Languages for the Semantic Web Springer, Berlin, Heidelberg. pp. 282-296.

(Bolye H., et. al., 2007) Boley, H., Kifer, M., Pătrânjan, P. L., & Polleres, A. (2007, September). Rule interchange on the web. In Reasoning Web International Summer School (pp. 269-309). Springer, Berlin, Heidelberg.

(ECL, 2015) The Epsilon Comparison Language (ECL) Access mode: https://www.eclipse.org/epsilon/doc/ecl/

(Hummel, O. et. al., 2004) Hummel, O., & Atkinson, C. Extreme harvesting: Test driven discovery and reuse of software components. In Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004. (pp. 66-72). IEEE.

(Lee, J. et. al., 2003). Lee, J., Kim, J., & Shin, G. S. Facilitating reuse of software components using repository technology. In Tenth Asia-Pacific Software Engineering Conference, 2003. (pp. 136-142). IEEE.

(OWL, 2018) Web Ontology Language (OWL) Assess mode: https://www.w3.org/OWL/

(ReqIF, 2013) Requirements Interchange Format (ReqIF) access mode: https://www.omg.org/spec/ReqIF/1.1/PDF

(RIF, 2005) [1] RIF Working group Access Mode:

https://www.w3.org/2005/rules/wiki/RIF_Working_Group2

(RDF, 2018) Resource Description Framework (RDF) Access Mode:https://www.w3.org/RDF/

(SPARQL, 2018) SPARQL Query Language for RDF Access Mode: https://www.w3.org/TR/rdf-sparql-query/

(Tiwari, U. K., & Kumar, S., 2020) Tiwari, U. K., & Kumar, S. Component-Based Software Engineering: Methods and Metrics. CRC Press., 2020

(XML, 2016) Extensible Markup Language (XML) Assess mode: https://www.w3.org/XML/

(W3C, 2013) RIF Framework for Logic Dialects (Second Edition) Assess mode: https://www.w3.org/TR/2013/REC-rif-fld-20130205/#ref-newsymbol

(Gerardo Canfora and Massimiliano Di Penta) New Frontiers of Reverse Engineering. June 2007, University of Sannio, Benevento, Italy

(RMF, 2011) Requirements Modeling Framework (RMF) Access mode: https://www.eclipse.org/proposals/modeling.mdt.rmf/

Authors' Information

Olena Chebanyuk *– D.Sc., Professor of Software Engineering Department, National Aviation University, Kyiv, Ukraine, e-mail: chebanyuk.elena@ithea.org (chebanyuk.elena@gmail.com)*

Major Fields of Scientific Research: *Domain Engineering, Software Engineering Foundation, Model-Driven Development, Software architecture, Mobile development, Software development*

Mykyta Krainii – *Assistant of Software Department in National Aviation University, Kyiv, Ukraine, email (nickkhryne@gmail.com)*

Major Fields of Scientific Research: Software Development, Mobile Development, Domain Engineering