COMPUTER-AIDED ENGINEERING

СИСТЕМЫ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ

УДК 004.89

А.М. Наместников, Г.Ю. Гуськов, А.А. Филиппов

ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ПРОЕКТОВ ПРОГРАММНЫХ СИСТЕМ НА ОСНОВЕ ОНТОЛОГИЧЕСКОГО ПОДХОДА¹

Наместников Алексей Михайлович, доктор технических наук, доцент, окончил радиотехнический факультет Ульяновского государственного технического университета. Профессор кафедры «Информационные системы» УлГТУ. Имеет более 80 работ в области автоматизированного проектирования и интеллектуальных систем. [e-mail: nam@ulstu.ru].

Гуськов Глеб Юрьевич, кандидат технических наук, окончил факультет информационных систем и технологий УлГТУ. Доцент кафедры «Информационные системы» УлГТУ. Имеет работы в области онтологического моделирования и интеллектуального анализа временных рядов. [e-mail: g.guskov@ulstu.ru].

Филиппов Алексей Александрович, кандидат технических наук, окончил факультет информационных систем и технологий УлГТУ, доцент кафедры «Информационные системы» УлГТУ. Имеет статьи в области интеллектуальных систем хранения и обработки информации. [e-mail: al.filippov@ulstu.ru].

Аннотация

В данной статье представлены новая эффективная модель, алгоритмы и методики представления предметной области разработки программных систем (ПС) в форме фрагментов базы знаний системы поддержки процесса проектирования. База знаний формируется в процессе анализа диаграмм классов в нотации UML. Предложенные подходы позволяют сократить время процесса проектирования и увеличить качество полученной ПС за счет использования удачных проектных решений, использованных в других проектах. Поиск удачных проектных решений осуществляется с применением разработанных метрик определения схожести проектов ПС на основе вычисления степени выраженности шаблонов проектирования в исходном коде проекта.

Ключевые слова: база знаний, программная система, UML, диаграмма классов, проектное решение.

doi: 10.35752/1991-2927-2020-1-5-75-85

INTELLIGENT ANALYSIS OF SOFTWARE SYSTEMS PROJECTS BASED ON THE ONTOLOGICAL APPROACH

Aleksei Mikhailovich Namestnikov, Doctor of Sciences in Engineering, Associate Professor; graduated from the Radioengineering Faculty of Ulyanovsk State Technical University; Professor of the Department of Information Systems of UlSTU; an author of more than 80 scientific papers in the field of computer-aided design and intelligence systems. e-mail: nam@ulstu.ru.

Gleb Iurevich Guskov, Candidate of Sciences in Engineering; graduated from the Faculty of Information Systems and Technologies of UlSTU; Associate Professor of the Department of Information Systems of UlSTU; an author of scientific papers in the field of ontology modeling and time-series data mining. e-mail: g.guskov@ulstu.ru.

Aleksei Aleksandrovich Filippov, Candidate of Sciences in Engineering; graduated from the Faculty of Information Systems and Technologies of UlSTU; Associate Professor of the Department of Information Systems of UlSTU; an author of articles in the field of intelligence systems for data storage and processing. e-mail: al.filippov@ulstu.ru.

Abstract

This article presents a new effective model, algorithms, and methods for representing the subject area of the process of developing a software systems. The subject area is presented in the form of fragments of the knowledge base of the design process support system. The knowledge base is formed in the process of analyzing class diagrams in UML notation. The proposed approaches can reduce the time of the design process and increase the quality of the obtained software system through the use of successful design solutions used in other projects. Search for successful design solutions is carried out using the developed metrics for determining the similarity of software systems projects. The metric allows calculating the design pattern match in the source code of the project.

Key words: knowledge base, software system, UML, class diagram, project solution.

ВВЕДЕНИЕ

Современный подход к разработке программных систем (ПС) предполагает использование интеллектуальных автоматизированных средств поддержки процесса проектирования, позволяющих выполнять поиск и повторное использование удачных архитектурных решений на основе общего семантического представления предметных и проектных знаний. Знания в подобных автоматизированных средствах в настоящее время, как правило, представляются в форме онтологий.

Разработка онтологий – долгий и ресурсоемкий процесс, требующий привлечения специалистов, обладающих компетенциями в онтологическом инжиниринге, разработке программного обеспечения и следующих предметных областях: программная инженерия и радио- и схемотехника. Поиск специалиста с таким набором компетенций представляет собой сложную, довольно часто не имеющую решения задачу.

Разработчики ПС, как правило, не имеют достаточных знаний о предметной области, в которой решаются задачи разработки и автоматизации. Подобная проблема появляется из-за того, что разработчики могут переходить с проекта на проект и менять место работы. В документах, регламентирующих предметную область, не всегда зафиксированы все принятые семантические значения сущностей и отношений. Создание базы знаний, позволяющей учитывать и повторно использовать удачные проектные решения, позволит сократить время проектирования и разработки, а также количество смысловых ошибок. Основной проблемой использования онтологий в процессе разработки ПС являются высокие требования к знаниям внутреннего устройства онтологий и их возможностей, предъявляемые к разработчикам ПС.

Важность формализации концептов проблемной области (ПрО) для разработки ПС привела к появлению специальных языков проектирования, которые включа-

ют в себя формализацию концептов (сущностей) ПрО. Самые распространенные средства проектирования основываются на языке UML (Unified Modeling Language) [1].

Средства, использующие язык UML для автоматизации проектирования, позволяют построить проект создаваемой ПС, состоящий из концептуальных элементов. Диаграммы на языке UML применимы к описанию конкретных особенностей ПС, например:

1. Классов, составляющих архитектуру ПС;
2. Таблиц и связей, описывающих схему базы данных;
3. Свойств и характеристик АРМ-операторов и серверов для создания схемы физического размещения и т. д.

На данный момент сформировалось достаточно большое сообщество исследователей в области онтологического проектирования и создания прикладных систем, основанных на онтологиях. В работах Боргеста Н.М. [2] описываются различные методы и подходы к онтологическому проектированию. В работах Бениаминова Е.М. [3] рассмотрены существующие прикладные онтолого-ориентированные системы. Научная проблематика, посвященная описанию и дальнейшему применению знаний в виде онтологий, представлена несколькими обобщающими трудами, например, Гавриловой Т.А., Хорошевского В.Ф. [4], Лукашевич Н.В [5]. В работах Голенкова В.В [6] рассматривается технология OSTIS в разрезе различных проблем поиска оптимальных форм представления знаний. Fernando Bobillo и Umberto Straccia предложили формат описания нечеткой онтологии [7].

Также необходимо отметить работы следующих исследователей в области онтологического проектирования: Виттих В.А., Грибова В.В., Добров Б.В., Загорюлько Ю.А., Ландэ Д.В., Курейчик В.М, Соловьев В.Д., Смирнов С.В., Gruber T.R., Uschold M. и т. д. В работах перечисленных исследователей отмечается эффективность использования онтологического представления знаний в процессе проектирования, описываются попытки решения частных прикладных инженерных задач

и фундаментальные решения на основе расширения онтологического представления (ОП) знаний с помощью аппарата нечеткой логики.

На данном этапе развития промышленности создано большое количество ПС с открытым исходным кодом в различных предметных областях. Повторное использование модулей открытых ПС позволит существенно уменьшить затраты времени на разработку программного обеспечения.

В настоящее время сильное влияние особенностей ПрО, в рамках которой осуществляется разработка ПС, приводит к частому использованию методологии разработки на основе предметной области (Domain Driven Development – DDD). Данный метод основан на объектно-ориентированной парадигме программирования. При подобном подходе различные виды тестирования ПС позволяют выполнять функцию проверки качества исходного кода, но не корректность модели с точки зрения особенностей ПрО. Контроль ошибок такого рода остается на менеджерах проекта и ведущих разработчиках.

Формирование ОП модели ПС позволяет обнаружить ошибки восприятия особенностей ПрО.

ОП можно разделить на ОП предметной области и ОП проектной модели. Данное разделение удобно для разделения знаний и фиксации соответствия между знаниями о предметной области и знаниями, на которых построена проектная модель. Оба типа знаний фиксируются в онтологии в формате OWL (Ontology Web Language) [8] с заранее определенным *TBox*.

В качестве входных данных системы поддержки процесса проектирования выступают диаграммы классов в нотации UML в силу их большей связи с исходным кодом ПС.

Таким образом, для решения проблемы поддержки проектирования ПС необходимо формализовать:

1. Модель представления проектных решений в качестве содержимого базы знаний;
2. Метод онтологического индексирования диаграмм классов в нотации UML;
3. Методы поиска удачных проектных решений в содержимом базы знаний.

1 МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ПРОЕКТНЫХ РЕШЕНИЙ В КАЧЕСТВЕ СОДЕРЖИМОГО БАЗЫ ЗНАНИЙ

Для решения задачи интеллектуального анализа проектных диаграмм, входящих в состав проектной документации, необходимо обладать знаниями в области построения формализованных диаграмм (использования нотаций). Такие знания позволяют выполнять идентификацию применяемых в различных проектах шаблонов проектирования и, следовательно, находить проекты со схожими архитектурными решениями и подходами к реализации подсистем ПС.

В качестве основы базы знаний системы поддержки процесса проектирования планируется использовать систему онтологий в формате OWL. С учетом поддержки

на уровне стандартов консорциумом W3C группы языков OWL (OWL Light, OWL DL, OWL Full) будем в качестве логической основы языка описания онтологии рассматривать формализм *SHOINF(D)* [9–11].

В контексте дескрипционной логики *SHOINF(D)* онтология представляет собой базу знаний следующего вида [12]:

$$KB = \{TBox, ABox\};$$

где *TBox* – набор терминологических аксиом, представляющих общие знания о понятиях базы знаний и их взаимосвязях;

ABox – набор утверждений (фактов) об индивидах.

1.1 Терминология *Tbox*

Запишем терминологию проектных диаграмм, разделяя логическое представление нотации языка UML и логическое представление шаблонов проектирования.

$$Relationship \sqsubseteq \top$$

$$Dependency \sqsubseteq Relationship$$

$$Association \sqsubseteq Relationship$$

$$Generalization \sqsubseteq Relationship$$

$$Realization \sqsubseteq Relationship \sqcap \exists startWith.Class \sqcap \exists endWith.Interface,$$

где *startWith* и *endWith* – наименования ролей *исходум от* и *подходит к*, соответственно.

Представление основных классов запишем в виде:

$$Thing \sqsubseteq \top \forall hasAName.String$$

$$StructThing \sqsubseteq Thing$$

$$AnnotThing \sqsubseteq Thing$$

$$Note \sqsubseteq AnnotThing \sqcap$$

$$\sqcap \exists connectedTo.Association$$

$$Class \sqsubseteq StructThing$$

$$Object \sqsubseteq StructThing \sqcap \exists isObjectOf.Class$$

$$\sqcap \forall isObjectOf.Class$$

$$Interface \sqsubseteq StructThing$$

$$SimpleClass \sqsubseteq Class$$

$$AbstractClass \sqsubseteq Class,$$

где *hasAName* – наименование роли *имеет имя*;

String – конкретный домен (concrete domain) строкового типа.

isObjectOf – наименование роли *является объектом*;

connectedTo – наименование роли *связан с*.

Атрибуты и методы классов представляются следующим образом:

$$Attribute \sqsubseteq \top \sqcap \exists hasAAttrName.String$$

$$\sqcap \exists isAPartOf.Class$$

Рис. 1. Иерархия концептов разработанной онтологии в редакторе Protege

$Method \sqsubseteq \top \sqcap \exists \text{ hasAMethName.String}$
 $\sqcap \exists \text{ isAPartOf.Class},$

где *hasAAttrName* – наименование роли имеет наименование атрибута;

hasAMethName – наименование роли имеет наименование метода;

isAPartOf – наименование роли является частью.

Рассмотрим терминологию проектных шаблонов, связанных с логическим представлением нотации проектных диаграмм (на примере диаграммы классов языка UML):

$Template \sqsubseteq \top \sqcap \exists \text{ hasATempName.String}$
 $\sqcap \exists \text{ hasAExpValue.Double}$

$omeTemplate \sqsubseteq Template,$

где *hasATempName* – наименование роли имеет имя шаблона;

hasAExpValue – наименование роли имеет степень выраженности;

Double – конкретный домен (concrete domain) вещественного типа.

Отдельный шаблон проектирования в каждом конкретном проекте может иметь определенную степень выраженности.

Иерархия концептов разработанной онтологии представлена на рисунке 1:

Иерархия свойств (DataTypeProperty и ObjectProperty) разработанной онтологии представлена на рисунке 2.

1.2 Терминология ABox

На рисунке 3 приведен пример шаблона <<Делегирование>>, который в виде набора фактов ABox выглядит следующим образом:

Рис. 2. Иерархия DataTypeProperty и ObjectProperty разработанной онтологии в редакторе Protege

Рис. 3. Структура онтологии (включая пример шаблона проектирования)

class1 : SimpleClass
attribute1 : Attribute
method1 : Method
relation1 : Association
(method1,name1:String) : hasAMethName
(method2,name2:String) : hasAMethName
(attribute1,class1) : iaAPartOf
(object1,attribute1) : owl : sameAs
left(method2,class2) : iaAPartOf
(relation1,class1) : startWith

class2 : SimpleClass
object1 : Object
method2 : Method

(object1,class2) : isObjectOf
(method1,class1) : iaAPartOf

(relation1,class2) : endWith

Фактически, в онтологии базы знаний в набор фактов *ABox* включены все факты по используемым шаблонам проектирования. Далее в процессе индексирования происходит сопоставление фактов из *ABox* с фактами, извлекаемыми из проектных диаграмм, и определяется степень выраженности для каждого шаблона онтологии.

2 ОП ШАБЛОНОВ ПРОЕКТИРОВАНИЯ

Элементы шаблонов проектирования часто имеют одинаковые названия, это приводит к проблеме однозначной идентификации экземпляров концептов онтологии. Для решения данной проблемы необходимо ввести правило именования экземпляров онтологии. Название экземпляра онтологии начинается с названия шаблона проектирования и дополняется изначальным названием элемента. Для экземпляров онтологии, представляющих отношения в шаблонах проектирования, необходимо ввести аналогичное правило. Название такого экземпляра начинается с названия шаблона проектирования, далее записывается название элемента, из которого исходит отношение, и завершается названием экземпляра названием элемента, к которому направлено отношение.

Формально ОП шаблона проектирования можно представить следующим образом:

$$OV_i^{tmp} = \{C, D, R^{same_as}\},$$

где *C* – множество индивидов онтологии базы знаний;

D – множество отношений между элементами *i*-го шаблона проектирования, представленное индивидами онтологии базы знаний;

R – множество отношений эквивалентности индивидов онтологии базы знаний.

Рассмотрим описание шаблона проектирования *Builder* в формате разработанной онтологии. Шаблон проектирования *Builder* является одним из наиболее часто используемых в промышленной разработке программного обеспечения шаблонов. Шаблон проектирования *Builder* представляет собой порождающий шаблон проектирования и позволяет создавать сложные составные объекты. На рисунке 4 изображена UML-диаграмма классов шаблона проектирования *Builder*, выполненная в среде визуального проектирования Visual Paradigm.

Для представления шаблона проектирования *Builder* в виде фрагмента онтологии базы знаний необходимо определить следующие экземпляры концептов:

Builder.Client : SimpleClass

Builder.Director : SimpleClass

Рис. 4. Диаграмма классов шаблона проектирования Builder

Рис. 5. Иллюстративный пример ОП шаблона проектирования *Builder*

Builder.ConcreteBuilder : *SimpleClass*

Builder.Product : *SimpleClass*

Builder.AbstractBuilder : *AbstractClass*

Builder.Client_AbstractBuilder : *Association*

Builder.Client_Director : *Association*

Builder.Client_IProduct : *Association*

Builder.ConcreteBuilder_Product

: *Association*

Builder.ConcreteBuilder_AbstractBuilder

: *Generalization*

Builder.Product_IProduct : *Realization*

Указанный выше фрагмент онтологии представлен на рисунке 5.

Для осуществления функции поиска в базе знаний системы поддержки процесса проектирования схожих проектных решений необходимо использовать метрику для вычисления подобия между проектируемыми и уже реализованными программными проектами.

3 ОПРЕДЕЛЕНИЕ СТЕПЕНИ ВЫРАЖЕННОСТИ ШАБЛОНА ПРОЕКТИРОВАНИЯ

Для вычисления меры архитектурного подобия программных проектов предложен следующий метод вычисления степени выраженности шаблона проектирования в программном проекте:

$$\mu^{prj}(tmp_i) = \frac{|C^{prj} \cap C^{tmp_i}| + |R^{prj} \cap R^{tmp_i}|}{|C^{tmp_i}| + |R^{tmp_i}|}, \quad (1)$$

где $|C^{prj} \cap C^{tmp_i}|$ – количество совпадающих индивидуальных в ОП i -го шаблона проектирования базы знаний и ОП некоторого проекта;

$|R^{prj} \cap R^{tmp_i}|$ – количество совпадающих отношений в ОП i -го шаблона проектирования базы знаний и ОП некоторого проекта;

$|C^{tmp_i}|$ – количество индивидуальных в ОП i -го шаблона проектирования базы знаний;

$|R^{tmp_i}|$ – количество отношений в ОП i -го шаблона проектирования базы знаний.

Если количество фактов $(|C^{tmp_i}| \cup |R^{tmp_i}|)$ ОП i -го шаблона проектирования tmp_i определяется достаточно просто: суммированием количества фактов, то для вычисления количества фактов $(|C^{prj}| \cup |R^{prj}|)$ в ОП проекта необходимо использовать следующий разработанный алгоритм:

Шаг 1. Преобразование диаграммы классов в нотации UML j -го проекта prj_j в набор фактов $ABox^{prj}$:

$elem_k^{prj} : Concert$

$(elem_k^{prj}, elem_s^{prj}) : Role,$

где *Concept* – концепт онтологии базы знаний, определенный в *TBox*;

Role – роль, определенная в *TBox*;

$elem_k^{prj}$ – *k*-й индивидуал концепта онтологии, извлеченный из проектной диаграммы.

Шаг 2. Определение набора базовых классов из $ABox^{prj}$ относительно *i*-го шаблона проектирования tmp_i .

Базовым классом будем называть такой индивидуал $elem_k^{prj}$ концепта *Class* (или его дочернего концепта *Subclass*) из $ABox^{prj}$, который соответствует некоторому индивидуалу $elem_k^{tmp} : Class$ из $ABox^{tmp_i}$ и для которого в шаблоне tmp_i имеет место максимальное количество фактов вида:

$elem_k^{prj} : Concert$

$(elem_k^{prj}, *) : Role(*, elem_k^{prj}) : Role.$

Шаг 3. Вычисление количества истинных фактов, выполняя попарную замену индивидуалов классов *i*-го шаблона tmp_i и диаграммы классов *j*-го проекта prj_j :

$\forall k : elem_k^{tmp} \leftrightarrow elem_k^{prj}.$

Факт шаблона проектирования является истинным относительно диаграммы классов, если удастся найти ему соответствие в наборе фактов диаграммы классов с учетом того, что различные имена индивидуалов концептов обозначают различные сущности. Различные имена экземпляров концептов относятся к одному и тому же индивидуалу онтологии только в том случае, если явно данные индивидуалы связаны отношением «owl:sameAs».

Указанные шаги алгоритма индексирования диаграммы классов выполняются для каждого шаблона проектирования, доступного в онтологии базы знаний.

После вычисления меры выраженности каждого отобранного шаблона проектирования в каждом из рассматриваемых программных продуктов, становится возможным вычисление меры подобию между программными проектами по одной из трех метрик.

4 МЕТРИКИ МЕРЫ АРХИТЕКТУРНОГО ПОДОБИЯ ПРОЕКТОВ ПС

Первая метрика позволяет рассчитать меру подобию с помощью выделения наиболее выраженного шаблона проектирования в каждом из проектов:

$$\mu^1(prj_i, prj_j) = \bigvee_{tmp_k \in (prj_i \cap prj_j)} \mu^{prj}(tmp_k), \quad (2)$$

где prj_i, prj_j – ОП диаграммы классов в нотации UML *i*-го и *j*-го проектов соответственно;

$\mu^{prj}(tmp_k)$ – степень выраженности *k*-го шаблона проектирования в проекте (выражение 1).

Подобная метрика показывает хорошие результаты для сравнительно небольшого количества сложных

комбинированных шаблонов проектирования. Подобные шаблоны проектирования основываются на предметной области и, в меньшей мере, соответствуют шаблонам проектирования в привычном понимании промышленного программирования.

Вторая метрика расширяет первую (выражение 2) и учитывает меру выраженности шаблонов проектирования, превышающую определенное пороговое значение. Экспериментальным путем было подобрано пороговое значение, равное 0,3. В случае, если мера выраженности шаблона проектирования меньше значения 0,3, можно сделать вывод об отсутствии в программном проекте данного шаблона проектирования, и, как следствие, такой шаблон проектирования необходимо исключить из рассмотрения:

$$\mu^2(prj_i, prj_j) = \frac{\bigvee_{tmp_k \in (prj_i \cap prj_j) \geq 0,3} \mu^{prj}(tmp_k)}{N}, \quad (3)$$

где *N* – число шаблонов проектирования со степенью выраженности, значение которой больше или равно 0,3 для каждого из проектов.

Третья метрика схожа со второй метрикой (выражение 3), но накладывает дополнительное условие на учет вклада меры выраженности шаблона проектирования ($\tilde{\mu}^{prj}$) в меру архитектурного подобию проектов:

$$\mu^3(prj_i, prj_j) = \frac{\bigvee_{tmp_k \in (prj_i \cap prj_j) \geq 0,3} \tilde{\mu}^{prj}(tmp_k)}{N}, \quad (4)$$

где $\tilde{\mu}^{prj}(tmp_k)$ – взвешенная мера выраженности *k*-го шаблона tmp_k проектирования в программном проекте prj_j .

Под взвешенной мерой выраженности $\tilde{\mu}^{prj}(tmp_k)$ *k*-го шаблона проектирования tmp_k в проекте ПС prj_j будем понимать меру выраженности (выражение 1), нормализованную по числу элементов, входящих в состав шаблона проектирования с максимальным набором элементов:

$$\tilde{\mu}^{prj}(tmp_k) = \frac{|C^{prj} \cap C^{tmp_i}| + |R^{prj} \cap R^{tmp_i}|}{\bigvee_{tmp_k \in ABox} (|C^{tmp_k}| + |R^{tmp_k}|)}. \quad (5)$$

Данная модификация позволяет учитывать сложность внутреннего строения шаблона проектирования при вычислениях меры подобию программных проектов.

При использовании метрики $\mu^3(prj_i, prj_j)$ шаблон проектирования, состоящий из 20 элементов, полностью выраженный в проектах prj_i и prj_j , будет обладать в 4 раза большим весом, чем шаблон проектирования, состоящий из 5 элементов и также имеющий невзвешенную степень выраженности, равную 1 в проектах prj_i и prj_j .

5 АНАЛИЗ ЭЛЕМЕНТОВ ДИАГРАММ КЛАССОВ И ИХ ИСПОЛЬЗОВАНИЯ

В рамках данного исследования был проведен эксперимент по частоте использования различных сущно-

стей диаграммы классов в нотации UML. Эксперимент проводился на 30 проектах, загруженных из открытого репозитория исходного кода GitHub [13].

Задачей данного эксперимента стало выявление наиболее часто используемых разработчиками элементов проектных диаграмм. Гистограмма частотности использования представлена на рисунке 6.

Рис. 6. Гистограмма частотности использования элементов в проектах

Из-за отсутствия диаграмм классов в нотации UML в большинстве открытых репозиториях GitHub была разработана программа, анализирующая структуру классов уже реализованных ПС на языке Java и формирующая ОП проекта.

Язык Java был выбран, так как данный язык программирования предполагает строгий объектно-ориентированный стиль разработки и построения архитектуры ПС, а также существует библиотека `javaparser` [14], позволяющая формировать абстрактное синтаксическое дерево проекта.

Проекты были отобраны по следующим принципам:

- разнообразие проектов в выборке по размеру (количеству ветвлений и коммитов);
- разнообразие проектов в выборке по популярности (рейтинг проектов в рамках GitHub);
- разнообразие проектов по предметной области (базы данных, web-приложения, библиотеки и т. д.).

Перечень проектов, которые были проанализированы в рамках эксперимента: `javaparser/javaparser`, `Tencent/RapidView`, `Nightonke/GithubWidget`, `androidstarters/androidstarter`, `NikoYuwono/ToolbarPanel`, `dbachelder/CreditCardEntry`, `DesignativeDave/androrat`, `anupcowkur/Android-Wheel-Menu`, `xbmcremote`, `nekocode/CameraFilter`, `libgdx/packr`, `krasa/EclipseCodeFormatter`, `good-life/PushTalk`, `wasabeef/Takt`, `boxme/SquareCamera`, `ditclear/TimeLine`, `freezy/android-palantir/docker-compose-rule`, `yannecer/NCalendar`, `inaka/TinyTask`, `allegro/grunt-maven-plugin`, `pulse00/Symfony-2-Eclipse-Plugin`, `jhsain/`

`learnrxjava`, `apache/tomcat80`, `dnmilne/wikipediaminer`, `alexandrius/accordion-swipe-layout`.

При проведении эксперимента использовались самые последние версии проектов.

Набор наиболее часто используемых элементов нотации UML повлиял на формирование *TBox* онтологии базы знаний. Косвенным результатом данного эксперимента стала идея о недостаточной семантической силе отдельных элементов UML и переход к рассмотрению шаблонов проектирования.

6 ПОИСК ШАБЛОНОВ ПРОЕКТИРОВАНИЯ В ПРОЕКТАХ ПУБЛИЧНОГО РЕПОЗИТОРИЯ GitHub

Для проверки предлагаемого подхода к выделению шаблонов проектирования в проектах был проведен эксперимент, целью которого был поиск шаблонов проектирования в проектах, расположенных в репозитории GitHub. Для проведения эксперимента в онтологию была добавлена информация по 10 шаблонам проектирования: *Delegation*, *Interface*, *Abstract superclass*, *Builder*, *Adaptor*, *Singleton*, *Bridge*, *Facade*, *Decorator*, *Observer*.

Выбранные шаблоны проектирования отличаются по количеству элементов и связей между ними. Количество элементов варьируется от 3 до 20.

В данном эксперименте также рассматривались только проекты, разработанные с применением языка программирования Java.

Так как общее число проектов в репозитории GitHub, разработанных на языке Java, очень велико, то необходимо ограничить выборку проектов для эксперимента. В качестве ограничения будем использовать текстовый поиск, реализованный в GitHub.

В результате запроса `<<vk api>>` было получено 108 проектов, относящихся к множеству проектов, работающих с API социальной сети `<<ВКонтакте>>`.

Результатом запроса `<<design patterns>>` была выборка из 6516 проектов. Данная выборка необходима для проверки работы системы в условиях повышенного содержания шаблонов проектирования в проектах.

Для проведения тестирования выборка была ограничена первыми 100 проектами по обоим запросам. Результаты поиска шаблонов проектирования представлены в виде гистограмм на рисунках 7 и 8.

Рис. 7. Результаты поиска шаблонов среди проектов, полученных по запросу `<<vk api>>`

Рис. 8. Результаты поиска шаблонов среди проектов, полученных по запросу <<design patterns>>

В итоге были получены следующие результаты: высокая частота использования шаблонов *Delegation*, *Interface*, *Abstract superclass* и *Facade*. Данные результаты объясняются простой структурой данных шаблонов – сравнительно небольшое число структурных элементов и, как следствие, связей. Данные шаблоны проектирования могли применяться разработчиками неосознанно, или же они могут совпадать по структуре с частью более сложного шаблона.

Шаблонов проектирования *Builder*, *Adapter*, *Bridge*, *Decorator* и *Observer* в контрольной группе проектов нашлось относительно немного. Редкость этих шаблонов объясняется их сложной структурой – содержанием большого количества элементов.

7 РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ ПОИСКА СТРУКТУРНО СХОЖИХ ПРОГРАММНЫХ ПРОЕКТОВ

Для определения меры структурного сходства двух проектов, необходимо вычислить меру выраженности каждого шаблона проектирования в обоих проектах (выражения 1 и 5).

Мера выраженности шаблона проектирования в проекте вычисляется с помощью наложения *ABox* ОП проекта на *ABox* ОП шаблона проектирования.

В данном эксперименте все проекты были загружены из открытого репозитория GitHub. Все проекты были

отобраны по следующим ключевым словам: <<public API>>, <<social network>>, <<vkontakte>>, позволяющим определить принадлежность проектов к предметной области – работа с API социальной сети <<ВКонтакте>>: *Android-MVP*, *cordova-social-vk*, *cvk*, *DroidFM*, *VK-Small-API*, *VKontakteAPI*, *VK_TEST*.

В таблице 1 представлена степень выраженности каждого рассматриваемого шаблона проектирования во всех проектах экспериментальной выборки. Оценки меры подобия нормированы от 0 до 1.

Оценки меры сходства по первой метрике (выражение 2) всегда равны 1, потому что данная метрика отбирает шаблон проектирования с максимальной мерой выраженности для каждого из двух сравниваемых проектов. А так как, например, шаблон проектирования *Abstract superclass*, *Interface* и *Delegator* состоят из относительно малого количества элементов, это приводит к высокой мере выраженности таких шаблонов в большом количестве проектов.

Оценки меры схожести для проектов с использованием второй (выражение 3) и третьей (выражение 4) метрик представлены в таблицах 2 и 3 соответственно.

Степени похожести проектов, полученные в данных экспериментах, имеют весьма высокие значения, что можно объяснить двумя особенностями данного эксперимента.

Шаблоны проектирования с мерой выраженности менее 0,3 хотя бы в одном из сравниваемых проектов исключались. В рамках данного эксперимента предполагается, что шаблон проектирования, имеющий значение меры выраженности менее 0,3, не обнаружен в проекте. Если учитывать шаблоны проектирования с малой мерой выраженности, то при увеличении числа шаблонов будет существенно уменьшаться значение меры схожести любых проектов.

Проекты изначально выбирались по ключевым словам из одной предметной области, чтобы увеличить вероятность нахождения проектов со схожей структурой. Предполагается, что разработанная система будет использоваться в рамках крупного предприятия, специализирующегося на конкретной предметной области.

Таблица 1

Степени выраженности шаблонов проектирования в проектах

Project name / Design pattern name	Delegator (3)	Adapter (8)	Builder (12)	Abstract superclass (3)	Interface (5)
<i>Android-MVP</i>	1,0	0,875	0,83	1,0	1,0
<i>cordova-social-vk</i>	1,0	0,875	0,83	1,0	0,8
<i>cvk</i>	1,0	0,875	0,92	1,0	1,0
<i>DroidFM</i>	1,0	0,875	0,92	1,0	1,0
<i>VK-Small-API</i>	1,0	0,625	0,42	0,33	0,6
<i>VKontakteAPI</i>	1,0	0,875	0,83	1,0	0,8
<i>VK_TEST</i>	1,0	0,75	0,58	0,66	0,6

Таблица 2

Меры структурного подобия программных продуктов по второй метрике

<i>Project 1 / Project 2</i>	<i>Android-MVP</i>	<i>cordova-social-vk</i>	<i>cvk</i>	<i>DroidFM</i>	<i>VK-Small-API</i>	<i>VkontakteAPI</i>	<i>VK_TEST</i>
<i>Android-MVP</i>	-	0,96	0,96	0,98	0,78	0,96	0,78
<i>cordova-social-vk</i>	0,96	-	1	0,94	0,85	1	0,83
<i>cvk</i>	0,96	1	-	0,94	0,85	1	0,83
<i>DroidFM</i>	0,98	0,94	0,94	-	0,78	0,94	0,78
<i>VK-Small-API</i>	0,78	0,85	0,85	0,78	-	0,85	0,96
<i>VkontakteAPI</i>	0,96	1	1	0,94	0,85	-	0,83
<i>VK_TEST</i>	0,79	0,83	0,83	0,78	0,95	0,83	-

Таблица 3

Меры структурного подобия программных продуктов по третьей метрике

<i>Project 1 / Project 2</i>	<i>android-mvp</i>	<i>cordova-social-vk</i>	<i>cvk</i>	<i>DroidFM</i>	<i>VK-Small-API</i>	<i>VkontakteAPI</i>	<i>VK_TEST</i>
<i>android-mvp</i>	-	0,96	0,96	0,96	0,64	0,96	0,77
<i>cordova-social-vk</i>	0,96	-	0,99	0,93	0,67	0,99	0,80
<i>cvk</i>	0,97	0,99	-	0,93	0,67	0,99	0,80
<i>DroidFM</i>	0,97	0,93	0,93	-	0,61	0,93	0,74
<i>VK-Small-API</i>	0,64	0,67	0,68	0,61	-	0,67	0,87
<i>VkontakteAPI</i>	0,97	0,99	0,99	0,93	0,67	-	0,80
<i>VK_TEST</i>	0,77	0,80	0,80	0,74	0,87	0,80	-

ЗАКЛЮЧЕНИЕ

В ходе данного исследования получены следующие результаты:

1. Онтологически-ориентированная модель языка диаграмм UML и онтологическая модель шаблона проектирования.

2. Меры архитектурного подобия программных проектов; меры выраженности шаблона проектирования в рассматриваемых программных продуктах.

3. Алгоритм трансформации диаграммы классов в нотации UML в онтологию формата OWL.

4. Алгоритм трансформации исходного кода на языке программирования Java в онтологию формата OWL.

Были проведены вычислительные эксперименты, подтверждающие эффективность формирования базы знаний системы поддержки процесса проектирования на основе извлечения знаний о предметной области и проекте из диаграмм классов в нотации UML.

Таким образом, предложенный подход к поддержке процесса проектирования позволяет использовать удачные проектные решения при разработке новых ПС, тем самым сокращая время процесса проектирования и увеличивая качество полученных решений.

СПИСОК ЛИТЕРАТУРЫ

1. Booch G., Rumbaugh J., Jacobson I. Unified Modeling Language User Guide. 2nd ed. Addison-Wesley Object Technology Series, 2005. 496 p.

2. Боргест Н.М. Границы онтологии проектирования // Онтология проектирования. Самара : Предприятие «Новая техника». 2017. № 1 (7). С. 7–33.

3. Бениаминов Е.М. Некоторые проблемы широкого внедрения онтологий в ИТ и направления их развития // Тр. Симпозиума «Онтологическое моделирование». М. : ИПИ РАН, 2008. С. 71–82.

4. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. СПб. : Питер, 2000.

5. Лукашевич Н.В. Тезаурусы в задачах информационного поиска. М. : Изд-во МГУ, 2011. 512 с.

6. Голенков В.В. Онтологическое проектирование интеллектуальных систем // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2017) : матер. VII Междунар. научн.-техн. конф. Минск : БГУИР, 2017. С. 37–56.

7. Bobillo F., Straccia U. Fuzzy Ontology Representation using OWL 2 // International Journal of Approximate Reasoning. October 2010. Vol. 52, no. 7. pp. 1073–1094.

8. OWL 2 Web Ontology Language. URL: <https://www.w3.org/TR/owl2-overview/> (дата обращения: 17.02.2020).

9. The Description Logic Handbook: Theory, Implementation, and Applications. / Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. Cambridge : Cambridge University Press, 2003. 563 p.

10. Bonatti P., Tettamanzi A. Some complexity results on fuzzy description logics // WILF 2003 International

Workshop on Fuzzy Logic and Applications. Berlin, SpringerVerlag, 2004, LNCS 2955, pp. 19–24.

11. Horrocks I., Patel-Schneider Peter F., Harmelen F. From SHIQand RDF to OWL: The making of a web ontology language // *Journal of WebSemantics*. 2003. Vol. 1, no. 1. pp. 7–26.

12. Description logic programs: Combining logic programs with description logics / B. Groszof, I. Horrocks, R. Volz, and S. Decker. // *In Proc. of WWW 2003*. Budapest, Hungary, May 2003. pp. 48–57.

13. GitHub is a development platform inspired by the way you work. URL:<https://github.com/> (дата обращения: 17.02.2020).

14. Our goal is to build a simple and lightweight set of tools to generate, analyze, and process Java code. URL: <https://javaparser.org/> (дата обращения: 17.02.2020).

REFERENCES

1. Booch G., Rumbaugh J., Jacobson I. *Unified Modeling Language User Guide. 2nd ed. Addison-Wesley Object Technology Series*. 2005. 496 p.

2. Borgest N.M. Granitsy ontologii proektirovaniia [Boundaries of the Ontology of Designing]. *Ontologiiia proektirovaniia* [Ontology of Designing], 2017, vol. 7, no. 1 (23), pp. 7–33.

3. Beniaminov E.M. Nekotorye problemy shirokogo vnedreniia ontologii v IT i napravleniia ikh razvitiia [Some Problems of a Wide Application of Ontology in IT and their Development Areas]. *Tr. Simpoziuma "Ontologicheskoe modelirovanie"* [Proc. of Symposium on Ontology Modeling]. Moscow, IPI RAN Publ., 2008, pp. 71–82.

4. Gavriloa T.A., Khoroshevskii V.F. *Bazy znanii intellektualnykh sistem* [Knowledge Bases of Intelligence Systems]. St.Petersburg, Piter Publ., 2000.

5. Lukashevich N.V. *Tezaurusy v zadachakh informatsionnogo poiska* [Thesauri in Information Retrieval Problems]. Moscow, MGU Publ., 2011. 512 p.

6. Golenkov V.V. Ontologicheskoe proektirovanie intellektualnykh sistem [Ontological Designing the Intelligence Systems]. *Otkrytye semanticheskie tekhnologii proektirovaniia intellektualnykh sistem (OSTIS-2017). Mater. VII Mezhdunar. nauchn.-tekhn. konf.* [Proc. of the 7th Int. Sci.-Tech. Conf. Open Semantic Technologies for Intelligence System Designing]. Minsk, BGUIR Publ., 2017, pp. 37–56.

7. Bobillo F., Straccia U. Fuzzy Ontology Representation using OWL 2. *International Journal of Approximate Reasoning*, October, 2010, vol. 52, no. 7, pp. 1073–1094.

8. OWL 2 Web Ontology Language. Available at: <https://www.w3.org/TR/owl2-overview/> (accessed 17.02.2020).

9. *The Description Logic Handbook: Theory, Implementation, and Applications*. Edited by Baader, Franz, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. Cambridge, Cambridge University Press, 2003. 563 p.

10. Bonatti P., Tettamanzi A. Some Complexity Results on Fuzzy Description Logics. *WILF 2003, International Workshop on Fuzzy Logic and Applications*. Berlin, Springer Verlag, 2004, LNCS 2955, pp. 19–24.

11. Horrocks I., Patel-Schneider Peter F., Harmelen F. From SHIQand RDF to OWL: The Making of a Web Ontology Language. *Journal of WebSemantics*, 2003, vol. 1, no. 1, pp. 7–26.

12. Groszof B., I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. *Proc. of WWW 2003*. Budapest, Hungary, May, 2003, pp. 48–57.

13. GitHub is a Development Platform Inspired by the Way You Work. Available at: <https://github.com/> (accessed 17.02.2020).

14. Our Goal is to Build a Simple and Lightweight Set of Tools to Generate, Analyze, and Process Java Code. Available at: <https://javaparser.org/> (accessed 17.02.2020).