УДК 004.054
DOI: 10.25140/2411-5363-2017-2(8)-110-115

Ирина Богдан

ВЕРИФИКАЦИЯ МОДЕЛЕЙ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРОГРАММ: ПРОВЕРКА НА НЕПРОТИВОРЕЧИВОСТЬ И СОГЛАСОВАННОСТЬ

Актуальность темы исследования. *Качество является важнейшей характеристикой любого программного обеспечения. Для обеспечения высокого качества создаваемой программы необходимо выполнение ряда процедур, одной из основных среди которых является верификация.*

Постановка проблемы. *Верификации подлежит как само программное обеспечение, так и его модель, которая в случае с объектно-ориентированным программным обеспечением представлена множеством UML-диаграмм.*

Анализ последних исследований и публикаций. *Практически все из существующих на данный момент методов верификации моделей объектно-ориентированных программ позволяют выполнить верификацию исключительно отдельных диаграмм, не проверяя при этом всю модель в целом на непротиворечивость ее составляющих.*

Выделение нерешенных ранее частей общей проблемы. *Таким образом, актуальной является задача проверки на непротиворечивость и согласованность всех UML-диаграмм, входящих в состав создаваемого объектно-ориентированного программного обеспечения.*

Постановка задачи. *Главной целью данной статьи является описание условий и ограничений, выполнение которых позволит обеспечить согласованность и непротиворечивость между UML-диаграммами внутри модели объектно-ориентированного программного обеспечения.*

Изложение основного материала. *Так как диаграмма вариантов использования описывает требования к программному обеспечению, то непротиворечивость перехода от данной диаграммы к диаграмме классов проверить нельзя. Диаграммы взаимодействия и поведения строятся на основе диаграммы классов, поэтому необходимо проверить согласованность и непротиворечивость перехода от диаграммы классов к данным диаграммам. Диаграмма компонентов также создается на основании диаграммы классов, таким образом следует проверить корректность перехода от диаграммы классов к диаграмме компонентов, а потом – от диаграммы компонентов к диаграмме развертывания.*

Выводы. *В статье предложены условия и ограничения, проверка и выполнение которых позволит обеспечить согласованность и непротиворечивость между UML-диаграммами внутри конкретной модели объектно-ориентированного программного обеспечения.*

Ключевые слова: *объектно-ориентированное программное обеспечение; верификация; модель; согласованность; непротиворечивость; UML-диаграмма.*

Постановка проблемы. Важнейшей характеристикой любого программного обеспечения многие годы было и на сегодняшний день остается его качество. Существует несколько процедур, которые позволяют обеспечить высокое качество создаваемых программ. К таким процедурам принадлежат аудиты, ревью, инспекции на базе опросов экспертов, инспекции на базе деловых игр экспертов, отчетность, тестирование и верификация, которая является основной для обеспечения высокого качества готового программного продукта.

Верификация программного обеспечения – это приемы и методы доказательства (или опровержения) того, что программное обеспечение удовлетворяет заданной формальной спецификации [1]. В случае, если верификации подлежит объектно-ориентированное программное обеспечение, то актуальной является верификация не только самого программного обеспечения, но и его модели, представляющей собой множество UML-диаграмм: диаграммы вариантов использования, диаграммы классов, диаграммы последовательности, диаграммы кооперации, диаграммы состояний, диаграммы развертывания, диаграммы компонентов и диаграммы деятельности.

Анализ последних исследований и публикаций. На данный момент существует множество различных методов верификации моделей объектно-ориентированного программного обеспечения. Однако, практически все из них позволяют выполнить верификацию исключительно отдельных диаграмм, не проверяя при этом всю модель в целом на непротиворечивость ее составляющих.

Выделение нерешенных ранее частей общей проблемы. Таким образом, актуальной является задача проверки на непротиворечивость и согласованность всех UML-диаграмм, входящих в состав создаваемого объектно-ориентированного программного обеспечения.

Цель статьи. Главной целью данной статьи является описание условий и ограничений, выполнение которых позволит обеспечить согласованность и непротиворечивость между UML-диаграммами внутри модели объектно-ориентированного программного обеспечения.

Изложение основного материала. Первым при создании модели объектно-ориентированного программного обеспечения выполняется построение диаграммы вариантов использования. Данная диаграмма позволяет сформулировать общие требования к функциональному поведению создаваемого программного обеспечения. После чего путем анализа уже созданной и согласованной как внутри группы разработчиков, так и с заказчиком диаграммы вариантов использования создается одна или несколько диаграмм классов. Диаграмма классов служит для представления статической структуры модели создаваемого программного обеспечения (рис. 1). Однако случаи использования, отображенные на диаграмме вариантов использования, не являются представлениями программного обеспечения. Они представляют требования, которым должно соответствовать программное обеспечение. В свою очередь диаграмма классов отображает различные взаимосвязи между отдельными сущностями одной и той же предметной области, а также описывает их внутреннюю структуру и типы отношений [2]. Поэтому корректность перехода от диаграммы вариантов использования к диаграмме классов проверить невозможно.

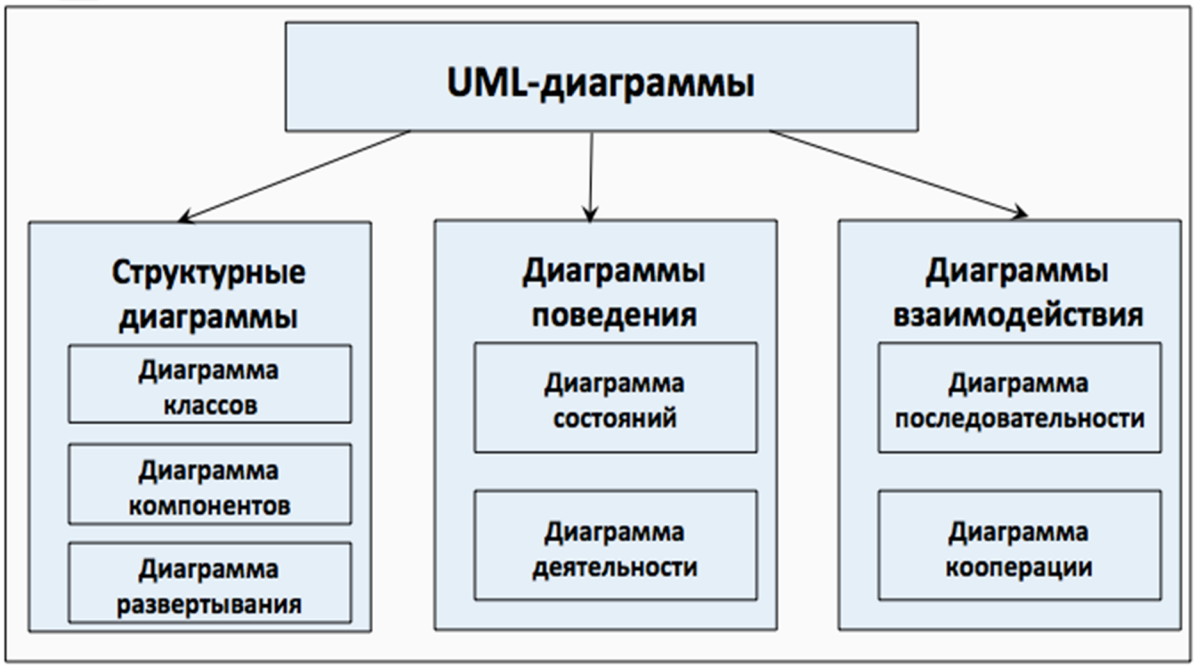

Рис. 1. Типы UML-диаграмм

После создания одной или нескольких диаграмм классов, а также на их основе происходит построение диаграмм поведения и диаграмм взаимодействия (рис. 1).

К диаграммам поведения принадлежат диаграммы состояний и диаграммы деятельности. Диаграмма состояний описывает динамическое поведение сущностей на основании описания их реакций на некоторые события. Диаграмма деятельности позволяет отобразить особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий.

К диаграммам взаимодействия принадлежат диаграммы последовательности и диаграммы кооперации. Диаграмма последовательности необходима для изучения временного аспекта поведения создаваемой программы при моделировании синхронных процессов, описывающих взаимодействие объектов. Диаграмма кооперации служит для обозначения множества взаимодействующих с определенной целью объектов и описания особенностей реализации отдельных, наиболее значимых операций в создаваемом объектно-ориентированном программном обеспечении.

Поскольку диаграмма последовательности предназначена для представления временных особенностей передачи и приема сообщений между объектами классов, описанных на диаграмме классов, то для корректного перехода от диаграммы классов к диаграмме последовательности и обеспечения их согласованности необходимо выполнение следующих условий:

− в качестве сообщений на диаграмме последовательности могут быть указаны только методы и атрибуты из соответствующей диаграммы классов. Данное условие можно проверить путем использования метода протоколов [3];

− на диаграмме могут присутствовать объекты только тех классов, которые указаны на диаграмме классов.

Диаграмма кооперации описывает структурный аспект поведения объектов. Существует две разновидности данной диаграммы: диаграмма кооперации уровня спецификации и диаграмма кооперации уровня примеров [4].

Поскольку диаграмма кооперации уровня спецификации относится к отдельному варианту использования и детализирует особенности его последующей реализации, то и проверка на согласованность с другими диаграммами для нее состоит исключительно в проверке наличия на диаграмме вариантов использования такого случая использования, который описан на диаграмме кооперации.

Так как диаграмма кооперации уровня примеров представляет собой совокупность объектов классов, описанных на диаграмме классов, и связей, также описанных на диаграмме классов в качестве методов классов, то для корректного перехода от диаграммы классов к диаграмме кооперации и обеспечения их согласованности необходимо выполнение следующих условий:

− в качестве связей на диаграмме кооперации уровня примеров могут быть указаны только методы и атрибуты из соответствующей диаграммы классов;

− на диаграмме могут присутствовать объекты только тех классов, которые указаны на диаграмме классов.

Каждая диаграмма состояний описывает возможную последовательность состояний и переходов, которые в совокупности характеризируют поведение отдельного элемента модели в течении его жизненного цикла [4]. Для корректного перехода от диаграммы классов к диаграмме состояний необходимо выполнение таких условий:

– в качестве имен действий внутри состояний на диаграмме состояний могут быть указаны только методы из соответствующей диаграммы классов;

– в качестве имен атрибутов внутри состояний могут быть указаны только атрибуты из соответствующей диаграммы классов.

Диаграмма деятельности позволяет детализировать особенности алгоритмической и логической реализации выполняемых системой операций, а также имеет такую графическую нотацию, которая похожа на графическую нотацию, используемую на диаграмме состояний. Отличие заключается в семантике состояний, которые используются для представления не деятельности, а действий, и в отсутствии на переходах сигнатуры событий [5]. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Таким образом, для корректного перехода от диаграммы классов к диаграмме деятельности необходимо выполнение таких условий:

– в качестве имен состояний на диаграмме деятельности могут быть указаны только методы из соответствующей диаграммы классов;

– в качестве имен атрибутов внутри состояний могут быть указаны только атрибуты из соответствующей диаграммы классов.

После завершения построения всех диаграмм поведения и взаимодействия необходимо убедиться, что объекты всех присутствующих на диаграмме классов классов участвуют во взаимодействии на данных диаграммах.

Диаграмма компонентов описывает особенности физического представления создаваемого объектно-ориентированного программного обеспечения. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы и обеспечить согласованный переход от логического представления к конкретной реализации проекта в виде программного кода. Основным элементом на данной диаграмме является компонент – элемент, который служит для общего обозначения элементов физического представления модели. Таким образом, данная диаграмма логически связана прежде всего с диаграммой классов и для корректного перехода от диаграммы классов к диаграмме компонентов необходимо выполнение таких условий:

– в качестве имен компонентов-рабочих продуктов на диаграмме компонентов могут быть указаны исключительно имена классов и интерфейсов с соответствующей диаграммы классов;

– количество компонентов-рабочих продуктов не должно превышать суммарное количество интерфейсов и классов на соответствующей диаграмме классов.

Помимо диаграммы компонентов для описания особенностей физического представления создаваемой программы используется также диаграмма развертывания. Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам [5]. Также данная диаграмма предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения. Таким образом, данная диаграмма логически связана прежде всего с диаграммой компонентов. Даже не смотря на то, что диаграмма развертывания является самой «демократичной» из всех диаграмм UML, так как помимо канонического представления элементов на данной диаграмме могут также присутствовать и графические примитивы, в случае, если на узлах указаны компоненты из диаграммы компонентов, их названия должны совпадать с именами компонентов на самой диаграмме компонентов.

В [6] предложена автоматизированная система верификации моделей объектно-ориентированного программного обеспечения (рис. 2), в которой для создания диаграмм используется утилита uml2 modelingtools. При создании диаграммы с помощью данной утилиты появляются два файла: непосредственно сама диаграмма, а также файл *.uml, в котором диаграмма представляется в виде XML-файла. Именно этот файл используется далее для парсинга диаграммы, так как формат XML является одним из простейших и существенно упрощает работу разработчика.

Представленные в удобном для анализа XML-формате диаграммы, составляющие модель создаваемого программного обеспечения, и используются далее для отслеживания вышеуказанных ограничений.

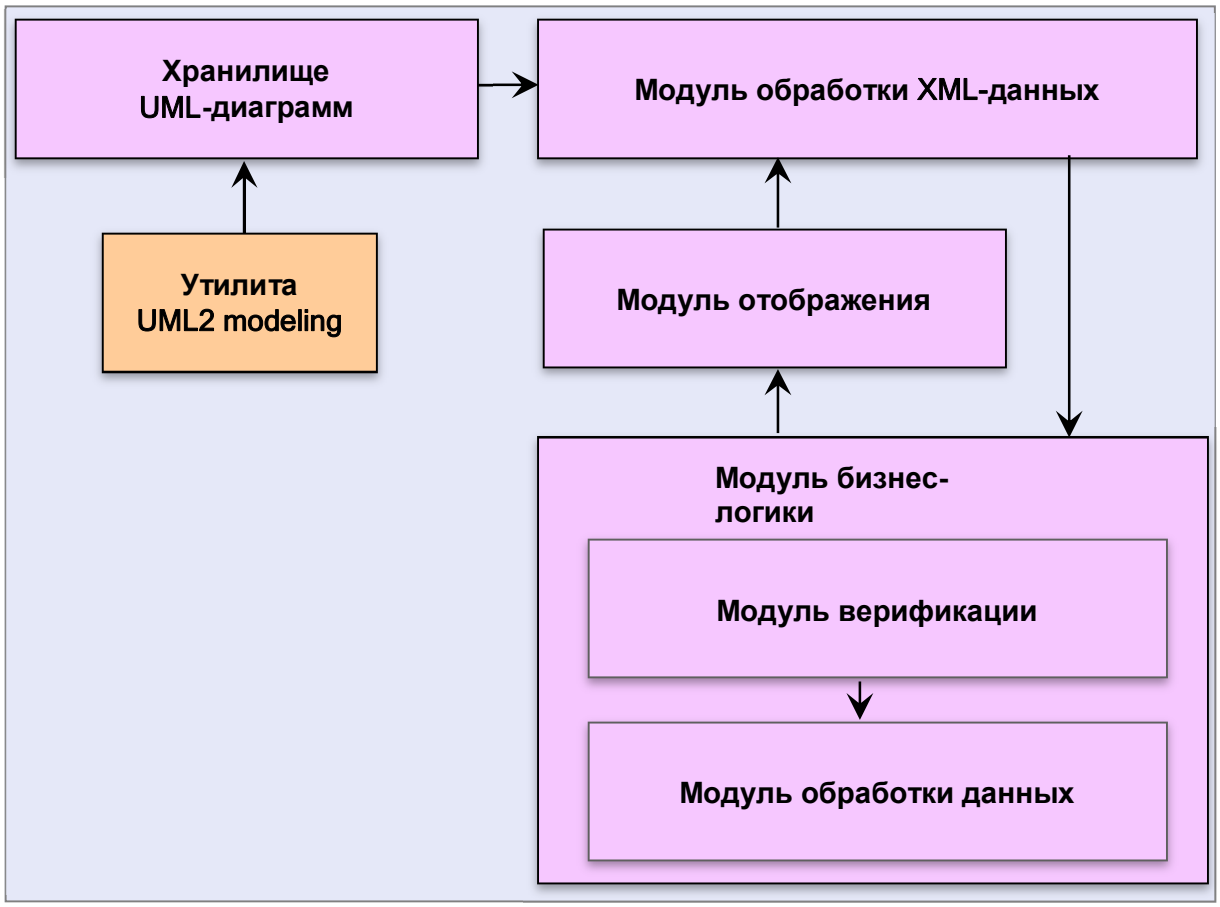

Рис. 2. Архитектура автоматизированной системы верификации моделей объектно-ориентированного программного обеспечения

Выводы и предложения. В статье предложены условия и ограничения, проверка и выполнение которых позволит обеспечить согласованность и непротиворечивость между UML-диаграммами внутри конкретной модели объектно-ориентированного программного обеспечения.

Список использованных источников

1. *OMG*. Architecture Board ORMSC // Model Driven Architecture (MDA). – ormsc/2001-07-01, 2001. –July 9. – 28 p.

2. *Синицын С. В.* Верификация программного обеспечения : учеб. пособие / С. В. Синицын, Н. Ю. Налютин. – М. : Интернет-университет информационных технологий ; БИНОМ. Лаборатория знаний, 2008. – 368 с.

3. *Макгрегор Дж.* Тестирование объектно-ориентированного программного обеспечения : практическое пособие / Дж. Макгрегор, Д. Сайкс ; пер. с англ. – К. : ООО «ТИД "ДС"», 2002. – 432 с.

4. *Леоненков А.* Самоучитель UML. Эффективный инструмент моделирования информационных систем / А. Леоненков – СПб. : BHV-Санкт-Петербург, 2001. – 304 с.

5. *Тамре Л.* Введение в тестирование программного обеспечения / Л. Тамре ; пер. с англ. – М. : Вильямс, 2003. – 368 с.

6. *Литвинов В. В.* Автоматизированная система верификации моделей объектно-ориентированного программного обеспечения / В. В. Литвинов, И. В. Богдан // Вісник Чернігівського державного технологічного університету. Серія «Технічні науки». – 2015. – № 1 (77). – С. 83–90.

References

1. OMG. Architecture Board ORMSC (2001). *Model Driven Architecture (MDA)* – ormsc/2001-07-01. July 9, 28 p.

2. Sinicyn, S.V., Nalyutin N.YU. (2008). *Verifikatsiia programmnogo obespecheniia [Programs verification]*. Moscow: Internet-universitet informatsionnykh tekhnologii; BINOM. Laboratoriia znanii (in Russian).

3. Makgregor, Dzh., Sajks, D. (2002). *Testirovanie obektno-orientirovannogo programmnogo obespecheniia [Testing of object-oriented programs]*. Kyiv: OOO "TID «DS»" (in Russian).

4. Leonenkov, A. (2001) Samouchitel UML. Effektivnyi instrument modelirovaniia informatsionnykh sistem [UML Tutorial. An effective tool for modeling of information systems]. Saint-Petersburg: BHV-Sankt-Peterburg (in Russian).

5. Tamre, L. (2003) *Vvedenie v testirovanie programmnogo obespecheniya [Introduction to Software Testing]*. Moscow: Viliams (in Russian).

6. Litvinov, V.V., Bogdan, I.V. (2015). Avtomatizirovannaia sistema verifikatsii modelei obektno-orientirovannogo programmnogo obespecheniia [The automated system of verification of the object-oriented software models]. *Visnik Chernihivskoho derzhavnoho tekhnolohichnoho universitetu. Seriia «Tekhnichni nauki» – Visnyk of Chernihiv State Technological University. Series "Technical sciences"*, vol. 1 (77), pp. 83–90 (in Russian).

Iryna Bohdan

VERIFICATION OF MODELS OF OBJECT-ORIENTED PROGRAMS: CHECKING FOR NON-OPINIONITY AND CONSISTENCY

Urgency of the research. *Quality is an essential characteristic of any software. To ensure the high quality of the created program, it is necessary to perform a number of procedures, one of the main among which is verification.*

Target setting. *Verification is performed to both the software itself and its model, which in the case of object-oriented software is represented by a variety of UML diagrams.*

Actual scientific researches and issues analysis. *Nearly all of the existing methods of verification of models of object-oriented programs allow you to verify only individual diagrams without checking the entire model as a whole for the consistency of its components.*

Uninvestigated parts of general matters defining. *Accordingly, the problem of checking for consistency of all UML diagrams that are a part of the created object-oriented software is actual.*

The research objective. *The main purpose of this article is to describe the conditions and constraints, the implementation of which will ensure the consistency between UML diagrams within the object-oriented software model.*

The statement of basic materials. *Since the use case diagram describes the software requirements, the consistency of the transition from a given diagram to a class diagram can not be verified. Diagrams of interaction and behavior are built on the basis of a class diagram, so it is important to check the consistency of the transition from the class diagram to these diagrams. The component diagram is also created based on the class diagram, so it is compulsory to check the correctness of the transition from the class diagram to the component diagram, and then - from the component diagram to the deployment diagram.*

Conclusions. *The article represents conditions and limitations, the verification and the implementation of which will ensure consistency between UML diagrams within a particular model of object-oriented software.*

Key words: *diagram; verification; model.*

Ірина Богдан

ВЕРИФІКАЦІЯ МОДЕЛЕЙ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ПРОГРАМ: ПЕРЕВІРКА НА НЕСУПЕРЕЧЛИВІСТЬ ТА УЗГОДЖЕНІСТЬ

Верифікація є однією з основних процедур забезпечення якості програмного забезпечення. У випадку з об'єктно-орієнтованим програмним забезпеченням верифікації підлягає як сама програма, так і її модель, що представлена множиною UML-діаграм. Існує досить багато різних методів верифікації UML-діаграм, однак жоден з них не перевіряє на несуперечливість та узгодженість діаграми, що входять до складу тієї ж самої моделі.

Стаття присвячена опису умов та обмежень, виконання яких дозволить забезпечити узгодженість та несуперечливість між UML-діаграмами всередині моделі об'єктно-орієнтованого програмного забезпечення.

Ключові слова: *об'єктно-орієнтоване програмне забезпечення; верифікація; модель; узгодженість; несуперечливість; UML-діаграма.*

Богдан Ирина Валентиновна – кандидат технических наук, доцент кафедры информационных технологий и программной инженерии, Черниговский национальный технологический университет (ул. Шевченко, 95, г. Чернигов, 14027, Украина).

Богдан Ірина Валентинівна – кандидат технічних наук, доцент кафедри інформаційних технологій та програмної інженерії, Чернігівський національний технологічний університет (вул. Шевченко, 95, м. Чернігів, 14027, Україна**).**

Bohdan Iryna – PhD in Technical Sciences, Associate Professor of Department of Information Technologies and Software Engineering, Chernihiv National University of Technology (95 Shevchenka Str., 14027 Chernihiv, Ukraine).

E-mail: irakirienko@gmail.com

ORCID: http://orcid.org/0000-0003-1521-6958

Богдан И. Верификация моделей объектно-ориентированных программ: проверка на непротиворечивость и согласованность / И. Богдан // Технічні науки та технології. – 2017. – № 2 (8). – С. 110-115.

УДК 004.42:004.75
DOI: 10.25140/2411-5363-2017-2(8)-116-122

Сергей Точилин

ПРОИЗВОДИТЕЛЬНОСТЬ RESTful И SOAP PHP WEB-СЕРВИСОВ ПРИ ПОИСКЕ В ДАННЫХ MySQL

Актуальность темы исследования. *При построении распределенных компьютерных систем широко используется сервис-ориентированная архитектура (СОА). Для практической реализации систем с СОА применяют технологии Web-сервисов. Одним из основных требований предъявляемых к Web-сервисам является производительность функционирования, которая зависит от их программной реализации.*

Постановка проблемы. *Web-сервисы используются для доступа к информации, которая хранится в базах данных СУБД, размещенных на Web-узлах.*

В этой связи актуальной является проблема выбора оптимального программного обеспечения для реализации информационного Web-сервиса с оперативным доступом к данным.

Анализ последних исследований и публикаций. *При доступе к данным Web-сервисы используют запросы к СУБД, которые, как правило, реализуют в виде команд SQL.*

Для повышения эффективности работы Web-приложений, которые работают с данными, вместо команд SQL применяют хранимые процедуры и функции СУБД.

Выделение неисследованных частей общей проблемы. *Исследований производительности RESTful и SOAP PHP Web-сервисов, которые при доступе к данным MySQL с различным объемом хранимой информации в одном режиме функционирования применяли команды SQL, в другом – хранимые процедуры, не проводилось.*

Постановка задачи. *Определить и проанализировать производительность RESTful и SOAP PHP Web-сервисов, которые используют команды SQL и функционально подобные хранимые процедуры при поиске в данных MySQL, для выбора оптимального программного обеспечения Web-служб, работающих с СУБД MySQL.*

Изложение основного материала. *Как выяснилось, использование RESTful и SOAP PHP Web-сервисами при запросах к MySQL на поиск в данных хранимых процедур, вместо команд SQL, снижало их производительность.*

В тоже время, при поиске в исследованных объемах данных $V \le 2$ *МБайт с применением в запросах к MySQL, как команд SQL, так и хранимых процедур, RESTful Web-сервис был более производительным, чем SOAP PHP сервис.*

Выводы статьи. *Установлены особенности производительности RESTful и SOAP PHP сервисов при поиске в данных MySQL с использованием, как команд SQL, так и хранимых процедур.*

При поиске в данных MySQL RESTful Web-сервис во всех режимах функционирования был более производительный чем SOAP PHP сервис.

Ключевые слова: *производительность; SOAP; RESTful; Web-сервис; MySQL.*

Постановка проблемы. При построении современных распределенных компьютерных систем широко используется сервис-ориентированная архитектура (СОА). Для практической реализации систем с СОА применяют технологии Web-сервисов (Web-служб) [1–3].

Web-сервисы предоставляют Web-услуги в рамках слабосвязанных Web-приложений. К таким услугам, в частности, относятся услуги доступа к информации, которая хранится в базах данных (БД), созданных и поддерживаемых определенной системой управления БД (СУБД). При этом для доступа к данным используют запросы к соответствующей СУБД, которые, как правило, реализуют на структурированном языке запросов (Structured Query Language – SQL) в виде команд SQL.

В тоже время для повышения эффективности работы Web-приложений, которые работают с данными, вместо команд SQL применяют хранимые процедуры и функции СУБД [4].

В соответствии с [4] их использование, в частности, позволяет:

- Обеспечить дополнительную защиту данных СУБД.
- Минимизировать изменения кода приложения при изменении структуры БД.
- Уменьшить сетевой трафик, обусловленный обменом информации с сервером СУБД.
- Инкапсулировать логику работы с данными.
- Повысить транспортабельность приложений для работы с СУБД.

Однако при использовании хранимых процедур и функций может измениться производительность Web-приложений, работающих с информацией.