

УНИФИЦИРОВАННЫЙ ДОСТУП К УНИВЕРСАЛЬНОМУ ХРАНИЛИЩУ ДАННЫХ

© 2016 Д. Е. Яблоков

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва (национальный исследовательский университет)

В статье рассматривается возможность применения обобщённых концепций, основанных на применении нескольких парадигм программирования, для организации унифицированного доступа к универсальному хранилищу данных. Исследование проводится на примере разработки библиотеки программных компонентов в рамках проекта создания экспертной системы с использованием информационной базы программного комплекса ToposPro. Целью статьи является проверка возможности формирования механизмов для конструирования самостоятельных компонентов доступа и обработки данных, позволяющих создавать программные продукты с максимальной независимостью частей программы. При этом эти части должны быть довольно сильно связаны на уровне внутренних отношений компонентов, чтобы конкретная реализация отличалась идейной целостностью и обеспечивала лучшее понимание в процессе проектирования. Структуры хранения и средства доступа к данным должны быть связаны с тем, что рассматривается в статье как обобщённые концепции адаптеров контейнерных классов и итераторов. Это позволяет представлять код программы в виде набора элементарных независимых примитивов, дающих основу для проектирования логики приложений в терминах явно выраженных обобщённых абстракций, предусматривающих процесс расширения компонентов библиотеки и их адаптации к понятиям предметной области. Подход с использованием нескольких парадигм программирования в сочетании с возможностью расширения и настройки вновь создаваемых компонентов на базе библиотечных абстракций обеспечивает новый уровень осмысления программирования. Главными инструментами такого подхода становятся обобщённые концепции, а реализации конкретных классов на их основе могут настраиваться за счёт перегрузки, агрегирования, наследования и спецификации требований к абстрактному типу данных. После предварительной реализации некоторых библиотечных компонентов, а также конкретных классов доступа к универсальной модели данных можно сделать вывод о том, что направленные взаимодействия с универсальным хранилищем на основе обобщённых концепций выбрано правильно. К основным достоинствам выбранной стратегии относятся: уменьшение уровня сложности, сокращение количества кода и возможность сосредоточения разработчика только на бизнес-логике приложений, осуществляющих доступ к базам данных, поддерживающим универсальную модель хранения.

Универсальная модель данных, ToposPro, парадигмы программирования, развитие концепций, абстракция данных, обобщённые концепции итераторов и адаптеров контейнерных классов.

Быстрый рост направления компьютерных наук оказал влияние и на современное состояние индустрии программирования, которую сейчас нельзя представить без качественных технологий построения абстракций и базовых принципов, дающих основу соответствующему стилю проектирования, а также инструментальных средств, поддерживающихся используемыми парадигмами [1; 2], которые определяют стратегию конструирования программного обеспечения. Обду-

манное и правильное использование подобного рода механизмов и связанной с ними терминологии способствует формированию каркаса понятий, определяющего логические рамки для описания объектов или их семейств, работа с которыми становится возможной в терминах используемого понятийного аппарата. Это приводит к необходимости формирования набора абстракций, определяющих семантическое и синтаксическое поведение объектов, то есть к появлению семейства

обобщённых концепций [3], содержащих набор требований и дающих представление о свойствах и ограничениях, которыми должна обладать абстракция, являющаяся моделью одной или нескольких концепций.

Под моделированием будем понимать процесс, определяющий отношение между концепцией и представляющей её семантические и синтаксические свойства абстракцией. Отношения между концепциями формализуются процессом развития одной концепции относительно другой, когда развивающая концепция предоставляет всю функциональность развиваемой и, возможно, некоторую дополнительную, являющуюся только частью развиваемой. Абстракция может быть моделью более чем одной концепции, а последняя, в свою очередь, может быть развитием нескольких концепций. Практически это означает, что если некоторый алгоритм требует, чтобы его формальные параметры являлись моделью какой-либо концепции, то его фактические параметры могут представлять собой тип, являющийся моделью другой концепции, при условии, что она развивает концепцию, определяющую набор требований к формальным параметрам. В этих терминах можно определить базовые принципы объектно-ориентированной и обобщённой парадигм программирования [4], такие как наследование, полиморфизм и спецификация требований к типу или их семейству [5].

Обобщённые концепции – это мощный инструмент решения задач, связанных с данными, дающий возможность как определения поведения и отношения между дискретными объектами и их семействами, так и применяемый для описания синтаксических и семантических особенностей компонентов, предназначенных для унифицированной обработки и представления большого объёма информации. Немаловажным фактором является необходимость использования соответствующих парадигм и проверенных временем идиом, основанных на опыте и твёрдом понимании выбранного языка

программирования и объединении воедино наилучших практических решений, рекомендаций и правил, определяющих стандартизированный способ получения на выходе наиболее качественного результата. Одним из примеров реализации такого стандартизированного набора рекомендаций и правил является разрабатываемая в рамках проекта создания экспертной системы обобщённая библиотека расширений (GLEX), разрабатываемая в рамках проекта создания экспертной системы на базе информационной среды программного комплекса ToposPro [6]. Теоретическую основу GLEX составляет набор базовых, неявно заданных, концепций и требований к типам данных стандартной библиотеки шаблонов (STL) [7; 8], широко используемой при программировании на языке C++. В дальнейшем планируется широкое применение GLEX при реализации компонентов обработки и представления универсальных данных большого объёма. GLEX – это не просто библиотека, а скорее набор соглашений, объединяющих компоненты нескольких типов: адаптеры контейнеров, алгоритмы, концепции итераторов и функциональных адаптеров. Идея состоит в том, что адаптеры контейнеров и алгоритмы, работающие с ними, могут ничего не знать друг о друге. Это преимущество достигается за счет обобщённых концепций итераторов.

Обобщённые концепции итераторов (рис. 1–4) важны при проектировании библиотечных компонентов обработки и анализа данных, так как абстракции, построенные на базе этих понятий, могут являться интерфейсами между обобщёнными алгоритмами и адаптерами структур хранения данных. Они также имеют большое значение, поскольку являются обобщением указателей, т.е. объектов, которые указывают на другие объекты и могут использоваться как основа для создания компонентов, осуществляющих проход по диапазону. Если модель концепции итератора указывает на какую-либо позицию диапазона, то после применения опе-

рации продвижения (инкрементирование или декрементирование) она будет указывать позицию, являющуюся следующей или предыдущей в зависимости от направления обхода. Нужно отметить, что взаимосвязь концепций итераторов со свойствами указателей не совсем однозначна. Например, указатели в языке C++ имеют очень развитую семантику и к ним применимы операции адресной арифме-

тики [7], операция разыменования и т.п., но обобщённые алгоритмы на диапазонах, в большинстве случаев, используют лишь малое подмножество свойств указателей, другие же обобщённые алгоритмы используют иные подмножества. Таким образом, существует несколько различных способов обобщения семантики указателей, при этом каждый из способов является отдельной обобщённой концепцией.

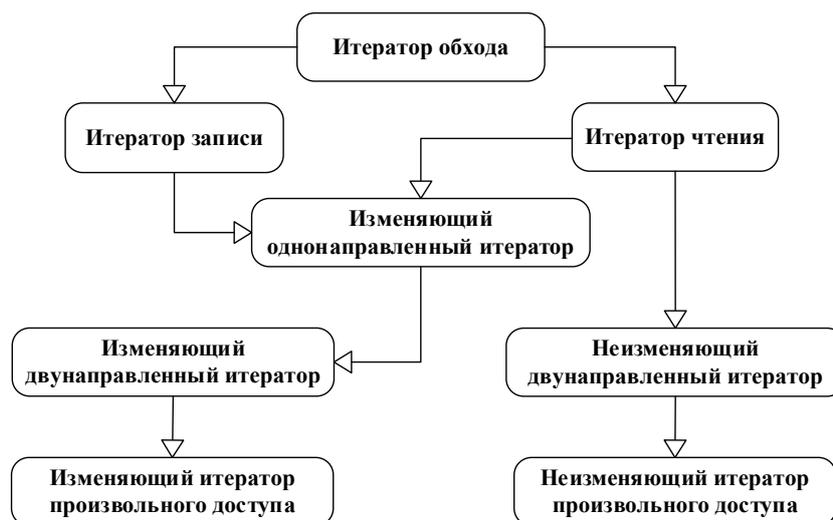


Рис. 1. Развитие концепций итераторов

```

public interface
ITraversalIterator<I extends ITraversalIterator<I>>
extends Cloneable
{
    void advance();
    I getIterator();
};
  
```

Рис. 2. Модель концепции итератора обхода

```

public interface
IReadableIterator<I extends IReadableIterator<I, E>, E>
extends ITraversalIterator<I>
{
    boolean equalTo(I other);
    IReadableIterator<I, E> clone();
    E get();
};
  
```

Рис. 3. Модель концепции итератора чтения

```

public interface
IMutableForwardIterator<I extends IMutableForwardIterator<I, E>, E>
extends IReadableIterator<I, E>, IWritableIterator<I, E>
{
    IMutableSinglePassIterator<I, E> clone();
};
  
```

Рис. 4. Модель концепции изменяющего однонаправленного итератора

Понятие адаптирующей концепции итератора (рис. 5–11) вводится для того, чтобы изменить поведение одной из уже имеющихся обобщённых концепций итераторов. Для этого необходимо создать адаптивный интерфейс, который, в определённом смысле, будет изменять поведение адаптирующей модели в терминах адаптируемой концепции. Важным аспектом применения адаптирующей модели является тот факт, что становится возможным сделать что-то вроде того, что было реализовано в качестве адаптируе-

мой концепции, но с конкретным альтернативным поведением, например, изменяющим направление обхода диапазона или контекст отдельных операций. В общем случае это гарантирует, что обобщённые алгоритмы на диапазонах [7] будут правильно и эффективно работать с адаптируемыми моделями итераторов, построенными на базе обобщённых концепций со строгими требованиями к семантике и формализованными интерфейсами.



Рис. 5. Концепции итераторов вставки

```

public interface
IWritableIterator<I extends IWritableIterator <I, E>, E>
extends ITraversalIterator<I>
{
    IWritableIterator <I, E> clone();
    void put(E element);
};
  
```

Рис. 6. Модель концепции итератора записи

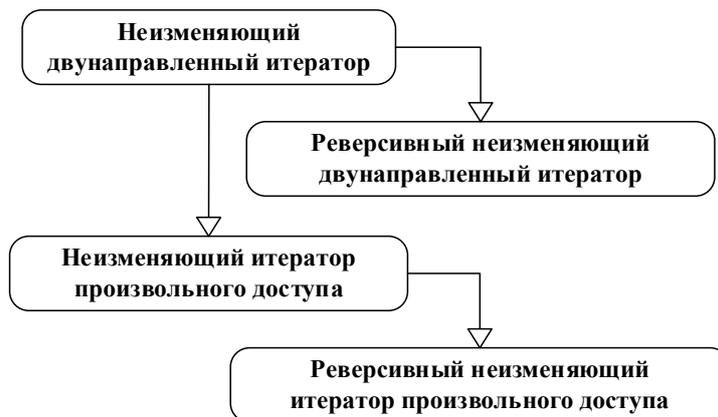


Рис. 7. Концепции реверсивных неизменяющих итераторов

```

public interface
IReadableBidirectionalIterator<I extends IReadableBidirectionalIterator<I, E>, E>
extends IReadableIterator<I, E>
{
    IReadableBidirectionalIterator<I, E> clone();
    void retreat();
};

```

Рис. 8. Модель концепции неизменяющего двунаправленного итератора

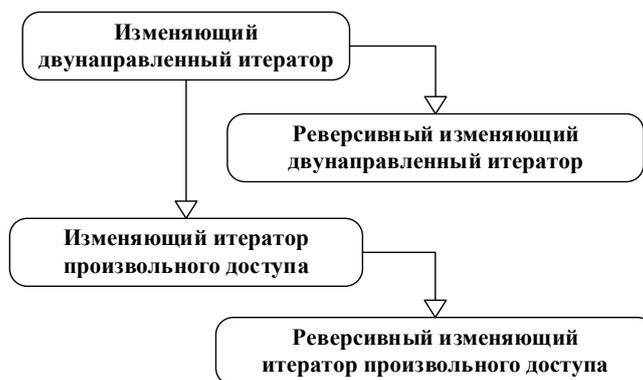


Рис. 9. Концепции реверсивных изменяющих итераторов

```

public interface
IMutableRandomAccessIterator<I extends
IMutableRandomAccessIterator<I, D, E>, D extends Number, E>
extends IMutableBidirectionalIterator<I, E>
{
    IMutableRandomAccessIterator<I, D, E> clone();
    D position();
    E get(D position);
    void advance(D distance);
    void retreat(D distance);
    void put(D distance, E element);
    D distance(I other);
    boolean lessThan(I other);
};

```

Рис. 10. Модель концепции изменяющего итератора произвольного доступа

```

public interface
IMutableReverseRandomAccessIterator<I extends
IMutableReverseRandomAccessIterator<I, D, E>, D extends Number, E>
extends IMutableRandomAccessIterator<I, D, E>
{
    IMutableReverseRandomAccessIterator<I, D, E>
clone();
    IMutableRandomAccessIterator<I, D, E> base();
};

```

Рис. 11. Модель концепции изменяющего реверсивного итератора произвольного доступа

Обобщённые концепции адаптеров контейнерных классов (рис. 12 – 16) – это механизм адаптации коллекций, т.е. хранилищ, поддерживающих различные способы накопления и упорядочения объек-

тов. Цель таких адаптеров – формализация в терминах базовых абстракций наиболее часто употребляемых операций добавления нового элемента, удаления элемента или доступа к элементу с целью

обеспечения наиболее эффективной работы с уже имеющимися структурами хранения [5]. Организация способа хранения данных для их последующей обработки является весьма важным этапом разработки компьютерных программ. Для реализации большинства приложений выбор соответствующей абстрактной структуры хранения и доступа к данным является

единственно важным решением, т.к. для одних и тех же обобщённых алгоритмов различные структуры данных и средства доступа к ним обладают неодинаковой эффективностью. Можно выделить два основных направления развития концепций адаптеров контейнерных классов. Это – диапазон и последовательность.

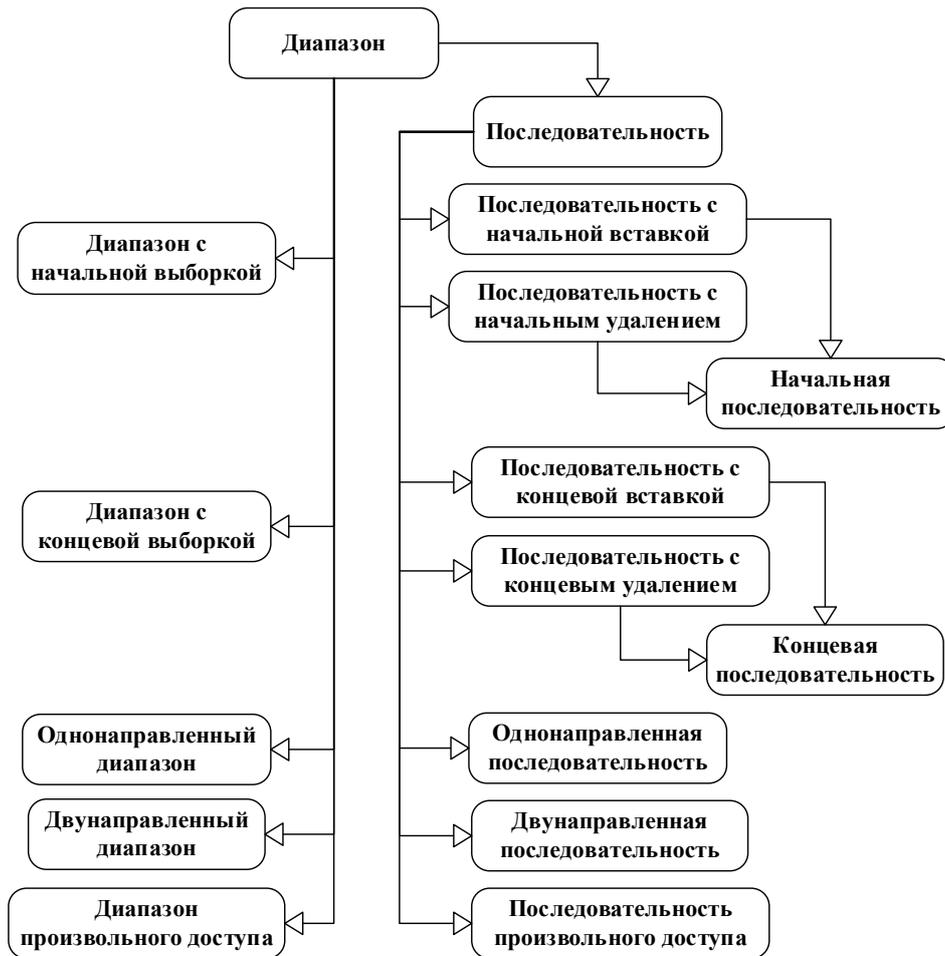


Рис. 12. Развитие концепций адаптеров контейнерных классов

```
public interface
ITraversalRange<R extends ITraversalRange<R, S>, S extends Number>
extends Cloneable
{
    boolean equalTo(R other);
    boolean lessThan(R other);
    boolean empty();
    R getRange();
    S size();
};
```

Рис. 13. Модель концепции диапазона

```
public interface
IFrontRange<R extends IFrontRange<R, S, E>, S extends Number, E>
extends ITraversalRange<R, S>
{
    IFrontRange<R, S, E> clone();
    E front();
};
```

Рис. 14. Модель концепции диапазона с начальной выборкой

```
public interface
IRangeSequence<C extends IRangeSequence<C, S>, S extends Number>
extends ITraversalRange<C, S>
{
    S maxSize();
    S capacity();
    void clear();
};
```

Рис. 15. Модель концепции последовательности

```
public interface
IBackInsertionSequence<C extends
IBackInsertionSequence<C, S, E>, S extends Number, E>
extends IRangeSequence<C, S>
{
    void pushBack(E element);
    IBackInsertionSequence<C, S, E> clone();
};
```

Рис. 16. Модель концепции последовательности с концевой вставкой



Рис. 17. Модель концепции структурного вида секции плоского списка

Считается, что наиболее действенным способом борьбы со сложностью программных продуктов является попытка построения соответствующего уровня абстракции, основанного на анализе общности и изменчивости объектов предметной области [5; 7]. Например, возможность использования иерархии полиморфных типов данных, связанных отношением наследования, а также наборов абстрактных требований к типам, модели-

рующим интерфейс и семантическое поведение, даст существенное преимущество при конструировании компонентов и подсистем работы с различными наборами данных, таких как коллекция строк, иерархия строк и т.п. (рис. 17). Это, в свою очередь, позволит стандартизировать процесс извлечения и обработки данных из универсального хранилища (рис. 18).



Рис. 18. Извлечение и обработка данных из универсального хранилища

Предварительная реализация части компонентов библиотеки GLEX показала, что выбранный подход, предполагающий использование обобщённых концепций в качестве базовых абстракций для компонентов доступа и обработки данных, является наиболее эффективным с точки зрения снижения уровня сложности и уменьшения количества кода при автоматизации решений задач, часто встречающихся в процессе научного исследования.

Автор выражает глубокое признание и благодарность профессору В.А. Блатову и профессору В.А. Салееву за обсуждение полученных результатов.

Представленная в статье работа выполняется при поддержке Министерства образования и науки Российской Федерации, Мегагрант № 14.25.31.0005.

Библиографический список

1. Blatov V.A., Shevchenko A.P., Proserpio D.M. Applied Topological Analysis of Crystal Structures with the Program Package ToposPro // Crystal Growth and Design. 2014. V. 14, Iss. 7. P. 3576-3586. DOI: 10.1021/cg500498k
2. Simmonds D.M. The Programming Paradigm Evolution // IEEE Computer. 2012. V. 45, Iss. 6. P. 93-95. DOI: 10.1109/mc.2012.219
3. Блинов И.Н., Романчик В.С. Java. Промышленное программирование. Минск: УниверсалПресс, 2007. 704 с.
4. Вандевурд Д., Джосаттис Н.М. Шаблоны C++: справочник разработчика. М.: Вильямс, 2003. 544 с.
5. Страуструп Б. Язык программирования C++. Специальное издание. М.: Бином, 2011. 1136 с.
6. Яблоков Д.Е. Применение парадигмы обобщённого программирования в объектно-ориентированных языках // Сборник научных трудов V Всероссийской школы-семинара аспирантов, студентов и молодых учёных «Информатика, моделирование, автоматизация проектирования». Ульяновск: УлГТУ, 2013. С. 216-221.

7. Яблоков Д.Е. Парадигмы программирования // Ежемесячный научный журнал «Prospero». 2015. № 2(14). С. 94-98.

8. Яблоков Д.Е. Использование обобщённых концепций в объектно-ориентированных языках программирования // Труды международной научно-технической конференции «Перспективные информационные технологии (ПИТ 2015)». Т. 2. Самара: Самарский научный центр РАН, 2015. С. 341-345.

Информация об авторе

Яблоков Денис Евгеньевич, ведущий инженер-программист, Межвузовский научно-исследовательский центр по теоретическому материаловедению, Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет). E-mail: dyablokov@gmail.com. Область научных интересов: парадигмы программирования, объектно-ориентированное программирование, обобщённое программирование, мультипарадигменное проектирование, применение паттернов проектирования, анализ и классификация данных.

UNIFORM ACCESS TO UNIVERSAL DATA STORAGE

© 2016 D. E. Yablokov

Samara State Aerospace University, Samara, Russian Federation

The paper deals with the possibility of application of generic concepts based on the use of multiple programming paradigms with the aim of providing uniform access to universal data storage. The investigation is conducted using the example of developing a library of software components within the framework of the project of creating an expert system using the information base of the Topos Pro software package. The aim of the paper is to check the possibility of forming mechanisms for construction of independent components for access and data processing that would make it possible to produce software products with a low level of coupling and high cohesion. In this case, the concrete implementation would be characterized by conceptual integrity and would provide better understanding of the design process. Storage structures and data access facilities should be associated with what is considered in the article as generic concepts of container class adapters and iterators. This makes it possible to present the program code as a set of elementary independent primitives forming the basis for application logic design in terms of generic abstractions implying the process of extending library components and their adaptation to the domain area. The approach of using multiple programming paradigms coupled with the possibility of extension and setting of newly-created components based on library abstractions provides a new level of understanding programming. The generic concepts in this case become the main tools, while implementations of specific classes on their basis can be set through the use of overloading, aggregation, inheritance and specification of requirements for an abstract data type. After the pre-implementation of some library components and specific classes of access to a universal data model a conclusion can be made that the chosen strategy of interaction with the universal storage based on generic concepts is the best. It makes possible to reduce the complexity level as well as the LOC and allows the developer to concentrate only on the domain logic of applications providing access to the databases supporting the universal data model.

Universal data model, ToposPro, programming paradigms, refinement of concepts, data abstraction, generic concepts of iterators and container class adapters.

References

1. Blatov V.A., Shevchenko A.P., Proserpio D.M. Applied Topological Analysis of Crystal Structures with the Program Package ToposPro. *Crystal Growth and Design*. 2014. V. 14, Iss. 7. P. 3576-3586. DOI: 10.1021/cg500498k
2. Simmonds D.M. The Programming Paradigm Evolution. *IEEE Computer*. 2012. V. 45, Iss. 6. P. 93-95. DOI: 10.1109/mc.2012.219
3. Blinov I.N., Romanchik V.S. *Java. Promyshlennoe programmirovaniye* [Java. Industrial programming]. Minsk: UniversalPress Publ., 2007. 704 p.
4. Vandevurd D., Dzhosattis N.M. *Shablony C++: spravochnik razrabotchika* [Templates in C++: developer's reference book]. Moscow: Vil'yams Publ., 2003. 544 p.
5. Straustrup B. *Yazyk programmirovaniya C++. Spetsial'noe izdanie* [The C++ Programming Language. Special edition]. Moscow: Binom Publ., 2011. 912 p.
6. Yablokov D.E. Primeneniye paradigmy obobshchennogo programmirovaniya v ob"ektno-orientirovannykh yazykakh. *Sbornik nauchnykh trudov V Vserossiyskoy shkoly-seminara aspirantov, studentov I molodykh uchenykh «Informatika, modelirovaniye, avtomatizatsiya proektirovaniya»*. Ulyanovsk: Ulyanovsk State Technical University Publ., 2013. P. 216-221. (In Russ.)
7. Yablokov D.E. The paradigms of programming. *Ezhemesyachnyy nauchnyy zhurnal «Prospero»*. 2015. No. 2(14). P. 94-98. (In Russ.)
8. Yablokov D.E. Ispol'zovaniye obobshchennykh kontseptsiy v ob"ektno-orientirovannykh yazykakh programmirovaniya. *International Scientific Conference Proceedings «Advanced Information Technologies and Scientific Computing»*. V. 2. Samara: SamarSKIY nauchnyy tsentr RAN Publ., 2015. P. 341-345. (In Russ.)

About the author

Yablokov Denis Evgenyevich, Principal Software Engineer, Samara Research Center for Theoretical Materials Science, Samara State Aerospace University, Samara, Russian Federation. E-mail: dyablokov@gmail.com. Area of Research: programming paradigms, object-oriented programming, generic programming, multi-paradigm design, using design patterns, data analysis and classification.